A guide to modeling reaction-diffusion of molecules with the E-Cell System

Satya N. V. Arjunan
satya@riken.jp
RIKEN Quantitative Biology Center, Furuedai, Suita, Osaka 565-0874, Japan

Abstract
The E-Cell System is an advanced platform intended for mathematical modeling and simulation of well-stirred biochemical systems. We have recently implemented the Spatiocyte method as a set of plug in modules to the E-Cell System, allowing simulations of complicated multicompartment dynamical processes with inhomogeneous molecular distributions. With Spatiocyte, the diffusion and reaction of each molecule can be handled individually at the microscopic scale. Here we describe the basic theory of the method and provide the installation and usage guides of the Spatiocyte modules. Where possible, model examples are also given to quickly familiarize the reader with spatiotemporal model building and simulation.
Keywords: spatial modeling, stochastic simulation, diffusion, membrane, multicompartment, intercompartment, Spatiocyte

Introduction

The E-Cell System version 3 can model and simulate both deterministic and stochastic biochemical processes (Takakashi et al., 2004). Simulated molecules are assumed to be dimensionless and homogeneously distributed in a compartment. Some processes such as cell signaling and cytokinesis, however, depend on cellular geometry and spatially localized molecules to carry out their functions. To reproduce such processes using spatially resolved models in silico, we have developed a lattice-based stochastic reaction-diffuson (RD) simulation method, called Spatiocyte (Arjunan and Tomita, 2010), and implemented it as a set of plug in modules to the E-Cell System (Arjunan and Tomita, 2009). Spatiocyte allows diffusion and reaction to take place between different compartments: for example, a volume molecule in the cytoplasm can diffuse and react with a surface molecule on the plasma membrane. Since molecules are represented as spheres with dimensions, it can also reproduce anomalous diffusion of molecules in a crowded compartment (Dix and Verkman, 2008, Hall and Hoshino, 2010). Using Spatiocyte simulated microscopy visualization feature, simulation results of spatiotemporal localization of molecules can be evaluated by directly comparing them with experimentally obtained fluorescent microscopy images.

The theory and algorithm of the Spatiocyte method are provided in (Arjunan and Tomita, 2010) while the implementation details are described in (Arjunan and Tomita, 2009). In this chapter, we provide a guide on how to build spatiotemporal RD models using Spatiocyte modules. We begin with the basic theory of the method and proceed with the installation procedures. The properties of each module are outlined in the subsequent section. Some example models are given to familiarize the reader with the common model

structures while describing the modules. We conclude this chapter by outlining the planned future directions of Spatiocyte development.

Spatiocyte Method

In this section, we summarize the underlying features of the Spatiocyte method that are necessary to build an RD model. For a more detailed description of the method we direct the reader to a previous article (Arjunan and Tomita, 2010).

The Spatiocyte method discretizes the space into a hexagonal close-packed (HCP) lattice of regular sphere voxels with radius r_v. Each voxel has 12 adjoining neighbors. To represent a surface compartment such as a cell or a nuclear membrane, all empty voxels of the compartment are occupied with immobile lipid molecules. The method also allows molecules to be simulated at microscopic and compartmental spatial scales simultaneously. In the former, each molecule is discrete and treated individually. For example, each diffusing molecule at the microscopic scale is moved independently by a *DiffusionProcess* from a source voxel to a target neighbor voxel after a given diffusion step interval. Immobile molecules are also simulated at the microscopic scale. Conversely at the compartmental scale, molecules are assumed to be homogeneously distributed (HD) and thus, the concentration information of each HD species is sufficient without explicit diffusion movements. Depending on the simulated spatial scale and the mobility of the reacting species, molecules can undergo either diffusion-influenced or diffusion-decoupled reactions.

All second-order reactions comprising two diffusing reactants, or a diffusing and an immobile reactant are diffusion-influenced, and are therefore, executed by the *DiffusionInfluencedReactionProcess*. The remaining reactions, which include all zeroth- and first-order reactions, and second-order reactions that involve two adjoining immobile reactants or at least one HD reactant, can be decoupled from diffusion. These diffusion-decoupled reactions are performed by the *SpatiocyteNextReactionProcess*.

We proceed with the execution of *DiffusionInfluencedReactionProcess* for a reaction *j*. Following our discretized scheme (Arjunan and Tomita, 2010) of the Collins and Kimball RD approach (Collins and Kimball, 1949), when a diffusing molecule collides with a reactant pair of *j* at the target voxel, they react with probability

$$p_j = \begin{cases} \frac{k_{AB}}{6\sqrt{2}(D_A+D_B)r_v}, & A_v + B_v \xrightarrow{k_{AB}} \text{product(s)}, \\ \frac{k_{AA}}{6\sqrt{2}D_A r_v}, & A_v + A_v \xrightarrow{k_{AA}} \text{product(s)}, \\ \frac{\gamma k_{AB}}{D_A+D_B}, & A_s + B_s \xrightarrow{k_{AB}} \text{product(s)}, \\ \frac{\gamma k_{AA}}{D_A}, & A_s + A_s \xrightarrow{k_{AA}} \text{product(s)}, \\ \frac{\sqrt{2}k_{AB}}{3D_A r_v}, & A_v + B_s \xrightarrow{k_{AB}} \text{product(s)}, \\ \frac{24k_S r_v}{(6+3\sqrt{3}+2\sqrt{6})D_A}, & A_v(+L_s) \xrightarrow{k_S} \text{product(s)}, \end{cases}$$

where the constant $\gamma = \frac{(2\sqrt{2}+4\sqrt{3}+3\sqrt{6}+\sqrt{22})^2}{72(6\sqrt{2}+4\sqrt{3}+3\sqrt{6})}$, *L* is the lipid species, *k* is the intrinsic reaction rate of *j*, *D* is the diffusion coefficient, while the species subscripts *v* and *s* denote volume and surface species respectively.

The *DiffusionProcess* handles the voxel-to-voxel random walk of diffusing molecules and the collisions that take place between each walk. The latter is necessary when a diffusing species participates in a

strongly diffusion-limited reaction and the time slice between each walk is too large for an accurate value of p_j. Given t_s is the current simulation time, the next time a molecule of a diffusing species i with a diffusion coefficient D_i can be moved to a randomly selected neighbor voxel is

$$t_d^i = t_s + \frac{\alpha_i {r_v}^2}{D_i},$$

where in the HCP lattice, the constant $\alpha_i = \frac{2}{3}$ if it is a volume species or $\alpha_i = (\frac{2\sqrt{2}+4\sqrt{3}+3\sqrt{6}+\sqrt{22}}{6\sqrt{2}+4\sqrt{3}+3\sqrt{6}})^2$ if it belongs to a surface compartment. However, if i participates in a diffusion-limited reaction, a reactive collision may take place at time slices smaller than the walk interval $\frac{\alpha_i {r_v}^2}{D_i}$, causing $p_j > 1$. To ensure $p_j \leq 1$, we reduce the *DiffusionProcess* interval such that its next execution time becomes

$$t_d^i = \begin{cases} t_s + \frac{\alpha_i {r_v}^2}{D_i}, & \rho_i \leq P_i, \\ t_s + \frac{\alpha_i {r_v}^2 P_i}{D_i \rho_i}, & \rho_i > P_i. \end{cases}$$

Here P_i is an arbitrarily set reaction probability limit (default value is unity) such that $0 \leq P_i \leq 1$, and $\rho_i = \max\{p_1, \ldots, p_J\}$ where J is the total number of diffusion-influenced reactions participated by the species i. At each process interval, the molecule can collide as usual with a neighbor reactant pair and react with a scaled probability of $p_j P_i / \rho_i$. In the diffusion-limited case, $\rho_i > P_i$ and because of the reduced interval, the walk probability becomes less than unity to P_i / ρ_i.

Reactions that can be decoupled from diffusion such as zeroth- and first-order reactions, and second-order reactions that involve two adjoining immobile reactants or at least one HD reactant, are event-driven by the *SpatiocyteNextReactionProcess*. The reaction product can be made up of one or two molecules, which can be either HD or nonHD molecules. The *SpatiocyteNextReactionProcess* is an adapted implementation of the Next Reaction (NR) method (Gibson and Bruck, 2000), which itself is a variation of the Gillespie algorithm (Gillespie 1976, 1977).

In the process, the reaction propensity a_μ (unit s^{-1}) is calculated from the rate coefficient according to

$$a_\mu = \begin{cases} k_A A^\#, & A \xrightarrow{k_A} \text{product(s)}, \\ \frac{k_S}{V} A^\# S, & A \xrightarrow{k_S} \text{product(s)}, \\ \frac{k_{AB}}{V} A^\# B^\#, & A + B \xrightarrow{k_{AB}} \text{product(s)}, \\ \frac{k_{AA}}{V} A^\# (A^\# - 1), & A + A \xrightarrow{k_{AA}} \text{product(s)}. \end{cases}$$

Here, S and V are area and volume of the reaction compartment respectively, while k_S (unit ms^{-1}) is the surface-average adsorption rate of an HD volume species A. In the second-order reactions, V is replaced with S if both reactants are in a surface compartment. The next reaction time of a randomly selected molecule in a first order reaction or a pair of molecules in a second-order reaction is given by

$$t_r^\mu = t_s - \frac{\ln u_r}{a_\mu},$$

with u_r a uniformly distributed random number in the range (0,1).

If a reaction has a nonHD product, the new molecule will replace a nonHD reactant in the product compartment. Otherwise if the reaction only involves HD reactants or if the product belongs to a different compartment, the new nonHD molecule will be placed in a random vacant voxel of the product compartment. The placement of a second nonHD product also follows the same procedure. For intercompartmental reactions, a nonHD product will occupy a vacant voxel adjoining both compartments.

Dynamic localization patterns of simulated molecules can be directly compared with experimentally

obtained fluorescence microscopy images and videos using the *MicroscopyTrackingProcess* and the *SpatiocyteVisualizer*. Together, these modules simulate the microphotography process by recording the trajectory of simulated molecules over the camera exposure time and displaying their spatially localized densities. The *MicroscopyTrackingProcess* records the number of times the molecules of a species occupy each voxel at diffusion step intervals over the exposure time. The *SpatiocyteVisualizer* then displays the species color at each voxel with intensity and opacity levels that are directly proportional the voxel occupancy frequency. Colors from different species occupying the same voxel are blended to mimic co-localization patterns observed in multiple-labeling experiments.

