2010

Handling External Events Efficiently in Gillespie's Stochastic Simulation Algorithm

Brad Geltz
Virginia Commonwealth University

Handling External Events Efficiently in Gillespie's Stochastic Simulation Algorithm

Brad R. Geltz

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at Virginia Commonwealth University

Director - Dr. James M. McCollum, Assistant Professor

Department of Electrical and Computer Engineering

Virginia Commonwealth University

December 2010

DEDICATION

To my parents, Becky and Jeff,

my brother, Bryan

and my grandmother, Elaine

for their love, support, and encouragement

ACKNOWLEDGMENTS

First, I wish to thank my advisor Dr. James M. McCollum for his continued support and inspiration throughout my graduate career. I am fortunate to have such a great source guidance and understanding to drive my research. I would also like to thank Dr. Alen Docef, Dr. Kayvan Najarian, and Dr. Stephen Fong for serving on my thesis committee.

Second, I would like to thank Oak Ridge National Laboratory for helping support this work. A portion of this research was conducted at the Center for Nanophase Materials Sciences, which is sponsored at Oak Ridge National Laboratory by the Division of Scientific User Facilities, U.S. Department of Energy. I would like to thank Dr. Chris Cox (Univ. of Tennessee - Knoxville), Dr. Michael L. Simpson (ORNL-CNMS), and Roy Dar (ORNL-CNMS) for their helpful discussions and review of this work.

Third, I would like to thank my colleague Thomas V. Trimeloni for his aid in generating models and data that depict the impact of the EEHDM.

Finally I would like to thank my family and friends for believing in me. Without their faith and backing, I would never have been able to achieve this feat. I am truly grateful.

ABSTRACT

Gillespie's Stochastic Simulation Algorithm (SSA) provides an elegant simulation approach for simulating models composed of coupled chemical reactions. Although this approach can be used to describe a wide variety biological, chemical, and ecological systems, often systems have external behaviors that are difficult or impossible to characterize using chemical reactions alone.

This work extends the applicability of the SSA by adding mechanisms for the inclusion of external *events* and external *triggers*. We define *events* as changes that occur in the system at a specified time while *triggers* are defined as changes that occur to the system when a particular condition is fulfilled. We further extend the SSA with the efficient implementation of these model parameters.

This work allows numerous systems that would have previously been impossible or impractical to model using the SSA to take advantage of this powerful simulation technique.

Keywords: Stochastic simulation, Modeling biochemical systems, Gillespie algorithm, Systems biology, Efficient event handling

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The way systems are conceptualized, constructed, and designed has been revolutionized by the integration of computer modeling and simulation into the design process. Computer models of analog and digital circuitry enable systems to undergo extensive verification and testing prior to implementation, significantly reducing costs [1]. Computer models of physical structures allows estimations of how buildings, roads, and bridges will perform under certain stresses over time, allowing the design to be optimized for particular environments [2].

The development of software tools that will revolutionize the design and analysis of biological systems is currently underway. A variety of mathematical techniques exist for representing biological systems, including directed graphs, differential equations, Bayesian networks, Boolean networks, molecular dynamics and stochastic methods. Each method trades off computational performance for model detail [3].

The focus of this work is on improving the performance and utility of stochastic simulation in chemically reacting systems. Recent research has shown that fluctuation in the populations of chemical species can significantly impact the performance and evolution of biochemical system [4]. Ordinary differential equations are typically insufficient for modeling systems affected by noise. Molecular dynamics simulations provide too much detail, track-

ing the individual positions of each molecule as they move and interact through the system. Stochastic simulation, first pioneered by Gillespie [5, 6], has been shown to be an effective tool for modeling systems of this type [7].

Given a set of chemical reactions and a list of initial populations of chemical species, Gillespie's Stochastic Simulation Algorithm (SSA) predicts potential time evolutions of spatially homogeneous chemical systems by stochastically predicting the time and type of each individual reaction event. The computational acceleration of the SSA is an area of active research. Algorithms that are mathematically equivalent to the SSA, while enhancing computational performance of the SSA, include the *Next Reaction Method* (NRM) by Gibson and Bruck [8], the *Optimized Direct Method* (ODM) by Cao et al. [9], the *Logarithmic Direct Method* (LDM) by Li and Petzold [10], and the *Sorting Direct Method* (SDM) by McCollum et al. [11]. Recent research has shown that the SDM is among the fastest of current SSA implementations [10, 12].

Current stochastic simulation algorithms limit model builders in that the entire system must be described using chemical equations alone. Massive changes in system state (e.g. cell division where the contents of the cell are binomially divided between two separate physical structures) can be easily represented in this manner. In order to correctly model systems that behave in this fashion, model builders are forced to write custom SSA implementations that rarely take advantage of the acceleration offered by the SDM or the LDM.

This work builds upon the SDM by creating a unified framework for handling these additional model constructs. We describe these parameters in terms of *events* and *triggers*. Here we define *events* as system changes that occur at specific times while *triggers* are system changes that occur when specific simulation states are reached. An SSA implementation known as the *Event Handling Direct Method* (EHDM) is proposed for efficiently simulating models of this type.

This thesis is organized as follows: the second chapter discusses the background information on relevant SSAs. The reader will be introduced to concepts necessary to understand the extensions the EHDM provides. Chapter 3 discusses our modifications to the SDM to produce the EHDM. Chapter 4 discusses a computationally efficient implementation of the EHDM. Chapters 5 discusses the performance results from this implementation.

This work strives to allow system biologists the utmost flexibility during model construction. This allows for numerous new research problems to be investigated with SSA that were previously inapplicable.

Chapter 2

Background

Biological modeling involves the generation of a mathematical model that accurately represents the state of a system at any given time. An example of this would be the population of a particular fish species in a lake over a fixed time period. If one sampled the population at fixed intervals, it would seemingly fluctuate randomly about an average value. That average value depends on the many factors that affects the fish population (e.g. how big the lake is, how much food is available, how many predators the fish may have, etc.).

Stochastic simulation algorithms are particularly well suited at simulating systems with "random" fluctuations providing a "random walk" time evolution of the reacting system. The SSA works by utilizing the Monte Carlo method to produce random walks of a system [13]. This is necessary as population trajectories in an actual experiment will have random fluctuations (noise).

To illustrate the Monte Carlo method, take the example problem of calculating Pi (π). Recalling that Pi is the ratio between the circumference and the diameter of a circle, we can estimate its value utilizing the Monte Carlo method. By generating random coordinate (X, Y) pairs between 0 and 1 and evaluating them with the circle equation as shown in

Equation 2.1, we see that we can compare the resulting radius to that of the unit circle.

$$X^2 + Y^2 = r^2 \tag{2.1}$$

This example works by comparing the number of times a random pair generates an r value less than 1 versus not. When an r value is generated inside the unit circle, a value is incremented. This is done in a similar fashion for those outside the circle. After a number of iterations, the ratio of the number of pairs that generated results inside the circle to the total number of iterations will reasonably approximate $\pi/4$. We are only approximating the value of a quarter Pi here since the random values we generate produce values in the first quadrant only.

SSAs work in a similar fashion with regard to the Monte Carlo method. Random numbers are utilized in the selection of which reaction executes. This allows for proper simulation to occur and will be discussed in detail in the next sections.

2.1 First Reaction Method

The *First Reaction Method* (FRM) was the first algorithm developed for performing stochastic simulations of biochemical systems [6]. The FRM parameters include a set of species m with initial integer value populations X_i $(t = 0)$ with a set of governing reactions R_j consisting of rate constants (real values) k_j, and reactant coefficients $r_{j,i}$ and product coefficients $p_{j,i}$ that express the stoichiometry of the system. Execution of the FRM proceeds as shown in Figure 2.1.

The propensity α of each reaction directly corresponds to how often that particular reaction is executed during simulation. This propensity value is used in determining the putative time τ for each reaction. The propensity is calculated by multiplying the stochastic rate constant k_j by the number of ways a particular reaction can occur. The number of ways

A. $\forall\{m\}$: Let X_i = Initial value (specified in model)

B. $\forall\{R_j\}$: Let k_j = Initial value

C. $t = 0$

D. $\forall\{R_j\}$: Let $\alpha_j = f(X_i(t), k_j, r_j)$

E. $\forall\{R_j\}$: Let $\tau_j = \dfrac{-ln(\lambda_j)}{\alpha_j}$

F. Let μ = Array index of Minimum(τ_j)

G. $\forall\{S_i\}$: Let $X_i(t+\tau) = X_i(t) - r_{\mu,i} + p_{\mu,i}$

H. Let $t = t+$ Minimum(τ_j)

I. if $(t < T)$ then Goto D

Figure 2.1 FRM Algorithm Overview

a reaction occurs is specified in model creation. This is shown in Equation 2.2.

$$f(X(t), k, r) = k \prod_{i=1}^{m} \prod_{j=1}^{r_i} \frac{X_j(t)}{j} \tag{2.2}$$

Suppose we have the following (chemical) equation:

$$A + B \Rightarrow C \quad , \quad k_1 \tag{2.3}$$

The propensity of this equation depends on the species populations of A (X_A) and B (X_B), and also depends on the reaction's rate constant k_1. By multiplying the dependencies together we can calculate the propensity (α_1). Thus:

$$\alpha_1 = X_A * X_B * k_1 \tag{2.4}$$

The putative time (τ_j) represents the time needed for the reaction to occur based on the current status of the system. This is accomplished by generating a uniform random number (or URN) between 0 and 1, scaling it to the exponential distribution, and multiplying it by the propensity. The equation for doing such is shown below:

$$\tau_j = -\frac{log(\text{URN})}{\alpha_i} \tag{2.5}$$

By always selecting the reaction with the smallest putative time, we allow for random noise in which reactions execute since the reaction with the smallest τ value will vary during execution as a function of the propensity and the exponentially distributed random number.

While the FRM accurately simulates systems, Gillespie noted that at each time step j exponentially distributed random numbers must be generated. This is a computationally intensive operation for two reasons. First because it requires the generation of a URN, and second because it requires the evaluation of the log function.

2.1.1 Example System

Consider a model with the following species:

- Gene
- mRNA
- Protein
- Dimer

The species interaction is specified in Table 2.1.

In this example system, Genes produce mRNA at a rate of k_1. mRNA produces Proteins at a rate k_3 and so on. Reactions 2 and 4 represent decay of the corresponding species (via

Table 2.1 SSA Example System

#	***Reaction***	***Rate Constant***
1	Gene ⇒ Gene + mRNA	$k_1 = 10$
2	mRNA ⇒ *	$k_2 = 1$
3	mRNA ⇒ mRNA + Protein	$k_3 = 10$
4	Protein ⇒ *	$k_4 = 1$
5	Protein + Protein ⇒ Dimer	$k_5 = 1$
6	Dimer ⇒ 2*Protein	$k_6 = 1$

the asterisk). The initial populations for all the species save the Gene are 0. The Gene population begins at 1. Reactions 5 and 6 are used to represent the dimerization of the protein.

The rate constants (k_i) for each reaction are shown to the right of the equations. Utilizing these values, calculating propensities for the system described above proceeds as follows:

$$\begin{aligned} \alpha_1 &= X_{\text{Gene}} * k_1 \\ &= 1 * 10 = 10 \end{aligned} \tag{2.6}$$

$$\begin{aligned} \alpha_2 &= X_{\text{mRNA}} * k_2 \\ &= 0 * 1 = 0 \end{aligned} \tag{2.7}$$

$$\begin{aligned} \alpha_3 &= X_{\text{mRNA}} * k_3 \\ &= 0 * 10 = 0 \end{aligned} \tag{2.8}$$

$$\begin{aligned} \alpha_4 &= X_{\text{Protein}} * k_4 \\ &= 0 * 1 = 0 \end{aligned} \tag{2.9}$$

$$\begin{aligned} \alpha_5 &= 2 * X_{\text{Protein}} * k_5 \\ &= 2 * 0 * 1 = 0 \end{aligned} \tag{2.10}$$

$$\begin{aligned} \alpha_6 &= X_{\text{Dimer}} * k_6 \\ &= 0 * 1 = 0 \end{aligned} \tag{2.11}$$

These equations represent the very first execution step for the FRM, and in being the first step only Reaction 1 will execute (since all other populations are initialized to zero). As all propensities except that of Reaction 1 evaluate to 0, Reaction 1 executes. In doing so, the mRNA population is incremented by its corresponding coefficient (in this case 1), and the Gene population remains the same (this is why it is also present on the right side of Reaction 1).

The next reaction step illustrated how noise is present in the output data. Since now we have a Gene population and a mRNA population that our both greater than zero, the propensities will also evaluate to non-zero values (and thus the putative times will also be non-zero). Now the simulator has 3 possible reactions that can execute (Reactions 1, 2, and 3). The one that is chosen is now dependent on the calculated putative time shown in Equation 2.5. This process of determining which reaction to execute, executing it, and updating the corresponding species populations continues until the end of simulation time.

2.2 Direct Method

The *Direct Method* (D) was another SSA designed by Gillespie to overcome the shortcomings of the FRM [5, 6]. He noted that at each iteration of the FRM, j (the number of reactions) random numbers must be generated. These numbers must then be scaled to the exponential distribution. Both operations make it a computationally intensive algorithm. These limi-

tations became more apparent as the number of species/reactions increased. It effectively limited the maximum number of both parameters that could be simulated in a reasonable amount of time.

To reduce the algorithm's complexity, Gillespie modified the FRM to require the need of only two random numbers for any size system. This also reduces the amount of operations that must be done in order to scale the number into the desired distribution.

The DM's parameters (similar to that of the FRM) include set of species m with initial integer value populations X_i $(t = 0)$ with a set of governing reactions R_j and rate constants (real values) k_j, and reactant coefficients $r_{j,i}$ and product coefficients $p_{j,i}$ that express the stoichiometry of the system.