Installing and Running Spatiocyte

The Spatiocyte source code is distributed as open source software under the GNU General Public License and is available at GitHub. At the time of writing, the Spatiocyte modules of the E-Cell System have been tested to run on Linux systems. Spatiocyte does not yet support other operating systems. Here we describe the installation procedures on a Ubuntu Linux system.

On a freshly installed Ubuntu Linux, E-Cell System version 3 and Spatiocyte requires several additional packages:

```
$ sudo apt-get install automake libtool g++ libgsl0-dev python-numpy python-ply libboost-python-dev libgtkmm-2.4-dev libgtkglextmm-x11-1.2-dev libhdf5-serial-dev git valgrind
```

The general installation procedure of the E-Cell System version 3 is as follows:

```
$ cd
$ mkdir wrk
$ cd wrk
$ git clone https://github.com/ecell/ecell3.git
$ cd ecell3
$ ./autogen.sh
$ ./configure --prefix=$HOME/root
$ make -j3 (or just make, if there is only one CPU core available)
$ make install (files will be installed in the $HOME/root directory)
$ gedit ~/.bashrc (other editors such as emacs or vim can also be used here)
```

The following lines, which specify the environment variables of the E-Cell System should be appended to the .bashrc file:

```
  export PATH=$HOME/root/bin:$PATH
  export LD_LIBRARY_PATH=$HOME/root/lib:$LD_LIBRARY_PATH:.
  export PYTHONPATH=$HOME/root/lib/python:$HOME/root/lib/python2.7/site-packages:$PYTHONPATH
  export ECELL3_DM_PATH=.
```

In the line 3 above, the Python version number '2.7' should be updated if it is different in the installed system. Next, we load the new environment variables:

```
$ source ~/.bashrc
$ ecell3-session-monitor (try opening it, the window shown in Figure 1 should appear, and then close it)
```

We can now attempt to run a simple model in the E-Cell Model (EM) language, `simple.em`:

```
$ cd $HOME/wrk/ecell3/doc/samples/simple/
$ ecell3-em2eml simple.em
$ ecell3-session-monitor
```

Using `ecell3-em2eml`, the model file `simple.em` was converted into `simple.eml` in Extensible Markup Language (XML) format. The `simple.eml` file can now be loaded from the File menu of the E-Cell Session Monitor or the File open button (see Figure 1). Try running the simulation by clicking on the Start button.

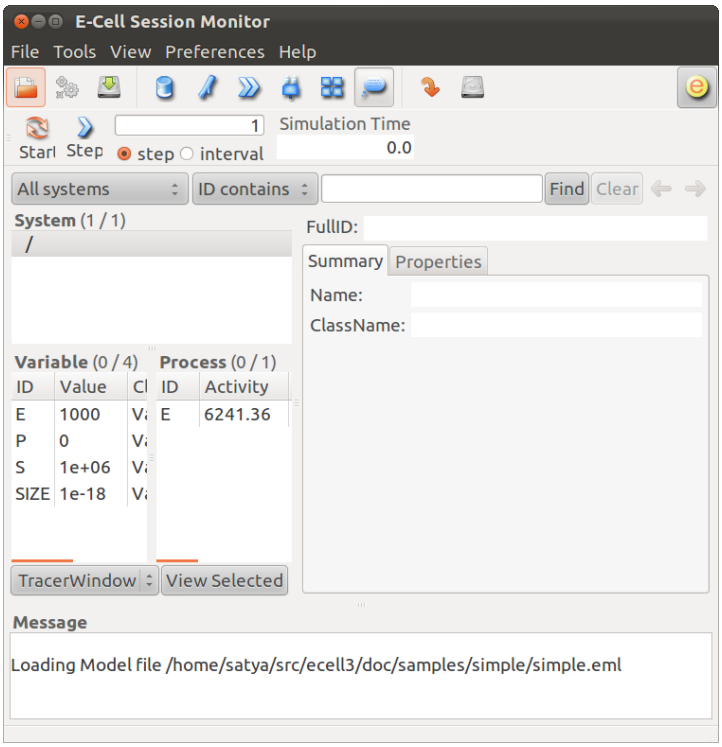

Figure 1: The E-Cell Session Monitor

The steps to install E-Cell3-Spatiocyte are as follows:

```
$ cd $HOME/wrk
$ git clone git://github.com/ecell/ecell3-spatiocyte.git
$ cd ecell3-spatiocyte
$ make -j3 (or just make, if there is only one CPU core available)
```

The E-Cell3-Spatiocyte package includes the MinDE model (see Figure 2) reported in (Arjunan and Tomita, 2010). We can now attempt to run the model with the following steps:

```
$ cd $HOME/wrk/ecell3-spatiocyte/
$ ecell3-em2eml 2010.arjunan.syst.synth.biol.wt.em
$ ecell3-session-monitor
```

Load the model `2010.arjunan.syst.synth.biol.wt.eml` and try running the simulation for 90 seconds. We can also run Spatioctye models using command line interface of the E-Cell System:

```
$ ecell3-session -f 2010.arjunan.syst.synth.biol.wt.em
<2010.arjunan.syst.synth.biol.wt.eml, t=0>>> run(90)
<2010.arjunan.syst.synth.biol.wt.eml, t=90>>> exit()
```

Models that are created using the Python script can be run as,

```
$ ecell3-session 2012.arjunan.chapter.neuron.py
```

When running a Spatiocyte model with the *VisualizationLogProcess* module enabled, the three-dimensional positional information of a logged molecule species will be stored in `visualLog0.dat` (default file name). The molecules can be viewed in a separate visualizer window even while the simulation is still running. To view them, we can run *SpatiocyteVisualizer* by issuing

```
$ ./spatiocyte
```

The visualizer will load the `visualLog0.dat` file by default and display the molecules at every log interval (see Figure 3). The keyboard shortcuts that are available for the visualizer are listed in the *SpatiocyteVisualizer* module section.

If the program fails and crashes when loading or running a model, we can get some debugging information using the Valgrind tool:

```
$ valgrind --tool=memcheck --num-callers=40 --leak-check=full python $HOME/root/bin/ecell3-session -f modelFileName.eml
```

```
Stepper SpatiocyteStepper(SS) { VoxelRadius 1e-8; } # m
System System(/) {
  StepperID SS;
  Variable Variable(GEOMETRY) { Value 3; } # rod shaped compartment
  Variable Variable(LENGTHX) { Value 4.5e-6; } # m
  Variable Variable(LENGTHY) { Value 1e-6; } # m
  Variable Variable(VACANT) { Value 0; }
  Variable Variable(MinDatp) { Value 0; } # molecule number
  Variable Variable(MinDadp) { Value 1300; } # molecule number
  Variable Variable(MinEE) { Value 0; } # molecule number
  Process DiffusionProcess(diffuseMinD) {
    VariableReferenceList [_ Variable:/:MinDatp] [_ Variable:/:MinDadp];
    D 16e-12; } # m^2/s
  Process DiffusionProcess(diffuseMinE) {
    VariableReferenceList [_ Variable:/:MinEE];
    D 10e-12; } # m^2/s
  Process VisualizationLogProcess(visualize) {
    VariableReferenceList [_ Variable:/Surface:MinEE] [_ Variable:/Surface:MinDEE] [_ Variable:/Surface:MinDEED]
                          [_ Variable:/Surface:MinD];
    LogInterval 0.5; } # s
  Process MicroscopyTrackingProcess(track) {
    VariableReferenceList [_ Variable:/Surface:MinEE 2] [_ Variable:/Surface:MinDEE 3] [_ Variable:/Surface:MinDEED 4]
                          [_ Variable:/Surface:MinD 1] [_ Variable:/Surface:MinEE -2] [_ Variable:/Surface:MinDEED -2]
                          [_ Variable:/Surface:MinEE -1] [_ Variable:/Surface:MinDEED -4] [_ Variable:/Surface:MinD -1];
    FileName "microscopyLog0.dat"; }
  Process MoleculePopulateProcess(populate) {
    VariableReferenceList [_ Variable:/:MinDatp] [_ Variable:/:MinDadp] [_ Variable:/:MinEE] [_ Variable:/Surface:MinD]
                          [_ Variable:/Surface:MinDEE] [_ Variable:/Surface:MinDEED] [_ Variable:/Surface:MinEE]; }
}

System System(/Surface) {
  StepperID SS;
  Variable Variable(DIMENSION) { Value 2; } # surface compartment
  Variable Variable(VACANT) { Value 0; }
  Variable Variable(MinD) { Value 0; } # molecule number
  Variable Variable(MinEE) { Value 0; } # molecule number
  Variable Variable(MinDEE) { Value 700; } # molecule number
  Variable Variable(MinDEED) { Value 0; } # molecule number
  Process DiffusionProcess(diffuseMinD) {
    VariableReferenceList [_ Variable:/Surface:MinD];
    D 0.02e-12; } # m^2/s
  Process DiffusionProcess(diffuseMinEE) {
    VariableReferenceList [_ Variable:/Surface:MinEE];
    D 0.02e-12; } # m^2/s
  Process DiffusionProcess(diffuseMinDEE) {
    VariableReferenceList [_ Variable:/Surface:MinDEE];
    D 0.02e-12; } # m^2/s
  Process DiffusionProcess(diffuseMinDEED) {
     VariableReferenceList [_ Variable:/Surface:MinDEED];
    D 0.02e-12; } # m^2/s
  Process DiffusionInfluencedReactionProcess(reaction1) {
    VariableReferenceList [_ Variable:/Surface:VACANT -1] [_ Variable:/:MinDatp -1] [_ Variable:/Surface:MinD 1];
    k 2.2e-8; } # m/s
  Process DiffusionInfluencedReactionProcess(reaction2) {
    VariableReferenceList [_ Variable:/Surface:MinD -1] [_ Variable:/:MinDatp -1] [_ Variable:/Surface:MinD 1]
                          [_ Variable:/Surface:MinD 1];
    k 3e-20; } # m^3/s
  Process DiffusionInfluencedReactionProcess(reaction3) {
    VariableReferenceList [_ Variable:/Surface:MinD -1] [_ Variable:/:MinEE -1] [_ Variable:/Surface:MinDEE 1];
    k 5e-19; } # m^3/s
  Process SpatiocyteNextReactionProcess(reaction4) {
    VariableReferenceList [_ Variable:/Surface:MinDEE -1] [_ Variable:/Surface:MinEE 1] [_ Variable:/:MinDadp 1];
    k 1; } # s^{-1}
  Process SpatiocyteNextReactionProcess(reaction5) {
    VariableReferenceList [_ Variable:/:MinDadp -1] [_ Variable:/:MinDatp 1];
    k 5; } # s^{-1}
  Process DiffusionInfluencedReactionProcess(reaction6) {
    VariableReferenceList [_ Variable:/Surface:MinDEE -1] [_ Variable:/Surface:MinD -1] [_ Variable:/Surface:MinDEED 1];
    k 5e-15; } # m^2/s
  Process SpatiocyteNextReactionProcess(reaction7) {
    VariableReferenceList [_ Variable:/Surface:MinDEED -1] [_ Variable:/Surface:MinDEE 1] [_ Variable:/:MinDadp 1];
    k 1; } # s^{-1}
  Process SpatiocyteNextReactionProcess(reaction8) {
    VariableReferenceList [_ Variable:/Surface:MinEE -1] [_ Variable:/:MinEE 1];
    k 0.83; } # s^{-1}
}
```

Figure 2: E-Cell Model (EM) description file for the MinDE model. The file is available in the Spatiocyte source package as 2010.arjunan.syst.synth.biol.wt.em.