The DM differs from the FRM in the propensity calculation and reaction selection steps. Instead of requiring j random numbers to determine reaction selection, the DM overcomes this by first summing all the propensities. This value is then multiplied by the first of the two required random numbers. The resulting value is decremented by the individual reaction propensities until it is less than 0. This will still choose reactions with higher propensities more often since they will decrement the scaled propensity total by larger values (and thus are more likely to be less than zero).

The mathematical description of the DM's execution is shown in Figure 2.2. Here, λ_1 and λ_2 represent the two required random numbers. Here, the reaction steps are similar to that of the FRM. The algorithm begins by calculating the propensities for all of the reactions (Step A). The function (f) that calculates this value is shown in Equation 2.2. Next, the first random number is used in the calculation of the total putative time (τ).

Step F shows how the loop proceeds that decides which reaction is chosen for execution. Here μ is the reaction that will be executed. This description shows that the total propensity from reaction 1 to reaction μ must be less than the total system propensity (multiplied by

A. $\forall\{m\}$: Let X_i = Initial value (specified in model)

B. $\forall\{R_j\}$: Let k_j = Initial value

C. $t = 0$

D. $\forall\{R_j\}$: Let $\alpha_j = f(X_i(t), k_j, r_j)$

E. Let $\tau = \dfrac{-ln(\lambda_1)}{\sum_{j=1}^{n} \alpha_j}$

F. Let μ satisfy $\sum_{j=1}^{\mu} \alpha_j \ <= \ \lambda_2 \sum_{j=1}^{n} \alpha_j \ < \ \sum_{j=\mu+1}^{n} \alpha_j$

G. $\forall\{S_i\}$: Let $X_i(t+\tau) = X_i(t) - r_{\mu,i} + p_{\mu,i}$

H. Let $t = t + \tau$

I. if $(t < T)$ then Goto D

Figure 2.2 DM Execution Overview

λ_2) which must be less than the remainder of the total propensity from $\mu + 1$ to the end of the number of reactions.

Steps G - I show identical behavior to that of the FRM. Species populations are updated according to the reactions (Step G). Next, the system time is incremented by the total putative time (Step H). Finally, unless the current time (t) is greater that the end of simulation time (T) continue iterating.

2.3 Sorting Direct Method

The *Sorting Direct Method* (SDM) was developed by McCollum et al. in 2006 [11]. Several of the advancements incorporated in the SDM were based on other advanced SSA implementations, the *Next Reaction Method* (NRM), and the *Optimized Direct Method*.

The NRM introduces the concept of a *dependency graph* for updating propensity values [8]. Instead of calculating new propensity values at each iteration through the SSA, the NRM reduces the number of calculations by only recalculating certain propensities. At initialization, the NRM determines which reactions propensities would be affected by another reaction executing. This list shows which reaction propensities "depend" on other reactions executing, hence the term *dependency graph*. An example of this based on the model in Table 2.1 is shown in Table 2.2.

Table 2.2 Dependency Graph Example

Reaction #	***Reaction***	***Depends***	***Effects***	***Update***
1	Gene ⇒ Gene + mRNA	Gene	mRNA	2,3
2	mRNA ⇒ *	mRNA	mRNA	2,3
3	mRNA ⇒ mRNA + Protein	mRNA	Protein	4,5
4	Protein ⇒ *	Protein	Protein	4,5
5	Protein + Protein ⇒ Dimer	Protein	Protein, Dimer	4,5,6
6	Dimer ⇒ 2*Protein	Dimer	Dimer,Protein	4,5,6

In addition to the graph, reaction execution times (potential) were stored in a heap structure sorted by potential execution times. This further reduced the time required per iteration by attempting to minimize the time it takes to select the reaction from its underlying

data structure.

The ODM strove to further decrease time taken during the reaction selection step by altering the order in which reactions were stored in a data structure. By altering the structure such that reactions that execute more frequently are located at the beginning of the structure, overall simulation time decreased. To implement these enhancements, the ODM pre-simulated a number of iterations and monitored the individual frequencies at which reactions execute. This information determines the sorting order.

The work outlined by McCollum et al. depicts how the the techniques utilized in the previous SSA implementations can be further enhanced to decrease overall simulation time [11]. The SDM first improves upon the ODM by altering the way in which reactions that execute with high frequency on the top of the data structure (i.e. the reaction search order, RSO_j). Instead of requiring pre-simulation to accomplish this, the SDM employs a "bubble-sort"-like technique. The structure is initialized to a default order. Upon execution, the reaction in question is swapped with the reaction immediately preceding it. During execution, reactions that execution with a high frequency will be "bubbled-up" toward the beginning of the structure while those that do not will be "bubbled-down".

This sorting technique also accounts for dynamic changes in the system during simulation. For example, suppose that during simulation a reaction executes very often for a limited amount of time, then slows. If during pre-simulation with the ODM the slow down did not occur, it will remain at the top of the structure indefinitely. The SDM's approach will allow the reaction to be moved down in the structure regardless of when the execution frequency changes.

The SDM algorithm is formally stated in Figure 2.3.

A. $\forall\{m\}$: Let X_i = Initial value (specified in model)

B. $\forall\{R_j\}$: Let k_j = Initial value, $RSO_j = \{1...j\}$ (default ordering)

C. $\forall\{R_j\}$: Generate dependency graph DG_j

D. $\forall\{R_j\}$: Let $\alpha_j = f(X_i(t), k_j, r_j)$

E. $t = 0$

F. Goto Step H

G. $\forall\{R \text{ in } DG_j\}$: Let $\alpha_j = f(X_i(t), k_j, r_j)$

H. Let $\tau = \dfrac{-ln(\lambda_1)}{\sum_{j=1}^{n} \alpha_j}$

I. Let μ satisfy $\sum_{j=1}^{\mu} \alpha_j \ <= \ \lambda_2 \sum_{j=1}^{n} \alpha_j \ < \ \sum_{j=\mu+1}^{n} \alpha_j$

J. $\forall\{S_i\}$: Let $X_i(t+\tau) = X_i(t) - r_{\mu,i} + p_{\mu,i}$

K. Let $t = t + \tau$

L. if $(RSO_\mu \neq 0)$ Swap $(RSO_{\mu-1}$, $RSO_\mu)$

M. if $(t < T)$ then Goto G

Figure 2.3 SDM Algorithm Overview

2.4 Recent Research

SSA implementations have been modified to include events and triggers in recent research. As previously mentioned, these implementations rarely take advantage of the optimizations available in modern algorithms. An example of such research was performed by Swain et.

al. in 2002 [14]. The experiments conducted by Swain utilized a modified version of the NRM to incorporate events. Their simulations show the impact of different types of noise on a system.

In these experiments, the researchers seek to show that variations in gene expression consist of two components: intrinsic and extrinsic noise. The intrinsic component is derived from the stochasticity of the reactions contained within their model, while the extrinsic component deals with the populations of species varying expression levels. To simulate these two components, Swain developed a simulation modeled after *Escherichia Coli* containing gene replication and cell division.

They model cell division using a stochastic approach based on the SSA. To determine whether or not cell division has occurred, their simulator takes into account the current number of proteins in the system along with a certain probability that division will occur.

Their implementation of cell division could be easily described in terms of a trigger in the EHDM and more efficiently simulated using the methods outlined in this work. By having a unified framework for such parameters, numerous other functions can be examined for their impact on a simulation (not just cell division).

Chapter 3

Implementing Events and Triggers

As mentioned in the introduction, the addition of *events* and *triggers* seeks to increase the range and complexity of models that can be simulated using SSA. By eliminating the need for events and triggers to be described within the context of reactions we accomplish two things. First, we allow for these desired effects to occur exactly when they should during simulation as opposed to when they are stochastically executed in the reaction list. Second, we refine the way models are generated by separating these effects from the reactions allowing for a more concise design.

We define an *event* as an action that occurs in a system at a specified time. Biologically speaking, this could represent a temperature being changed, the addition of a chemical, rising and setting of the sun, etc. Similarly, we define *triggers* as actions that occur in a system immediately after their condition is satisfied based on the states of the population.

Consider a set of events E_k. Each event occurs at a model specified time T_k, which is stored in sorted order based on this time. Each event in the set also has a corresponding action A_k that is executed at time T_k. The event's action has various abilities (e.g. modifying species populations, rate constants of reactions, etc.). The variables available for modification through the action are only limited by what was described in the model file. Since during

construction of the model numerous other variables are available for modification, a multitude of other actions can be created.

Triggers can be described in a similar fashion. There are a set of triggers G_l, each of which has an action a_l that is executed when a specific condition C_l is met. Here conditions refer to the state of various variables in the simulation. The actions of triggers can modify the same set of variables available to event actions (i.e. the described variables in the model file). Our implementation also allows for the conditions to check whether or not another trigger's (or event's) action has fired (managed through flags).

This work introduces the Event Handling Direct Method (EHDM) that serves to implement our above description of events and triggers along side the SDM. An outline of the algorithm is shown in Figure 3.1.

In words, Figure 3.1 begins by examining the set of triggers T_l. If any of the l trigger's conditions are met, execute their respective action and reevaluate the set starting at the beginning. This is necessary since a dependency may exist between the action and condition of different triggers.

Care must be taken in trigger construction such that an infinite loop is not introduced. During model construction, if it becomes necessary to introduce a dependency between two triggers their actions must be constructed such that the simulator can escape from the loop of checking and executing the first trigger and checking and executing the second trigger. An infinite loop will occur if the first trigger action makes the second trigger's condition satisfied and the second trigger action makes the first trigger's condition satisfied.

Once all triggers are repeatedly checked until no conditions are satisfied, normal SSA execution can begin. The first two steps of the SDM SSA remain unchanged. The propensity values along with the next potential reaction time $(t + \tau)$ are calculated.

Step 4 of the EHDM serves to implement events. It begins by examining the sorted list

1. $\forall\{m\}$: Let X_i = Initial value (specified in model)
2. $\forall\{R_j\}$: Let k_j = Initial value, $RSO_j = \{1...j\}$ (default ordering)
3. $\forall\{R_j\}$: Generate dependency graph DG_j
4. $\forall\{R_j\}$: Let $\alpha_j = f(X_i(t), k_j, r_j)$
5. $\forall\{G_l\}$: if(C_l == true)
   (a) Execute a_l
6. (SSA Step A) $\forall\{R_j\}$: Let $\alpha_j = f(X_i(t), k_j, r_j)$
7. (SSA Step B) Let $\tau = \dfrac{-ln(\lambda_1)}{\sum_{j=1}^{n} \alpha_j}$
8. $\forall\{E_k\}$: if($T_k <= t + \tau$ && hasNotExecuted(A_k)) then
   (a) Execute A_k
   (b) Let $t = T_k$
   (c) Goto Step 5
9. (SSA Step C) Let μ satisfy $\sum_{j=1}^{\mu} \alpha_j \ <= \ \lambda_2 \sum_{j=1}^{n} \alpha_j \ < \ \sum_{j=\mu+1}^{n} \alpha_j$
10. (SSA Step D) $\forall\{S_i\}$: Let $X_i(t + \tau) = X_i(t) - r_{\mu,i} + p_{\mu,i}$
11. (SSA Step E) Let $t = t + \tau$
12. (SSA Step F) if ($t < T$) then Goto 5

Figure 3.1 EHDM Execution Overview

of events E_k. A comparison is made between the event's execution time T_k and the next potential reaction time $(t + \tau)$. If the event in question has an execution time E_k that is less than the next potential reaction time, the event would occur before the next reaction executes. Thus, if the condition is satisfied the event's action A_k is executed and the system time is set to the event's execution time T_k as opposed to the next reaction time.

At this point, the event has taken the place of the next reaction to execute and the algorithm must proceed to Step 1. If no event should execute based on the comparison, SSA execution continues normally. A reaction is selected for execution R_μ based on the satisfaction of Step 5. The species affected by that reaction and the system time are updated.

This process continues until the end of simulation time T. Examples that illustrate the effectiveness of the EHDM are discussed in Chapter 5 while efficiency considerations in the implementation of events and triggers are discussed in the next chapter.

The example depicted in Table 3.1 shows a simplistic model with a single event and a single trigger. During simulation, this model first checks the trigger conditions. If at any time the protein population exceeds 100, it is reset to 0. Similarly, at simulation time 20.0, the event's action is executed to reset the protein population. This example is merely meant to illustrate the functionality of the algorithm. More complicated cell functions are shown in Chapter 5.

3.1 Simple Examples

Figures 3.2 through 3.4 illustrate examples of how events and triggers would function in a simple model. Figure 3.2 shows simulation results from the system described in Figure 3.1 with the events and triggers removed. The steady state populations of the Protein, mRNA, and Gene species are 100,10, and 1 respectively. Figure 3.2 shows the random fluctuations of the population levels over time with the average value for each species plotted as a horizontal

Table 3.1 SSA Example System (with an Event/Trigger)

#	***Reaction*** (R_j)	***Rate Constant*** (k_j)
1	Gene $\Rightarrow$ Gene + mRNA	$k_1 = 10$
2	mRNA $\Rightarrow$ *	$k_2 = 1$
3	mRNA $\Rightarrow$ mRNA + Protein	$k_3 = 10$
4	Protein $\Rightarrow$ *	$k_4 = 1$
#	***Trigger Condition*** (C_l)	***Trigger Action*** (a_l)
1	$X_{\text{Protein}} > 100$	$X_{\text{Protein}} = 0$
#	***Event Execution Time*** (T_k)	***Trigger Action*** (A_k)
1	$T_1 = 20.0$	$X_{\text{Protein}} = 0$
2	$T_2 = 50.0$	$R_3 = 0$

line along with the numeric value above the line.