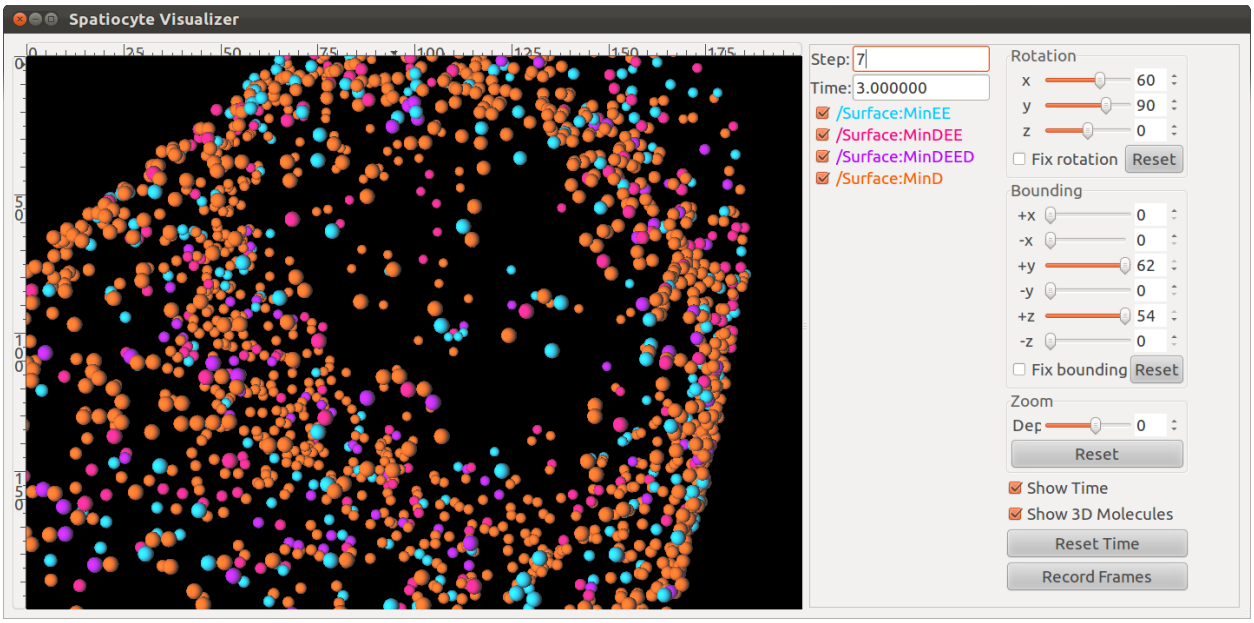

Figure 3: The *SpatiocyteVisualizer* displaying simulated membrane-bound proteins of the MinDE model.

Spatiocyte Modules

In Spatiocyte modules, the unit of numeric values is given in meters, seconds, radians and molecule numbers. A Spatiocyte model file created using the E-Cell Model (EM) language is shown in Figure 2. The file contains the wildtype *Escherichia coli* MinDE cytokinesis regulation model that was reported in (Arjunan and Tomita, 2010). A schematic representation of the model is given in Figure 4. Python script examples to build models with more complex compartments are provided in Figures 5 and 6. Figures 7 and 8 illustrate 3D visualizations of the resulting models.

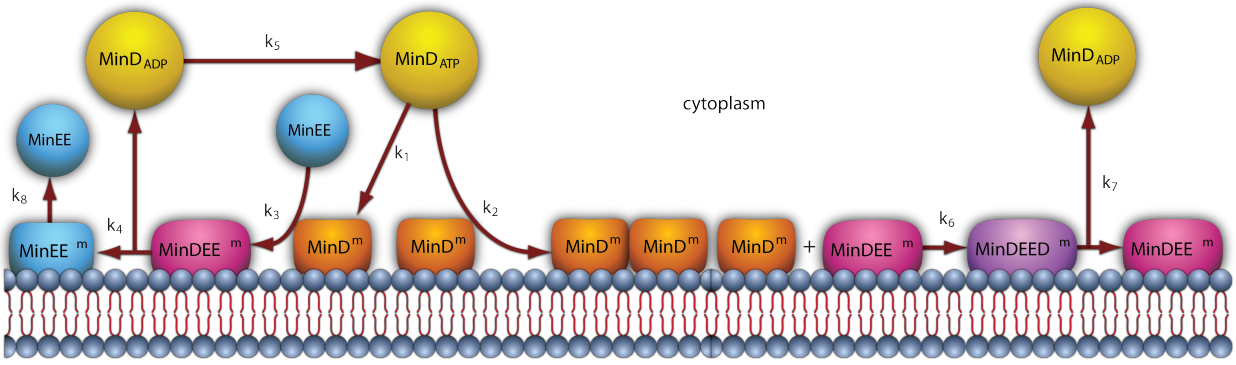

Figure 4: A schematic representation of the MinDE model.

Compartment

Compartments are defined hierarchically and follow the format used by the E-Cell System version 3 (see the E-Cell Simulation Environment Version 3 User's Manual for details). Each sub-compartment within a parent compartment is created according to the alphabetical order of the compartment names. Predefined *Variables* that specify the *Compartment* properties include DIMENSION, GEOMETRY, LENGTHX, LENGTHY, LENGTHZ, ORIGINX, ORIGINY, ORIGINZ, ROTATEX, ROTATEY, ROTATEZ, XYPLANE, XZPLANE, YZPLANE, VACANT, DIFFUSIVE and REACTIVE. Examples of these variable definitions can be seen in Figures 2 (lines 4-7 and 33-34), 5 (lines 5-8, 17-24, 31-32, 41-49 and 52-54) and 6 (lines 46-49, 59-60, 63-69 and 71-72).

```
# Example of python scripting to create a neuron with 5 minor processes
theSimulator.createStepper('SpatiocyteStepper', 'SS').VoxelRadius = 10e-8
# Create the root container compartment using the default Cuboid geometry:
theSimulator.rootSystem.StepperID = 'SS'
theSimulator.createEntity('Variable', 'Variable:/:LENGTHX').Value = 61e-6
theSimulator.createEntity('Variable', 'Variable:/:LENGTHY').Value = 25e-6
theSimulator.createEntity('Variable', 'Variable:/:LENGTHZ').Value = 5.5e-6
theSimulator.createEntity('Variable', 'Variable:/:VACANT')
logger = theSimulator.createEntity('VisualizationLogProcess', 'Process:/:logger')
logger.LogInterval = 1
logger.VariableReferenceList = [['_', 'Variable:/Soma/Membrane:VACANT'], ['_', 'Variable:/Soma:K']]
logger.VariableReferenceList = [['_', 'Variable:/Dendrite%d/Membrane:VACANT' %i] for i in range(5)]
populator = theSimulator.createEntity('MoleculePopulateProcess', 'Process:/:populate')
populator.VariableReferenceList = [['_', 'Variable:/Soma:K']]
# Create the Soma compartment of the Neuron:
theSimulator.createEntity('System', 'System:/:Soma').StepperID = 'SS'
theSimulator.createEntity('Variable', 'Variable:/Soma:GEOMETRY').Value = 1
theSimulator.createEntity('Variable', 'Variable:/Soma:LENGTHX').Value = 10e-6
theSimulator.createEntity('Variable', 'Variable:/Soma:LENGTHY').Value = 10e-6
theSimulator.createEntity('Variable', 'Variable:/Soma:LENGTHZ').Value = 6.5e-6
theSimulator.createEntity('Variable', 'Variable:/Soma:ORIGINX').Value = -0.48
theSimulator.createEntity('Variable', 'Variable:/Soma:ORIGINY').Value = -0.2
theSimulator.createEntity('Variable', 'Variable:/Soma:ORIGINZ').Value = -0.6
theSimulator.createEntity('Variable', 'Variable:/Soma:VACANT')
theSimulator.createEntity('Variable', 'Variable:/Soma:K').Value = 1000
diffuser = theSimulator.createEntity('DiffusionProcess', 'Process:/Soma:diffuseK')
diffuser.VariableReferenceList = [['_', 'Variable:.:K']]
diffuser.D = 0.2e-12
# Create the Soma membrane:
theSimulator.createEntity('System', 'System:/Soma:Membrane').StepperID = 'SS'
theSimulator.createEntity('Variable', 'Variable:/Soma/Membrane:DIMENSION').Value = 2
theSimulator.createEntity('Variable', 'Variable:/Soma/Membrane:VACANT')
# Parameters of Dendrites/Minor Processes:
dendritesLengthX = [40e-6, 10e-6, 10e-6, 10e-6, 10e-6]
dendritesOriginX = [0.32, -0.78, -0.48, -0.3, -0.66]
dendritesOriginY = [-0.2, -0.2, 0.52, -0.65, -0.65]
dendritesRotateZ = [0, 0, 1.57, 0.78, -0.78]
for i in range(5):
  # Create the Dendrite:
  theSimulator.createEntity('System', 'System:/:Dendrite%d' %i).StepperID = 'SS'
  theSimulator.createEntity('Variable', 'Variable:/Dendrite%d:GEOMETRY' %i).Value = 3
  theSimulator.createEntity('Variable', 'Variable:/Dendrite%d:LENGTHX' %i).Value = dendritesLengthX[i]
  theSimulator.createEntity('Variable', 'Variable:/Dendrite%d:LENGTHY' %i).Value = 1.5e-6
  theSimulator.createEntity('Variable', 'Variable:/Dendrite%d:ORIGINX' %i).Value = dendritesOriginX[i]
  theSimulator.createEntity('Variable', 'Variable:/Dendrite%d:ORIGINY' %i).Value = dendritesOriginY[i]
  theSimulator.createEntity('Variable', 'Variable:/Dendrite%d:ORIGINZ' %i).Value = -0.6
  theSimulator.createEntity('Variable', 'Variable:/Dendrite%d:ROTATEZ' %i).Value = dendritesRotateZ[i]
  theSimulator.createEntity('Variable', 'Variable:/Dendrite%d:VACANT' %i)
  theSimulator.createEntity('Variable', 'Variable:/Dendrite%d:DIFFUSIVE' %i).Name = '/:Soma'
  # Create the Dendrite membrane:
  theSimulator.createEntity('System', 'System:/Dendrite%d:Membrane' %i).StepperID = 'SS'
  theSimulator.createEntity('Variable', 'Variable:/Dendrite%d/Membrane:DIMENSION' %i).Value = 2
  theSimulator.createEntity('Variable', 'Variable:/Dendrite%d/Membrane:VACANT' %i)
  theSimulator.createEntity('Variable', 'Variable:/Dendrite%d/Membrane:DIFFUSIVE' %i).Name = '/Soma:Membrane'
run(100)
```