Figure 3.3 show the same model shown in Figure 3.2 with the inclusion of two events. The first of which reduces the Protein population to 0 when the simulation time reaches 20 (similar to the model shown in Table 3.1). The second turns off the translation reaction (mRNA producing Protein) when the simulation time reaches 50. The first event is reflected in the population trajectory by a sharp spike to 0 in the Protein population at time 20. The second is shown by the slow decline of the Protein population to 0. This occurs since the protein decay reaction is still active and there are still Proteins in the system.

Figure 3.4 shows the model described in Figure 3.2 with the inclusion of a trigger. The triggers condition checks to see whether or not the Protein population has exceeded 100. If it has, the population is immediately reset to 0. This is shown in the population trajectory by numerous spikes to 0 during simulation. Due to the rapid fluctuation in Protein levels

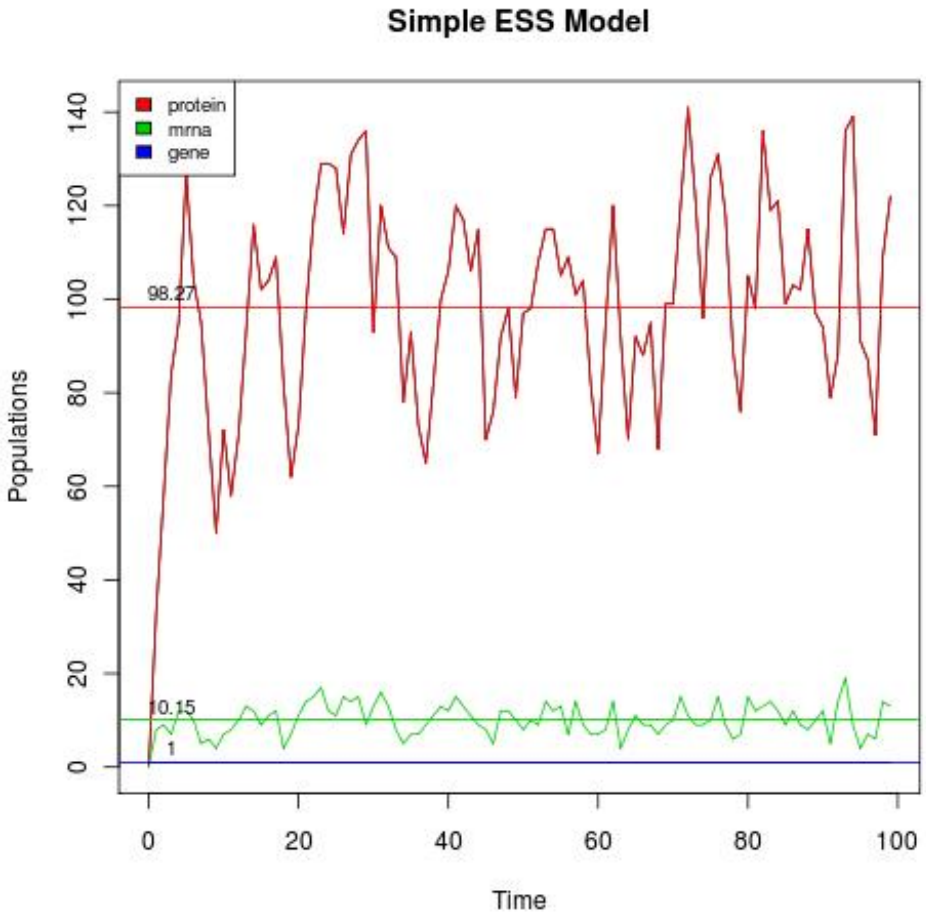

Figure 3.2 Simple ESS Model Population Trajectories

and the Figure's low resolution, it appears that the trigger only reduces the population to a value greater than zero. This is a limitation of the software utilized in generating the plot (the R programming language).

3.2 Real Life Examples

Figures 3.5 through 3.9 show the impact of events and triggers on real life models.

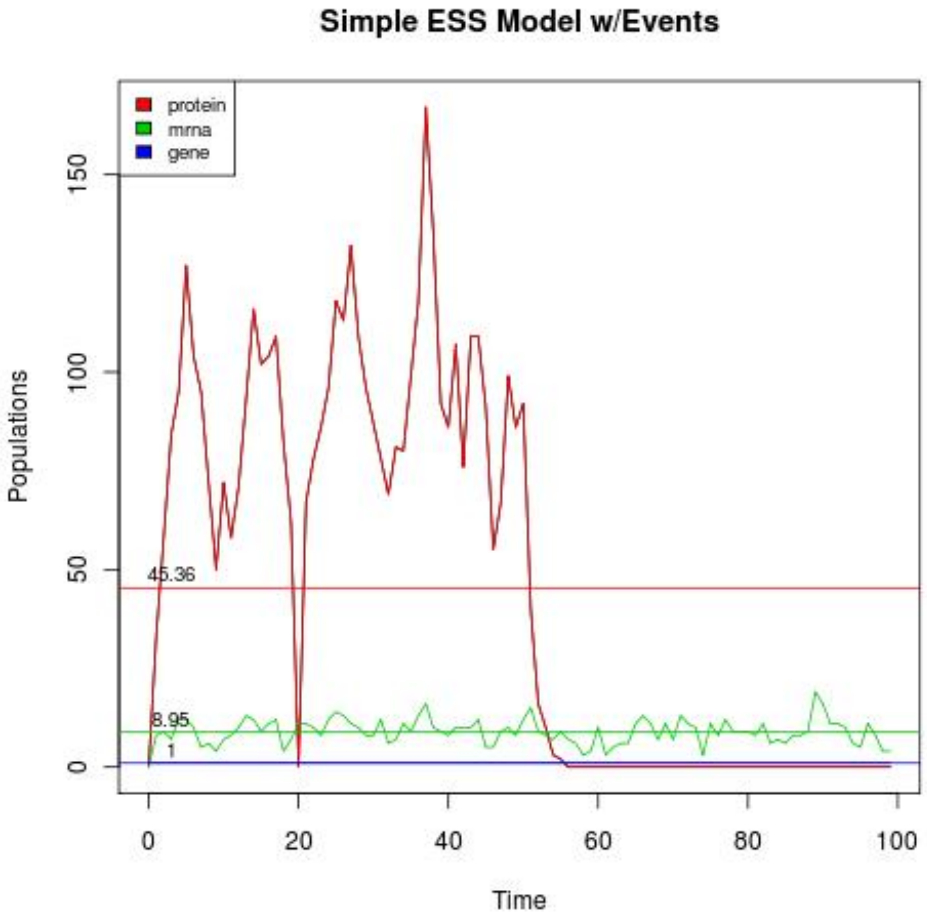

Figure 3.3 Simple ESS Model Population Trajectories (w/Events)

3.2.1 SIR Model

A model that is commonly found in ecological models of infectious disease, the SIR (or Susceptible, Infected, and Recovered) system, can be utilized to determine the an infected population level based on the current size of the population, rate of infection, and rate of recovery. When described under the current SSA context, the models simulate years at a time. During the course of simulation, seasonal and other external events are incorporated to give the most complete data possible.

Figure 3.5 shows the model described in [15] that models the effects of whether or not

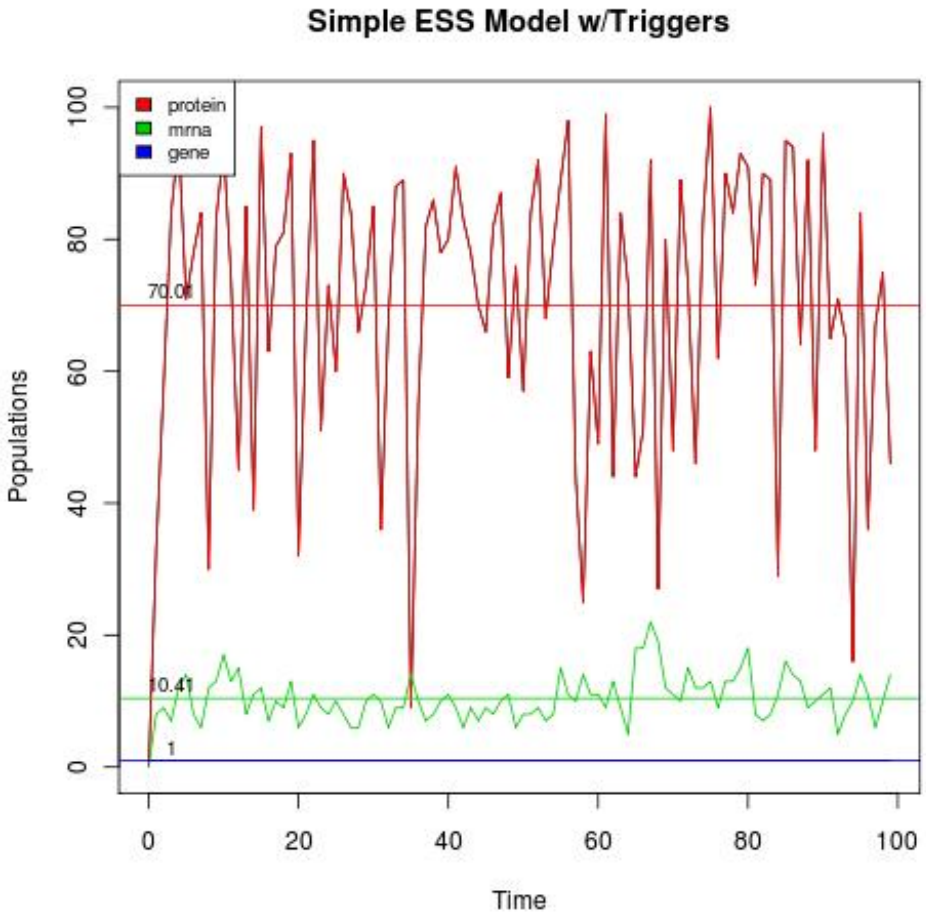

Figure 3.4 Simple ESS Model Population Trajectories (w/Triggers)

school is in session on the spread of whooping cough. Here the rate of exposure can be described in a piecewise function with school being in session (or not) and the ratio of infected individuals (vs. the total population) as parameters. The author simulated the effect of events and triggers by simple adding noise to his system. Modeling with the EEHDM allows for periodic events to represent the rate constant that governs disease exposure. Figure 3.6 shows an excerpt of this system. The population of infected individuals dips slightly between days 275 and 365, depicting the effect of school being out of session. Ranges outside these values show school being in session and its effect on the individuals infected. Here, the use

of the EEHDM allows the model to be accurately described while a typical implementation would have limitations on implementing the seasonal effects, etc.

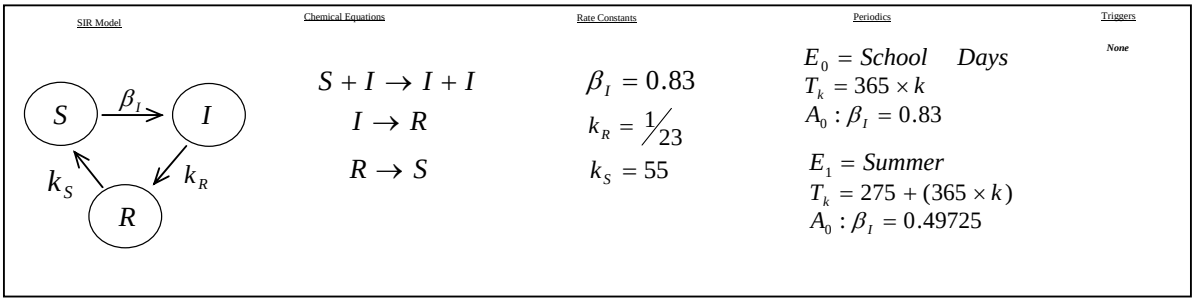

Figure 3.5 SIR Model

3.2.2 Protein Expression/Cell Doubling

Figure 3.7 shows a model based on the work in [16] combining protein levels over many generations to be contained in a single simulation. This model describes a single gene switching between two states: active and inactive (governed by rate constants k_f and k_r). While in the active state, proteins are produced from genes at a rate k_p. While inactive, nothing is produced. The produced protein will also decay at a rate of γ_p.

Three separate models were generated to illustrate the impact of the EEHDM on the model. First, a periodic event was utilized to accomplish the protein division once every 80 minutes (the doubling time of the bacterium). The population trajectory from this model is shown in Figure 3.8. In order to accentuate the effect of the cell division event the protein population was artificially inflated by reducing γ_p.

The second model shows cell division implemented as a binomial split (also shown in Figure 3.7). Previously, the split was accomplished by dividing the protein level by two rather than a binomial split. Here, triggers could also be utilized to implement cell division when the protein population reaches a certain level. The second simulation contains cell