Figure 5: A Python script to create a neuron-shaped model. The file is available in the Spatiocyte source package as 2012.arjunan.chapter.neuron.py.

```
import math
import random
minDist = 75e-9
dendriteRadius = 0.75e-6
dendriteLength = 10e-6
lengths = [8.4e-6, 6.3e-6, 4.2e-6, 2.1e-6, 1e-6]
lengthFreqs = [7, 10, 11, 21, 108]
mtOriginX = []
mtOriginZ = []
mtOriginY = []
expandedLengths = []

def isSpacedOut(x, y, z, length):
  for i in range(len(expandedLengths)-1):
    maxOriX = mtOriginX[i]*dendriteLength/2 + expandedLengths[i]/2
    minOriX = mtOriginX[i]*dendriteLength/2 - expandedLengths[i]/2
    maxX = x*dendriteLength/2 + length/2
    minX = x*dendriteLength/2 - length/2
    y2 = math.pow((y-mtOriginY[i])*dendriteRadius, 2)
    z2 = math.pow((z-mtOriginZ[i])*dendriteRadius, 2)
    if((minX <= maxOriX or maxX >= minOriX) and math.sqrt(y2+z2) < minDist):
      return False
    elif(minX > maxOriX and math.sqrt(y2+z2+math.pow(minX-maxOriX, 2)) < minDist):
      return False
    elif(maxX < minOriX and math.sqrt(y2+z2+math.pow(maxX-minOriX, 2)) < minDist):
      return False
  return True

for i in range(len(lengthFreqs)):
  maxX = (dendriteLength-lengths[i])/dendriteLength
  for j in range(int(lengthFreqs[i])):
    expandedLengths.append(lengths[i])
    x = random.uniform(-maxX, maxX)
    y = random.uniform(-0.95, 0.95)
    z = random.uniform(-0.95, 0.95)
    while(y*y+z*z > 0.9 or not isSpacedOut(x, y, z, lengths[i])):
      x = random.uniform(-maxX, maxX)
      y = random.uniform(-0.95, 0.95)
      z = random.uniform(-0.95, 0.95)
    mtOriginX.append(x)
    mtOriginY.append(y)
    mtOriginZ.append(z)

theSimulator.createStepper('SpatiocyteStepper', 'SS').VoxelRadius = 0.8e-8
theSimulator.rootSystem.StepperID = 'SS'
theSimulator.createEntity('Variable', 'Variable:/:GEOMETRY').Value = 3
theSimulator.createEntity('Variable', 'Variable:/:LENGTHX').Value = dendriteLength
theSimulator.createEntity('Variable', 'Variable:/:LENGTHY').Value = dendriteRadius*2
theSimulator.createEntity('Variable', 'Variable:/:VACANT')
theSimulator.createEntity('Variable', 'Variable:/:K').Value = 100
diffuser = theSimulator.createEntity('DiffusionProcess', 'Process:/:diffuseK')
diffuser.VariableReferenceList = [['_', 'Variable:/:K']]
diffuser.D = 0.2e-12
visualLogger = theSimulator.createEntity('VisualizationLogProcess', 'Process:/:visualLogger')
visualLogger.LogInterval = 1
visualLogger.VariableReferenceList = [['_', 'Variable:/Membrane:VACANT'], ['_', 'Variable:/:K']]
theSimulator.createEntity('MoleculePopulateProcess', 'Process:/:populate').VariableReferenceList = [['_', 'Variable:/:K']]
theSimulator.createEntity('System', 'System:/:Membrane').StepperID = 'SS'
theSimulator.createEntity('Variable', 'Variable:/Membrane:DIMENSION').Value = 2
theSimulator.createEntity('Variable', 'Variable:/Membrane:VACANT')
for i in range(len(expandedLengths)):
  theSimulator.createEntity('System', 'System:/:Microtubule%d' %i).StepperID = 'SS'
  theSimulator.createEntity('Variable', 'Variable:/Microtubule%d:GEOMETRY' %i).Value = 2
  theSimulator.createEntity('Variable', 'Variable:/Microtubule%d:LENGTHX' %i).Value = expandedLengths[i]
  theSimulator.createEntity('Variable', 'Variable:/Microtubule%d:LENGTHY' %i).Value = 6e-9
  theSimulator.createEntity('Variable', 'Variable:/Microtubule%d:ORIGINX' %i).Value = mtOriginX[i]
  theSimulator.createEntity('Variable', 'Variable:/Microtubule%d:ORIGINY' %i).Value = mtOriginY[i]
  theSimulator.createEntity('Variable', 'Variable:/Microtubule%d:ORIGINZ' %i).Value = mtOriginZ[i]
  theSimulator.createEntity('Variable', 'Variable:/Microtubule%d:VACANT' %i)
  theSimulator.createEntity('System', 'System:/Microtubule%d:Membrane' %i).StepperID = 'SS'
  theSimulator.createEntity('Variable', 'Variable:/Microtubule%d/Membrane:DIMENSION' %i).Value = 2
  theSimulator.createEntity('Variable', 'Variable:/Microtubule%d/Membrane:VACANT' %i)
  visualLogger.VariableReferenceList = [['_', 'Variable:/Microtubule%d/Membrane:VACANT' %i]]
run(100)
```

Figure 6: A Python script to create a compartment with randomly distributed microtubules. The file is available in the Spatiocyte source package as 2012.arjunan.chapter.microtubules.py.

Molecule species within a *Compartment* are also defined as a *Variable*. The *Value* property of each species stipulates the molecule number during initialization. All species by default are nonHD. Examples of nonHD species definitions can be seen in Figures 2 (lines 8-10 and 35-38), 5 (line 25) and 6 (line 50). To define a HD species, the *Name* property of the *Variable* should be set to “HD” as shown in the EM

and Python examples below:

```
Variable Variable(A) {
    Value 100;
    Name "HD"; }
```

```
A = theSimulator.createEntity('Variable', 'Variable:.:A')
A.Value = 100
A.Name = “HD”
```

DIMENSION

The DIMENSION variable defines the spatial dimension of the compartment, whether it is a line ('1'), surface (‘2’) or a volume (‘3’) type. At the time of writing, the line compartment type is still in development. A surface compartment encloses its parent volume compartment, and as a result, it cannot be defined independently without a volume compartment to enclose with. A surface compartment does not have any child volume or surface compartment. The root compartment should always be defined as a volume compartment. Since the default DIMENSION value is ‘3’, a volume compartment can be defined without the DIMENSION variable. A volume compartment can also use the predefined variables GEOMETRY, LENGTHX, LENGTHY, LENGTHZ, ORIGINX, ORIGINY, ORIGINZ, ROTATEX, ROTATEY, ROTATEZ, XYPLANE, XZPLANE, YZPLANE, DIFFUSIVE and VACANT, whereas a surface compartment only requires the DIMENSION and VACANT variables and inherits the remaining relevant properties from its parent compartment. In addition, surface compartments can also define the DIFFUSIVE and REACTIVE variables. See Figures 2 (line 33), 5 (lines 31 and 52) and 6 (lines 59 and 71) for examples of the DIMENSION variable definition.

Figure 7: A neuron-shaped compartment created from a combination of rod and ellipsoid compartment geometries. The model is created from the Python script shown in Figure 5.