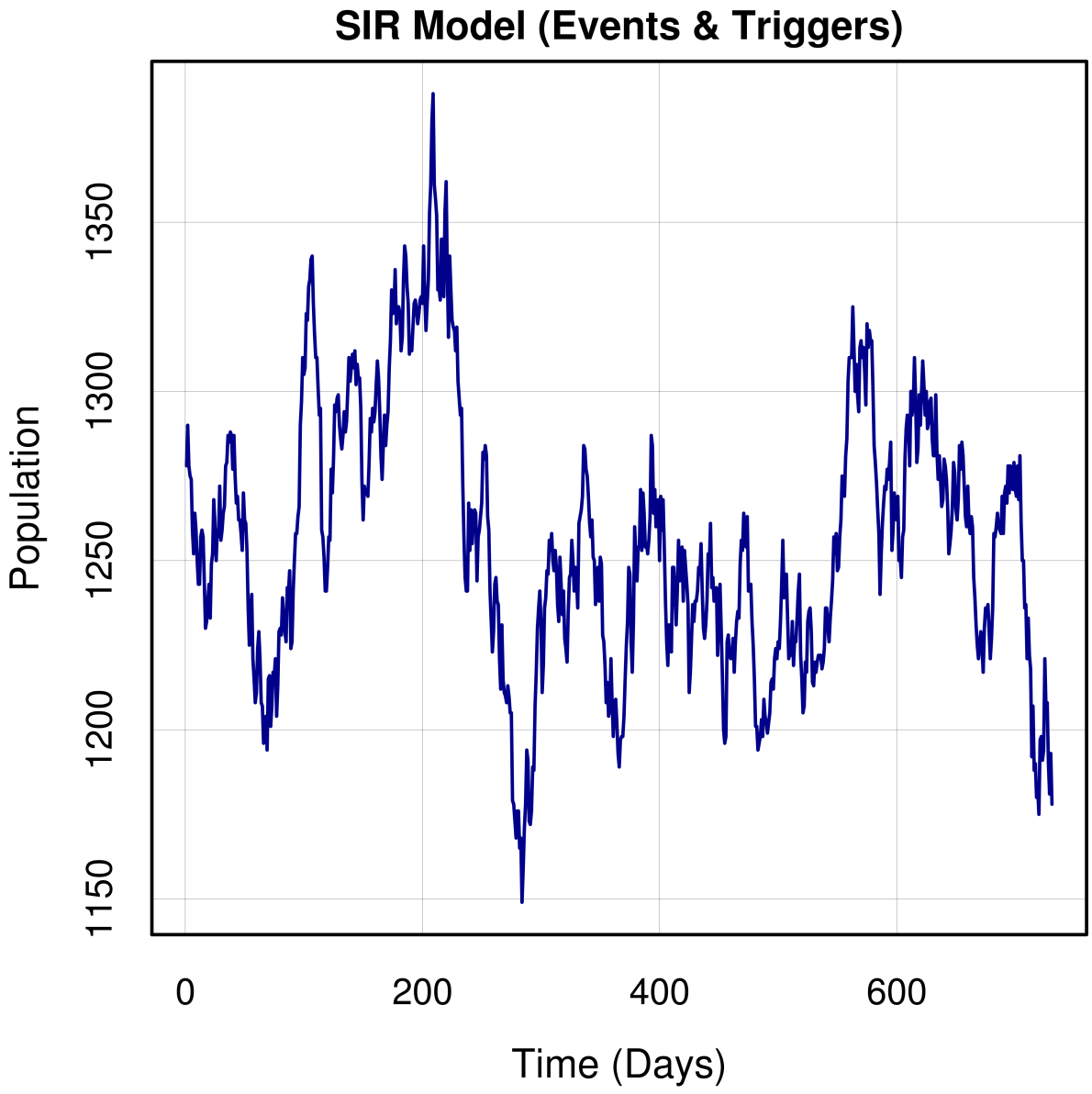

Figure 3.6 SIR Model Population Trajectory

division implemented with a trigger when the protein population exceeds 100. These results are shown in Figure 3.9. Both implementations of cell division are shown by the sharp drop in protein levels. These models provide a minimal introduction into the impact of events and triggers have on accurate model construction.

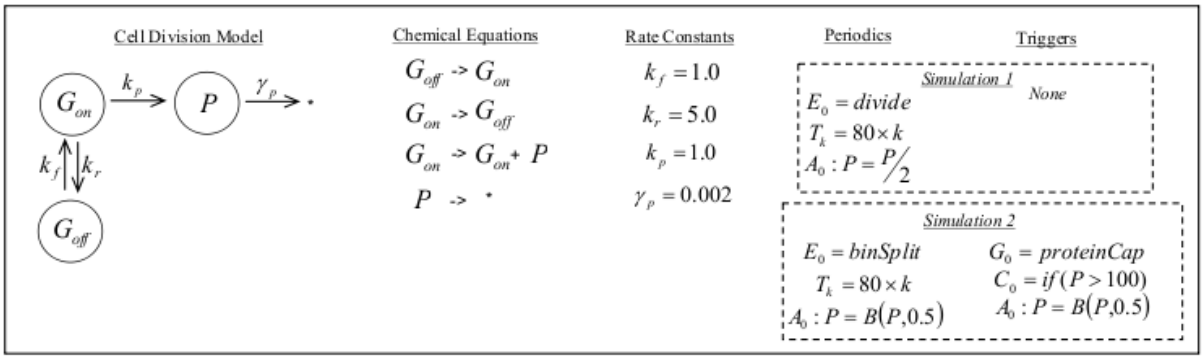

Figure 3.7 Cell Division Model

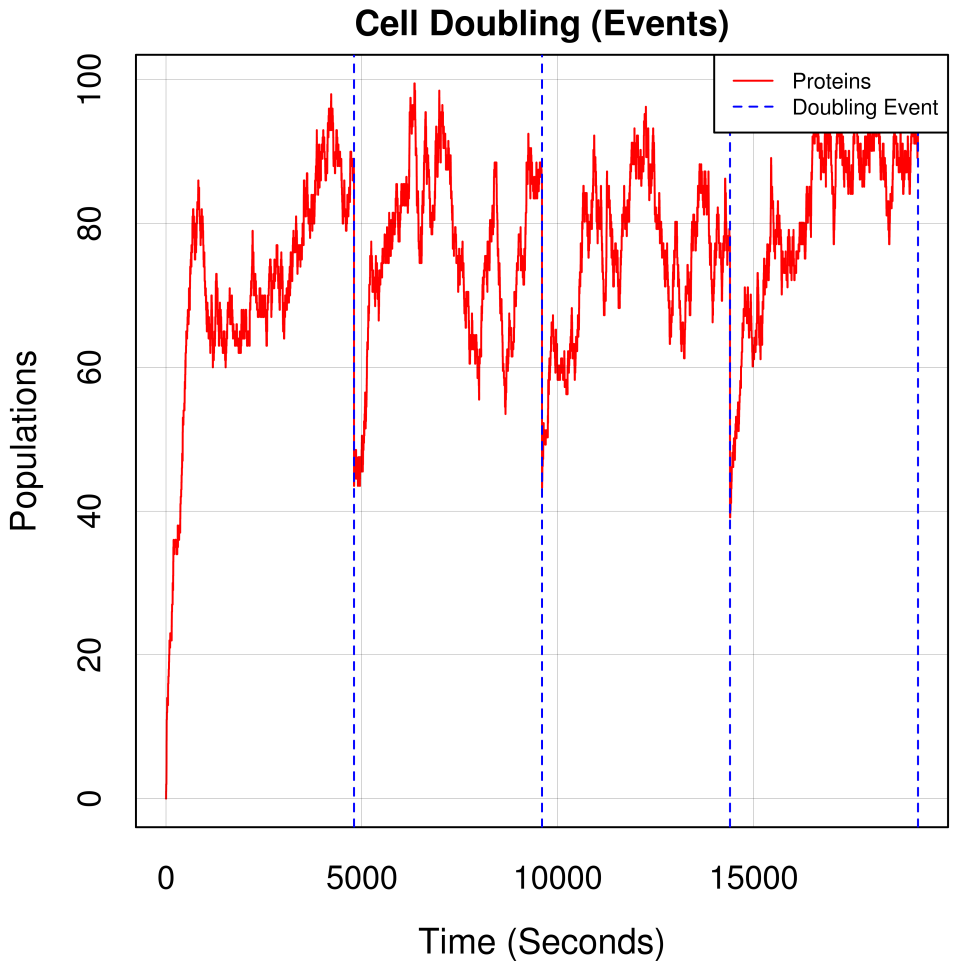

Figure 3.8 Cell Division Trajectory (w/Events)