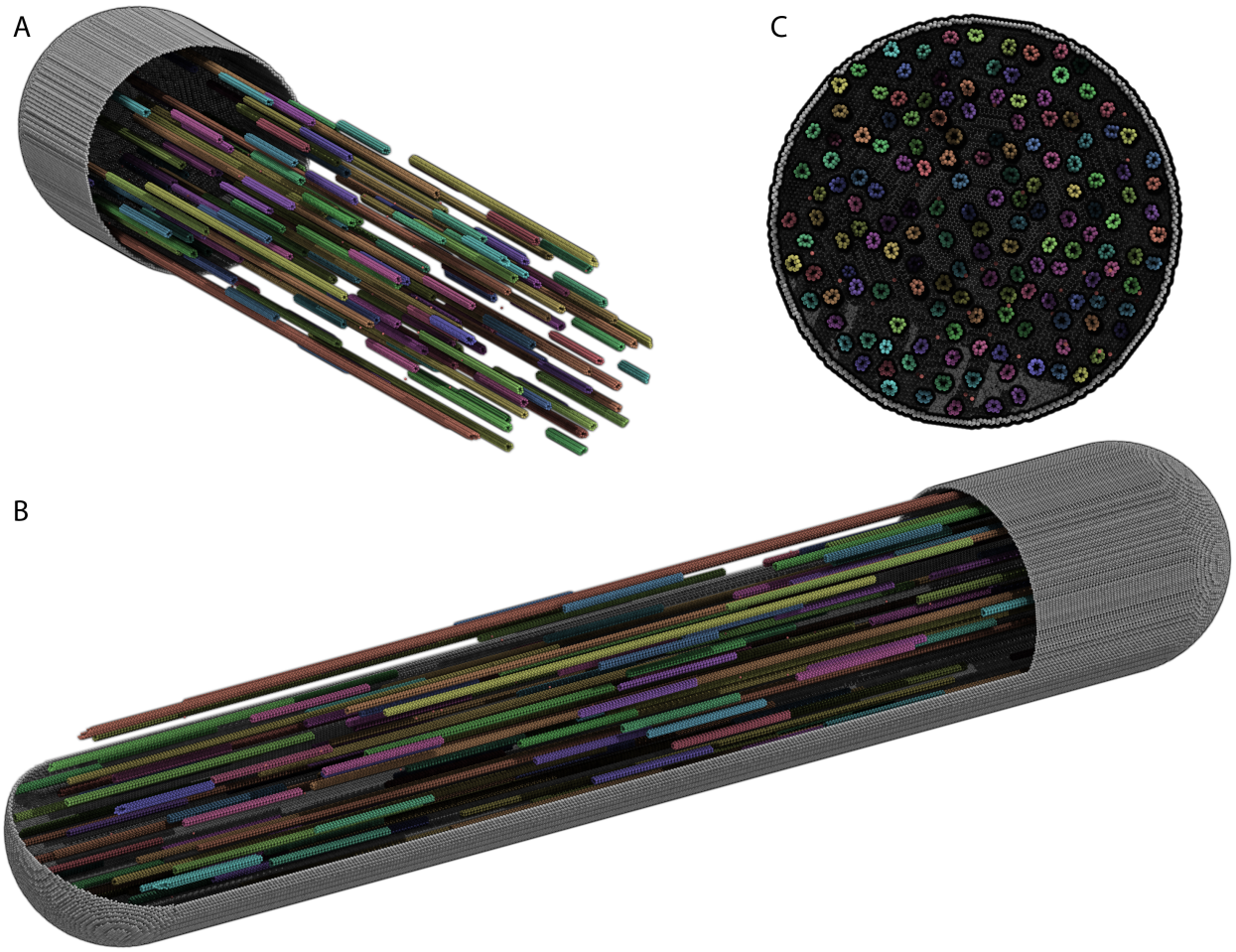

Figure 8: A rod compartment containing randomly distributed microtubules built from cylinder compartments. The model is created from the Python script shown in Figure 6. The steps to create each of the displayed panels in *SpatiocyteVisualizer* are as follows: (A) (i) select all species (i.e., the default configuration), (ii) decrease the +x range to the desired level, (iii) deselect the membrane.VACANT species, (iv) increase the +x range to the maximum level, and (v) select the membrane.VACANT species; (B) the same steps as in (A) and increase -y range to the desired level; and (C) the same steps as in (A) and rotate to the desired angle.

GEOMETRY

The GEOMETRY variable of a volume compartment specifies one of the six supported geometric primitives: cuboid ('0'), ellipsoid (‘1’), cylinder (‘2’), rod (‘3’), torus ('4') and pyramid ('5'). More complex forms can be constructed using a combination of these primitives. Figures 4 and 6 illustrate the construction of a neuron-shaped model using a combination of ellipsoid and rod compartments. Compartments without the GEOMETRY definition is set to the cuboid form since the default value is ‘0’. For examples of GEOMETRY definition see Figures 2 (line 4), 5 (lines 17 and 41) and 6 (lines 46 and 63).

LENGTH[X, Y, Z]

The three variables LENGTH[X, Y, Z] can specify the compartment lengths in the directions of [x, y, z]-axes, respectively. The cuboid, ellipsoid and pyramid compartments use all three variables. If all three

lengths are equal, a cube or a sphere compartment can be created with a cuboid or an ellipsoid geometry, respectively. For the pyramid compartment, LENGTH[X, Y, Z] stipulate its base length, height and base width, respectively. For a cylinder compartment, LENGTHX defines the cylinder length, while its diameter is given by LENGTHY. In the case of a rod compartment, LENGTHX indicates the length from the tip of one pole to the other while LENGTHY defines its diameter. For a torus, its larger diameter (from the torus center to the edge) is given by LENGTHX, whereas LENGTHY determines the tube diameter. LENGTH[X, Y, Z] definitions examples are given in Figures 2 (lines 5-6), 5 (lines 5-7, 18-20, and 42-43) and 6 (lines 47-48 and 64-65).

[XY, XZ, YZ]PLANE

When a volume compartment has the cuboid geometry, the boundary type or the presence of the [xy, xz, yz]-plane surfaces enclosing the compartment can be specified using [XY, XZ, YZ]PLANE variables. The boundary type can be reflective (‘0’), periodic (‘1’) or semi-periodic (‘2’). A semi-periodic boundary allows nonHD molecules to move unidirectionally from one boundary to the other. When a surface compartment is defined to enclose the cuboid compartment, we can remove one or both faces of the cuboid in a given [XY, XZ, YZ]PLANE. To remove the surface on the upper or the lower face of the cuboid in a plane, we can set the variable to ‘3’ or ‘4’, respectively, whereas to remove both faces we can set it to ‘5’. If the variable is not defined, the boundary type is set to the default reflective (‘0’) type. Examples in EM and Python to remove both of the cuboid XYPLANE faces are given below:

```
Variable Variable(XYPLANE) { Value 5; }
```

```
theSimulator.createEntity('Variable', 'Variable:.:XYPLANE').Value = 5
```

ORIGIN[X, Y, Z]

A child volume compartment can be placed at any location within a parent compartment using the variables ORIGIN[X, Y, Z]. The variables define the origin (center) coordinates of the child compartment relative to its parent center point. The variable values ‘-1’ and ‘1’ correspond to the normalized lowest and the highest points of the parent compartment in a given axis, respectively. Since the default value of these variables is ‘0’, the child compartment will be placed at the center of its parent if they are not defined. Figures 5 (lines 21-24 and 44-46) and 6 (lines 66-68) give some examples of the ORIGIN[X, Y, Z] variables definition.

ROTATE[X, Y, Z]

A compartment can be rotated along the [x, y, z]-axis with the origin at the compartment center using the ROTATE[X, Y, Z] variables respectively. The unit of the variables is in radians. If there are multiple rotation definitions, they follow the [x, y, z]-axis rotation order. Compartments are not rotated if the variables are not defined since their default value is '0'. An example of compartment rotation definition is given in Figure 5 (line 47).

VACANT

Every compartment must have a VACANT variable that represents the ‘species’ of empty voxels within the compartment. The VACANT voxels of a surface compartment are analogous to the lipid molecules mentioned in the Spatiocyte Method section and in (Arjunan and Tomita, 2010). Examples of the VACANT variable definition are shown in Figures 2 (lines 7 and 34), 5 (lines 8, 24, 32, 48 and 53) and 6 (lines 49, 60, 69 and 72). The variable can be used to define sink (e.g., A -> VACANT) and membrane binding reactions (e.g., B_V + $VACANT_S$ -> B_S) of nonHD species, as shown in the EM and Python examples below:

First-Order Sink Reaction, A → Ø

```
Process SpatiocyteNextReactionProcess(sink) {
    VariableReferenceList [_ Variable:/:A -1]
                          [_ Variable:/:VACANT 1];
    k 0.3; }
```

Second-Order Surface-Adsorption Reaction, B_v + Surface.VACANT → B_s

```
Process DiffusionInfluencedReactionProcess(bind) {
    VariableReferenceList [_ Variable:/:B -1]
                          [_ Variable:/Surface:VACANT -1]
                          [_ Variable:/Surface:B 1];
    k 2e-8; }
```

First-Order Sink Reaction, A → Ø

```
sinker = theSimulator.createEntity('SpatiocyteNextReactionProcess', 'Process:/:sink')
sinker.VariableReferenceList = [['_', 'Variable:/:A', '-1']]
sinker.VariableReferenceList = [['_', 'Variable:/:VACANT', '1']]
sinker.k = 0.3
```

Second-Order Surface-Adsorption Reaction, B_v + Surface.VACANT → B_s

```
binder = theSimulator.createEntity('DiffusionInfluencedReactionProcess', 'Process:/:bind')
binder.VariableReferenceList = [['_', 'Variable:/:B', '-1']]
binder.VariableReferenceList = [['_', 'Variable:/Surface:VACANT', '-1']]
binder.VariableReferenceList = [['_', 'Variable:/Surface:B', '1']]
binder.k = 2e-8
```

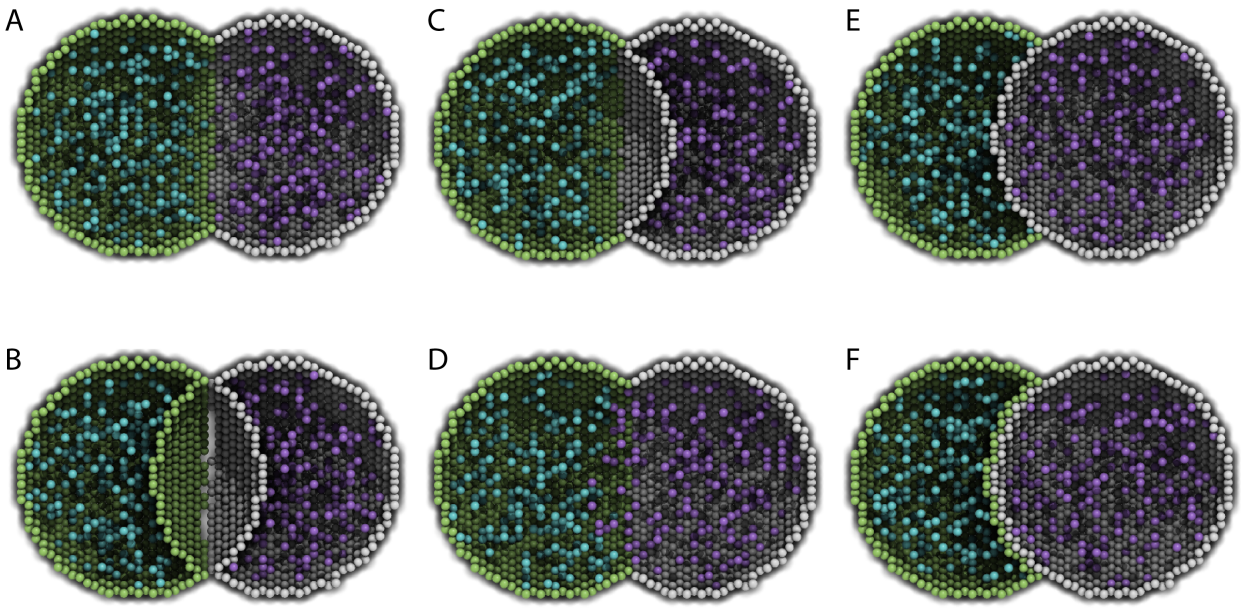

Figure 9: Cross-sections of two intersected peer compartments. Two sphere compartments in green and white are intersecting in space. Turquoise and purple molecules belong to the green and white compartments respectively. See text of the VACANT variable and Table 1 for a detailed description of the intersections. The EM file to create the intersections is available in the Spatiocyte source package as 2012.arjunan.chapter.peer.em.

For a volume compartment, the *Value* of the VACANT variable determines if the compartment has a

higher occupancy priority when it intersects with a peer compartment. Figure 9 displays cross-sections of various intersection forms of two spherical peer compartments with different volume and surface VACANT values (listed in Table 1). In the case of a surface compartment, the VACANT variable determines if it fully encloses a parent compartment that has an intersection. A nonzero value indicates that the parent will be fully enclosed even at the location of intersection. Otherwise if the value is '0', the surface will be open at the intersecting region. Figure 10 shows four possible enclosure forms when a compartment intersects with a root compartment. Figure 7 illustrates the intersection of various compartments to create a unified neuron-shaped compartment.

Table 1: Combinations of volume and surface VACANT values and their corresponding intersected peer compartment forms. In all cases X is an integer and the DIFFUSIVE variable is not set.

Green Sphere Compartment		**White Sphere Compartment**		**Intersection Form in Figure 9**
Volume VACANT.Value	**Surface VACANT.Value**	**Volume VACANT.Value**	**Surface VACANT.Value**	
X	0	X	0	A
X	nonzero	X	nonzero	B
X	0	X	nonzero	C
< X	0	X	0	D
< X	0	X	nonzero	E
< X	nonzero	X	nonzero	F

DIFFUSIVE

To unify intersecting compartments, the DIFFUSIVE variable can be specified. It enables nonHD molecules to diffuse into and from an intersecting compartment. The *Name* property of the DIFFUSIVE variable defines the path and name of the diffusible intersecting compartment. With the DIFFUSIVE variable defined, the VACANT species of the unified compartments become identical. Figure 5 (lines 49 and 54) gives some examples of the DIFFUSIVE variable definition and usage.

REACTIVE

The REACTIVE variable enables nonHD molecules in a surface compartment to collide and react with the VACANT voxels (i.e., lipids) and nonHD molecules in an adjacent surface compartment. The *Name* property of the REACTIVE variable specifies the path and name of the reactive adjacent surface compartment. Examples of the REACTIVE variable definition in EM and Python are given below:

```
Variable Variable(REACTIVE) { Name "/Cell:Surface"; }
```

```
theSimulator.createEntity('Variable', 'Variable:/Surface:REACTIVE').Name = "/Cell:Surface"
```

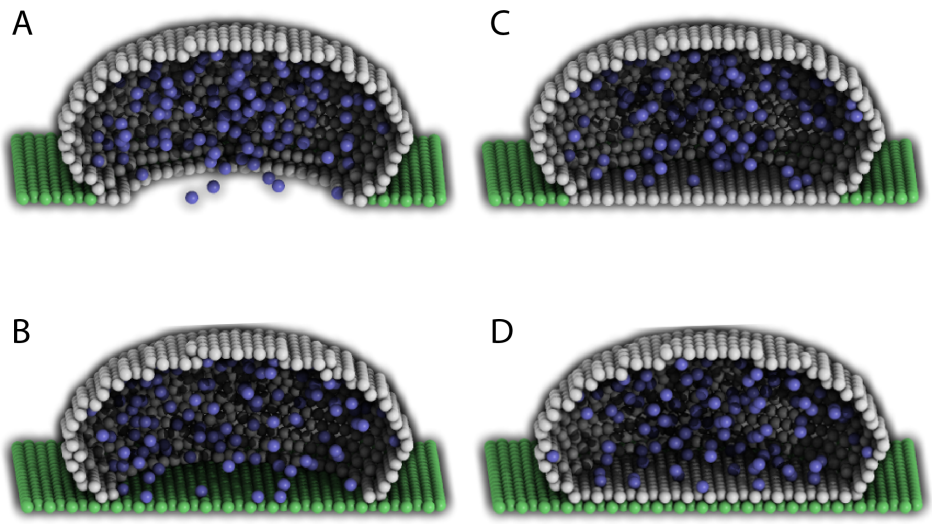

Figure 10: Cross-sections of intersected root and child compartments. The VACANT surface voxels of the cuboid root compartment are shown in green while those of the ellipsoid child compartment are in white. The blue molecules belong to the child volume compartment. (A) root surface.VACANT = 0 and child surface.VACANT = 0, (B) root surface.VACANT = 1 and child surface.VACANT = 0, (C) root surface.VACANT = 0 and child surface.VACANT = 1, and (D) root surface.VACANT = 1 and child surface.VACANT = 1. The EM file to create the intersections is available in the Spatiocyte source package as 2012.arjunan.chapter.root.em.

SpatiocyteStepper

The *SpatiocyteStepper* is the only stepper used by Spatiocyte in the E-Cell System and must be defined to run all simulations. It advances the simulation in an event-driven manner. Initialization examples of the *SpatiocyteStepper* are shown in Figures 2 (line 1), 5 (line 2) and 6 (line 44). In each compartment, the StepperID must be set to the *SpatiocyteStepper* ID. Examples of *SpatiocyteStepper* ID definition in compartments are given in Figures 2 (lines 3 and 32), 5 (lines 4, 16, 30, 40 and 51) and 6 (lines 45, 58, 62 and 70).

VoxelRadius

The radius of the HCP lattice voxels can be set in the *SpatiocyteStepper* using the VoxelRadius property. The default radius is 10e-9 m. Figures 2 (line 1), 5 (line 2) and 6 (line 44) show some examples of the VoxelRadius initialization.

SearchVacant

The SearchVacant property of the *SpatiocyteStepper* provides an option to direct the simulator to search for all adjacent voxels for vacancy during dissociation reactions that result in nonHD product molecules. The reaction can only take place if there is an available target vacant voxel. This option is useful when evaluating the effects of a crowded compartment. The value of SearchVacant by default is true (‘1’). To disable it, we can set it to ‘0’. When disabled, an adjacent target voxel is selected randomly and the reaction is only executed if the voxel is vacant. EM and Python examples of SearchVacant initialization are as follows:

```
Stepper SpatiocyteStepper(SS) { SearchVacant 0; }
```

```
theSimulator.createStepper('SpatiocyteStepper', 'SS').SearchVacant = 0
```

MoleculePopulateProcess

The initial positions of all nonHD species with nonzero initial molecule numbers must be specified with the *MoleculePopulateProcess*. The molecules can be either uniformly or normally distributed within the compartment. By default, without any *MoleculePopulateProcess* parameter definition, molecules are uniformly distributed over the entire compartment. Otherwise if the GaussianSigma is set to a nonzero value, the compartment will be populated according to the Gaussian distribution. *MoleculePopulateProcess* definitions can be seen in Figures 2 (lines 26-28), 5 (lines 13-14) and 6 (line 57). A Python example showing two different species populated at the poles of a rod surface compartment is also listed in Figure 11 with the corresponding output in Figure 12.

```
1  # Example of python scripting to populate molecules at the poles of a rod compartment
2  theSimulator.createStepper('SpatiocyteStepper', 'SS').VoxelRadius = 8e-8
3  # Create the root container compartment using the rod geometry:
4  theSimulator.rootSystem.StepperID = 'SS'
5  theSimulator.createEntity('Variable', 'Variable:/:GEOMETRY').Value = 3
6  theSimulator.createEntity('Variable', 'Variable:/:LENGTHX').Value = 10e-6
7  theSimulator.createEntity('Variable', 'Variable:/:LENGTHY').Value = 2e-6
8  theSimulator.createEntity('Variable', 'Variable:/:VACANT')
9  logger = theSimulator.createEntity('VisualizationLogProcess', 'Process:/:logger')
10 logger.LogInterval = 1
11 logger.VariableReferenceList = [['_', 'Variable:/Surface:A'], ['_', 'Variable:/Surface:B']]
12 populator = theSimulator.createEntity('MoleculePopulateProcess', 'Process:/:populateLeft')
13 populator.VariableReferenceList = [['_', 'Variable:/Surface:A']]
14 populator.OriginX = -1
15 populator.UniformRadiusX = 0.5
16 populator = theSimulator.createEntity('MoleculePopulateProcess', 'Process:/:populateRight')
17 populator.VariableReferenceList = [['_', 'Variable:/Surface:B']]
18 populator.OriginX = 1
19 populator.UniformRadiusX = 0.5
20 # Create the surface compartment:
21 theSimulator.createEntity('System', 'System:/:Surface').StepperID = 'SS'
22 theSimulator.createEntity('Variable', 'Variable:/Surface:DIMENSION').Value = 2
23 theSimulator.createEntity('Variable', 'Variable:/Surface:VACANT')
24 theSimulator.createEntity('Variable', 'Variable:/Surface:A').Value = 500
25 theSimulator.createEntity('Variable', 'Variable:/Surface:B').Value = 500
26 run(100)
```

Figure 11: A Python script to populate molecules at the poles of a rod surface compartment. The file is available in the Spatiocyte source package as 2012.arjunan.chapter.populate.py.

Figure 12: Visualization of molecules populated at the poles of a rod surface compartment. The model is created from the Python script shown in Figure 11.

Origin[X, Y, Z]

Origin[X, Y, Z] is the origin point relative to the compartment center point for a species population. The

molecules may have a uniform or a Gaussian distribution from this point. The range of the point along each axis covering the entire compartment is [-1, 1]. Therefore, the origin is at the center of the compartment if Origin[X, Y, Z] is fixed to [0, 0, 0], the default set of values.

GaussianSigma[X, Y, Z]

GaussianSigma[X, Y, Z] stipulates the sigma value for a Gaussian distributed population from the origin in [x, y, z]-axis, respectively.

UniformRadius[X, Y, Z]

The uniformly distributed normalized population radius from the origin point in [x, y, z]-axis is given by the UniformRadius[X, Y, Z] parameter. Since the default values of UniformRadius[X, Y, Z] and Origin[X, Y, Z] are [1, 1, 1] and [0, 0, 0], respectively, the molecules are spread uniformly within the entire compartment when the parameters are not defined.

ResetTime

To place the molecules at a certain interval after the simulation has started, we can use the ResetTime parameter. This parameter is useful when the positions of a molecule species need to be actively altered after a simulation interval.

DiffusionProcess

The *DiffusionProcess* handles the voxel-to-voxel random walk of diffusing molecules and the collisions that take place between each walk. Examples of the *DiffusionProcess* usage are shown in Figures 2 (lines 11-16 and 39-50), 5 (lines 26-28) and 6 (lines 51-53).

D

In the *DiffusionProcess*, the diffusion coefficient of the molecule species is set with *D*, which has the unit m^2s^{-1}. The default value is 0 m^2s^{-1}.

P

P is an arbitrarily set reaction probability limit of the diffusing species, within the range [0, 1]. The default value is ‘1’, which is sufficient to produce accurate simulations. We can set it to a smaller value to perform reaction-diffusion processes at smaller intervals.

PeriodicBoundaryDiffusionProcess

We can use the *PeriodicBoundaryDiffusionProcess* in place of the *DiffusionProcess* when a molecule species needs to be diffused across periodic two-dimensional surface edges. The surface compartment must be enclosing a cuboid parent compartment. The process overcomes the limitation of setting [XY, XZ, YZ]PLANE of the *Compartment* variable to periodic, which only supports periodic volume edges. It inherits the diffusion coefficient, D and the reaction probability limit, P from the *DiffusionProcess*. Examples of *PeriodicBoundaryDiffusionProcess* in EM and Python are as follows:

```
Process PeriodicBoundaryDiffusionProcess(diffuse) {
  VariableReferenceList [_ Variable:/Surface:A];
  D 0.2e-12; }
```

```
diffuser = theSimulator.createEntity('PeriodicBoundaryDiffusionProcess', 'Process:/:diffuse')
diffuser.VariableReferenceList = [['_', 'Variable:/Surface:A']]
diffuser.D = 0.2e-12
```

DiffusionInfluencedReactionProcess

The *DiffusionInfluencedReactionProcess* is used to execute all second-order reactions comprising two diffusing reactants, or a diffusing and an immobile reactant (Reactant 1 and Reactant 2 are nonHD molecules). Figure 2 (lines 51-60 and lines 67-69) shows several usage examples of *DiffusionInfluencedReactionProcess*. A python example of the process definition is provided below:

Second-Order Reaction, A + B → C

```
binder = theSimulator.createEntity('DiffusionInfluencedReactionProcess', 'Process:/:associate')
binder.VariableReferenceList = [['_', 'Variable:/:A', '-1']]
binder.VariableReferenceList = [['_', 'Variable:/:B', '-1']]
binder.VariableReferenceList = [['_', 'Variable:/:C', '1']]
binder.p = 0.5
```

k

The intrinsic rate constant of the diffusion-influenced reaction is set to *k*. In volume reactions, the relationship between the intrinsic rate constant with the macroscopic rate constant k_{on} is given by $1/k_{on} = 1/k + 1/k_d$, where $k_d = 4\pi DR$ is the maximally diffusion-limited reaction rate, *D* is the diffusion coefficient and *R* is the contact radius (i.e., $2r_v$). The units of *k* for various reaction types are given in Table 2.

p

The absolute reactive collision probability of the reaction is given by *p*. This process requires either the value of *k* or *p*.

Table 2: Units of the rate constant, *k* in *DiffusionInfluencedReactionProcess* (Reactant 1 and Reactant 2 are nonHD) and *SpatiocyteNextReactionProcess* (Reactant 1 and/or Reactant 2 are HD).

Reactant 1	Reactant 2	Product 1	Product 2	*k* (units)
*†Volume	Volume	Volume	Volume	m^3s^{-1}
*†Surface	Surface	Surface	Surface	m^2s^{-1}
*†Volume	Surface	Volume	Volume	m^2s^{-1}
*†Volume	Surface	Surface	Surface	m^3s^{-1}
*†Volume	Surface	Volume	Surface	m^3s^{-1}
*Volume	Surface.VACANT	Surface	None	ms^{-1}
†Volume	None	Surface	None	ms^{-1}
†Volume	None	Volume	None	s^{-1}
†Surface	None	Surface	None	s^{-1}
†Surface	None	Volume	None	s^{-1}

**DiffusionInfluencedReactionProcess*, †*SpatiocyteNextReactionProcess.*

SpatiocyteNextReactionProcess

The *SpatiocyteNextReactionProcess* is used to execute all reactions that can be decoupled from diffusion such as zeroth- and first-order reactions, and second-order reactions that involve two adjoining immobile reactants or at least one HD reactant. Each reaction is performed according to the Next Reaction method (Gibson and Bruck, 2000). Unlike in the *DiffusionInfluencedReactionProcess*, the membrane-adsorption reaction where a HD species binds to the membrane is represented as a first-order reaction (see example below). EM examples of the *SpatiocyteNextReactionProcess* are given in Figure 2 (lines 61-66 and 70-75), while Python examples of zeroth- and first-order (surface-adsorption) reactions are given below:

Zeroth-Order Reaction, Ø → A

```
zero = theSimulator.createEntity('SpatiocyteNextReactionProcess', 'Process:/:create')
zero.VariableReferenceList = [['_', 'Variable:/:A', '1']]
zero.k = 0.01
```

First-Order Surface-Adsorption Reaction, $A_v \rightarrow A_s$

```
uni = theSimulator.createEntity('SpatiocyteNextReactionProcess', 'Process:/:adsorp')
uni.VariableReferenceList = [['_', 'Variable:/:A', '-1']]
uni.VariableReferenceList = [['_', 'Variable:/Surface:A', '1']]
uni.k = 0.01
```

k

The rate constant of the event-driven reaction. For second-order reactions, the units are listed in Table 2. In the case of the intercompartmental surface-adsorption reaction, the unit is in ms^{-1}. For all other first-order reactions the unit is in s^{-1}.

Space[A, B, C]

Sometimes the size of the compartment containing the reacting species is too large and all the molecules within the compartment are HD species. To avoid unnecessarily allocating a large amount of memory to represent the compartment that are unpopulated with any nonHD species, we can override the declared size of the compartment with the variables Space[A, B, C]. SpaceA and SpaceB correspond to the size of the compartment containing the first and second reactants respectively, whereas SpaceC denotes the size of the product compartment. The units of Space[A, B, C] correspond to the dimensions of the respective compartment. By default, the values of Space[A, B, C] are set to zero. Only a nonzero positive value will override the respective compartment size.

VisualizationLogProcess

We can use the *VisualizationLogProcess* to log the coordinates of nonHD species at a specified periodic interval. The *SpatiocyteVisualizer* can load the log file to display the molecules in 3D. Figures 2 (lines 17-20), 5 (lines 9-12) and 6 (lines 54-56 and 73) show some examples of *VisualizationLogProcess* usage.

FileName

FileName is the name of the binary log file. The default name is ‘visualLog0.dat’, which is also the default file name loaded by *SpatiocyteVisualizer*.

LogInterval

The interval for logging the coordinates is determined by LogInterval. The default value is ‘0’, which means that the interval would be set to the smallest diffusion or collision interval of the logged nonHD species. If LogInterval > 0, the log interval will be set to the specified value. The unit of LogInterval is in seconds.

MicroscopyTrackingProcess

The *MicroscopyTrackingProcess* mimics the fluorescent microphotography process by logging the trajectory of nonHD molecules averaged over a specified camera exposure time. It inherits the FileName and LogInterval properties from the *VisualizationLogProcess*. After each LogInterval, the number of times a voxel is occupied by a molecule species is counted. At the end of a given ExposureTime**,** the frequency is averaged over the total number of intervals and logged. Figure 2 (lines 21-25) shows an example of the MicroscopyTrackingProcess definition. A Python example is given below:

```
tracker = theSimulator.createEntity('MicroscopyTrackingProcess', 'Process:/:track')
tracker.VariableReferenceList = [['_', 'Variable:/Surface:MinEE', '2']]
tracker.VariableReferenceList = [['_', 'Variable:/Surface:MinDEE', '3']]
tracker.VariableReferenceList = [['_', 'Variable:/Surface:MinE', '-2']]
tracker.VariableReferenceList = [['_', 'Variable:/Surface:MinDE', '-2']]
tracker.VariableReferenceList = [['_', 'Variable:/Surface:MinE', '-1']]
tracker.FileName = "microscopyLog0.dat"
```

MicroscopyTrackingProcess enables representation of different fluorescent colored subunits within a complex according to the coefficient assigned to each *variable*. In the Python example above, the coefficient of the first variable MinEE is 2, representing two subunits of MinE within the complex MinEE. Similarly for MinDEE, the three subunits (one MinD and two MinE's) are represented by the coefficient 3. Each unique variable with a negative coefficient is assigned a different color during visualization. The first negative variable, MinE, has a coefficient of -2, which means that two subunits from the first positive variable, MinEE, are assigned a unique color of MinE. The second negative variable MinDE also has a coefficient of -2, specifying that two subunits of the second positive variable, MinDEE, is assigned the color of MinDE. The third negative variable MinE has a coefficient of -1, corresponding to the color of the remaining one MinE subunit of the second positive variable MinDEE.

ExposureTime

The simulated camera exposure time is specified by ExposureTime. The default value is 0.5 s.

MeanCount

MeanCount is the maximum number of voxel occupancy frequency before it is averaged. The default value is '0', which indicates that the specified LogInterval or the smallest collision or diffusion interval should be used. In this case, the MeanCount will be ExposureTime/LogInterval. Otherwise if MeanCount > 0, the LogInterval is set to ExposureTime/MeanCount.

IteratingLogProcess

The *IteratingLogProcess* executes multiple simulation runs with different random seeds and logs the averaged physical values of molecules, such as their displacement or survival probability, over the total runs. The values are logged in a file using the comma-separated values (csv) format. By default the process logs the number of available molecules of recorded species at the specified interval periodically.

LogDuration

LogDuration is the total duration of a simulation run (i.e., an iteration).

LogInterval

LogInterval is the interval for logging physical values of molecules within an iteration.

Iterations

The number of simulation runs before the logged values are averaged and saved in the log file is specified by the Iterations parameter.

FileName

The file name of the log file is given by FileName. The default file name is "Log.csv".

SaveInterval

When running many iterations, it is useful to save the logged data in a backup file for quick analysis, or to avoid restarting the runs because of some unexpected failures (e.g., power failure). To this end, a backup

file of the logged values can be saved at the iteration intervals given by Iterations/SaveInterval. The default value of SaveInterval is ‘0’, which indicates that a backup file will not be saved.

Survival

The Survival parameter can be set to ‘1’ to log the survival probability of a molecule species. The default value of the parameter is ‘0’.

Displacement

Set the Displacement to ‘1’ to log the displacement of a molecule species. The default value of Displacement is ‘0’.

Diffusion

If the Diffusion parameter is set to ‘1’, the apparent diffusion coefficient of a molecule species will be logged. The default Diffusion value is ‘0’.

SpatiocyteVisualizer

The *SpatiocyteVisualizer* can be started by executing ./spatiocyte in the Spatiocyte directory. Figure 3 illustrates the *SpatiocyteVisualizer* interface, whereas its features and keyboard shortcuts are listed in Table 3. To change the color of a species, right mouse click on the species and select a desired color. The visualizer can display each species within a specified range in each axis using the bounding feature. Figure 8 displays the output after specifying a set of ranges for the cell membrane. Each displayed frame can be saved into the Portable Network Graphics (PNG) image format. A quick way to create a movie from the saved images is to use the ffmpeg program:

```
$ ffmpeg -i image%07d.png -sameq out.mp4
```

Table 3: *SpatiocyteVisualizer* features and keyboard shortcuts

Feature	**Keyboard shortcut(s)**
Play Forward	Right arrow
Play Backward	Left arrow
Step Forward	Up arrow or Enter
Step Backward	Down arrow or Shift+Enter
Pause/Play	Space
Zoom In	Ctrl++ or Ctrl+= or Page Up
Zoom Out	Ctrl+- or Page Down
Reset View	Ctrl+0 or Home
Rotate along x-axis clockwise	Ctrl+Up Arrow
Rotate along x-axis counter-clockwise	Ctrl+Down Arrow
Rotate along y-axis clockwise	Ctrl+Right Arrow
Rotate along y-axis counter-clockwise	Ctrl+Left Arrow
Rotate along z-axis clockwise	z
Rotate along z-axis counter-clockwise	Z
Translate Up	Shift+Up Arrow
Translate Down	Shift+Down Arrow

Translate Right	Shift+Right Arrow
Translate Left	Shift+Left Arrow
Save current frame as a PNG image	s
Start/Stop recording PNG frames	S

Parameter Tuning Example

Sometimes it is necessary to tune the parameters of a spatially resolved RD model, which usually takes a significant amount of effort and time when done manually. Here we provide a Python script example that automatically generates the visualization log data for a given range of a set of parameters. The user can then view the logged data using *SpatiocyteVisualizer* to select the set of parameters that most closely reproduces the expected spatiotemporal behavior of the simulated molecules. In this particular example, we would like to determine the parameters of a model that can generate clusters of molecules on the membrane. We know that the model consists of the following reactions,

$A + A \rightarrow Ac + Ac$

$A + Ac \rightarrow Ac + Ac$

$Ac \rightarrow A$

where A is a diffusing surface species and Ac is a nondiffusing surface cluster species. The first two reactions have the binding probabilities p_1 and p_2, respectively, while the third reaction has the rate k. We would like to determine the values of p_1, p_2 and k such that several Ac clusters are formed on the membrane. First, we need to create a Python model file that describes the reactions and the diffusion of the molecules, as shown in Figure 13.

```
message('\nrunning: ' + FileName)
theSimulator.createStepper('SpatiocyteStepper', 'SS').VoxelRadius = 0.5
# Create the system compartment:
theSimulator.rootSystem.StepperID = 'SS'
theSimulator.createEntity('Variable', 'Variable:/:LENGTHX').Value = 250
theSimulator.createEntity('Variable', 'Variable:/:LENGTHY').Value = 250
theSimulator.createEntity('Variable', 'Variable:/:LENGTHZ').Value = 20
theSimulator.createEntity('Variable', 'Variable:/:VACANT')
theSimulator.createEntity('Variable', 'Variable:/:XYPLANE').Value = 3
theSimulator.createEntity('Variable', 'Variable:/:YZPLANE').Value = 5
theSimulator.createEntity('Variable', 'Variable:/:XZPLANE').Value = 5
logger = theSimulator.createEntity('VisualizationLogProcess', 'Process:/:logger')
logger.LogInterval = 500
logger.VariableReferenceList = [['_', 'Variable:/Surface:A'], ['_', 'Variable:/Surface:Ac']]
logger.FileName = FileName
# Create the surface compartment:
theSimulator.createEntity('System', 'System:/:Surface').StepperID = 'SS'
theSimulator.createEntity('Variable', 'Variable:/Surface:DIMENSION').Value = 2
theSimulator.createEntity('Variable', 'Variable:/Surface:VACANT')
theSimulator.createEntity('Variable', 'Variable:/Surface:A').Value = 15300
theSimulator.createEntity('Variable', 'Variable:/Surface:Ac').Value = 250
populator = theSimulator.createEntity('MoleculePopulateProcess', 'Process:/:populate')
populator.VariableReferenceList = [['_', 'Variable:/Surface:A'], ['_', 'Variable:/Surface:Ac']]
diffuser = theSimulator.createEntity('PeriodicBoundaryDiffusionProcess', 'Process:/:diffuse')
diffuser.VariableReferenceList = [['_', 'Variable:/Surface:A']]
diffuser.D = 4.3e-3
binder = theSimulator.createEntity('DiffusionInfluencedReactionProcess', 'Process:/:Reaction1')
binder.VariableReferenceList = [['_', 'Variable:/Surface:A','-1']]
binder.VariableReferenceList = [['_', 'Variable:/Surface:A','-1']]
binder.VariableReferenceList = [['_', 'Variable:/Surface:Ac','1']]
binder.VariableReferenceList = [['_', 'Variable:/Surface:Ac','1']]
binder.p = p1
binder = theSimulator.createEntity('DiffusionInfluencedReactionProcess', 'Process:/:Reaction2')
binder.VariableReferenceList = [['_', 'Variable:/Surface:A','-1']]
binder.VariableReferenceList = [['_', 'Variable:/Surface:Ac','-1']]
binder.VariableReferenceList = [['_', 'Variable:/Surface:Ac','1']]
binder.VariableReferenceList = [['_', 'Variable:/Surface:Ac','1']]
binder.p = p2
uni = theSimulator.createEntity('SpatiocyteNextReactionProcess', 'Process:/:Reaction3')
uni.VariableReferenceList = [['_', 'Variable:/Surface:Ac','-1']]
uni.VariableReferenceList = [['_', 'Variable:/Surface:A','1']]
uni.k = k
```

```
run(100000)
```

Figure 13: A Python script to create clusters on a membrane. The file is available in the Spatiocyte source package as 2012.arjunan.chapter.cluster.py.

Next, we execute the Python script shown in Figure 14 by issuing
`$ python 2012.arjunan.chapter.parameter.py`
to run the cluster model multiple times with different parameter values. Each run will generate the visualization log data resulting from the given set of parameters. Finally, we can load and view the log data using *SpatiocyteVisualizer* and select the set of parameters that best captures the expected result. Figure 15 shows an example Python script that loads all the visualization log files within a directory.

```
import os
p1 = [2.2e-6]
p2 = [0.1, 0.2, 0.3]
k = [2.5e-3]
FileName = ''
for x in p1:
  for y in p2:
    for z in k:
      os.system('ecell3-session --parameters=\"{\'FileName\':\'' + FileName + \
          str(x) + '_' + str(y) + '_' + str(z) + '_visualLog0.dat\',\'p1\':' + \
          str(x) + ',\'p2\':' + str(y) + ',\'k\':' + str(z) +'}\" \
          2012.arjunan.chapter.cluster.py')
```

Figure 14: A Python script to run the cluster model multiple times with different parameter values. The file is available in the Spatiocyte source package as 2012.arjunan.chapter.parameter.py.

```
import glob
import os
files = glob.glob('*0.dat')
for i in files:
  print "\nloading file " + i + "..."
  os.system('./spatiocyte ' + i)
```

Figure 15: A Python script to sequentially load multiple visualization log files. The file is available in the Spatiocyte source package as 2012.arjunan.chapter.loadLogs.py.

Concluding Remarks

Building computational models of biochemical processes is usually a demanding task, especially for experimental biologists without modeling experience. This chapter aims to provide a guide on how one can quickly build and simulate spatially resolved biochemical models with the Spatiocyte software. We started with the basic theory of the Spatiocyte method and continued with the installation and simulation procedures. The various modules available to Spatiocyte users were also explained with accompanying model examples.

We plan to continuously develop and improve the Spatiocyte software and user experience. The contents of this guide will also therefore, evolve with the addition of new features and enhancements. The latest

version of this guide will be available along with the Spatiocyte source code, which at the time of writing, is hosted at GitHub. The Spatiocyte website, http://spatiocyte.org also contains the latest information about the Spatiocyte method and software.

In future, we would like to introduce the ability of subunits to polymerize on the membrane and in the cytoplasm. A polymerization strategy using the HCP lattice was proposed recently (Arjunan, 2009). Diffusion of compartments, and molecules with different shapes and sizes are also in the future development plan. Parallel implementation of the Spatiocyte method to run on multi-core architectures and graphics processing units is also being considered. We are also currently working on introducing compartments with complex surface geometries. Spatiocyte users are encouraged to submit feature requests and bug reports, while independent developers can submit their own algorithm modules, code improvements and bug fixes.

Acknowledgments

The author thanks Goh Su Hua for creating the cluster model in the parameter tuning example.

References

1. Arjunan, S. N. V. (2009) Modeling three-dimensional spatial regulation of bacterial cell division. PhD Thesis, Keio University.
2. Arjunan, S. N. V. and Tomita, M. (2009). Modeling reaction-diffusion of molecules on surface and in volume spaces with the E-Cell System. International Journal of Computer Science and Information Security. 3(1): 211–216.
3. Arjunan, S. N. V. and Tomita, M. (2010). A new multicompartmental reaction-diffusion modeling method links transient membrane attachment of E. coli MinE to E-ring formation. Systems and Synthetic Biology 4(1): 35-53
4. Collins, F. C. and Kimball, G. E. (1949). Diffusion-controlled reaction rates. J Colloid Sci 4(4):425–437.
5. Gibson, M. and Bruck, J. (2000). Efficient exact stochastic simulation of chemical systems with many species and many channels. J Phys Chem A 104(9):1876–1889
6. Gillespie, D. T. (1976). A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. J Comput Phys 22(4):403–434
7. Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. J Phys Chem 81(25):2340–2361
8. Dix JA, Verkman AS. (2008). Crowding effects on diffusion in solutions and cells. Annu Rev Biophys. 37:247-63.
9. Hall D, Hoshino M. (2010). Effects of macromolecular crowding on intracellular diffusion from a single particle perspective. Biophysical Reviews. 2(1):39-53.
10. Takahashi, K., Kaizu, K., Hu, B. and Tomita, M. (2004). A multi-algorithm, multi-timescale method for cell simulation. Bioinformatics. 20(4):538–546