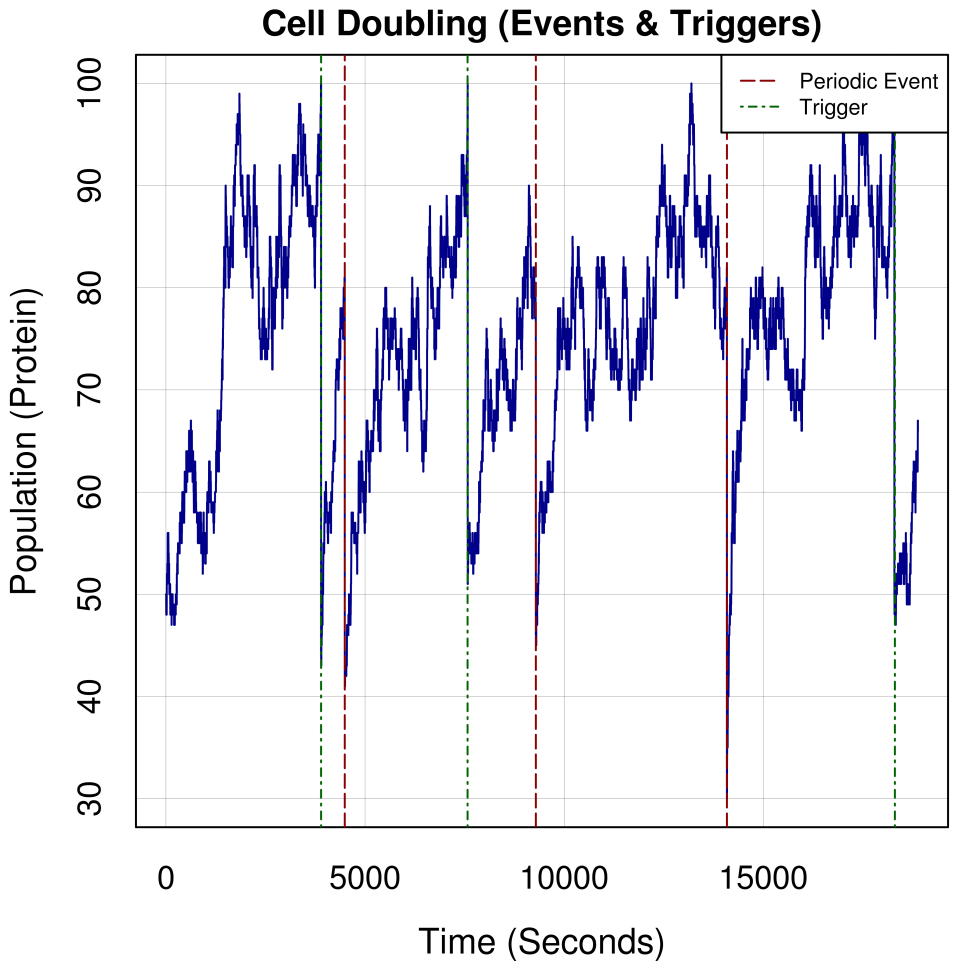

Figure 3.9 Cell Division Trajectory (w/Events & Triggers)

Chapter 4

Efficiency and the EHDM

Various approaches to the implementation of events and triggers can produce logically correct output. The naïve DM approach illustrated earlier would be a route employed by a researcher trying to generate ESS results quickly. It represents the simplest approach in getting data generated. While this method is effective, it is still inefficient in terms of reaction execution.

A more efficient approach would be to make the correct modifications to the SDM to implement events and triggers. This approach, depicted in Chapter 3, shows how a researcher already versed in ESS would approach the problem. This chapter introduces our contribution to the simulation methods.

While the algorithm outlined in Figure 3.1 produces a logically correct output, there are several modifications that can be made in order to increase the efficiency of the algorithm. This chapter presents the Efficient Event Handling Direct method (EEHDM) which serves to extend the concept of the dependency graph to our new additions (events/triggers) in order to develop a more efficient and complete SSA.

As previously mentioned, the SDM takes advantage of a dependency graph by only recalculating the necessary propensities between iterations. These "necessary propensities" are determined by examining the dependence between an executing reaction and other reactions

that reference species that are affected by the executing reaction.

In a similar fashion, the EEHDM employs a dependency graph to recalculate propensities based on the actions of events/triggers and on the execution of reactions. If a trigger or event is executed, only a small subset of propensities will need to be updated (those referenced in its action). Those that are not referenced will not need to be recalculated (as they did not change between iterations). The list of pairs that must be examined for dependence is as follows:

1. Trigger actions and reaction propensities
2. Event actions and reaction propensities
3. Reaction executions and reaction propensities

Step 2 of Figure 3.1 is modified here to only include the necessary propensity recalculations. Since this is typically a sparse graph, the speed-up is significant.

The second modification we make is in the checking of events. The naïve implementation would be to check all events in the list and to compare their execution time against the current time. Two possible optimizations can be made here. First, by keeping track of which events have already executed comparisons can be eliminated by removing the events in question from the list. Second, by maintaining a sorted list (based on execution time) of the events only one comparison is necessary to check whether or not the next event should be executed.

Consider a model with two events that have the same execution time. If on the first iteration through the EEHDM the event executes, the previous data that determines which reaction should execute is discarded in favor of the event's action. The system time is then set to the event's execution time as opposed incrementing it by the next potential reaction time.

On the second iteration a similar set of events occurs, only this time it is for the second event. The second event's action executes and the time is updated to the second event's execution time. Two iterations have effectively occurred at the same (simulation) time.

Our last optimization deals with the dependencies and triggers. To prevent the repeated checking of every trigger at each iteration, the following pairs must be examined for dependence.

1. Trigger actions and trigger conditions
2. Event actions and trigger conditions
3. Reaction executions and trigger conditions

By establishing these dependencies, we reduce the amount of triggers that must be rechecked to those that have dependent species populations in their conditions. This reduction must be done with care to prevent an endless loop of checking trigger conditions and executing actions.

The enhanced algorithm is shown in Figure 4.1. The modifications here first show all of the triggers being checked at the beginning of execution. After this initial check, trigger conditions are checked and executed based on the calculated dependencies between reactions, events, and triggers. The call to these trigger checks is therefore embedded in the actual execution of the reaction, event, or other trigger. To accomplish this we populate a list of affected triggers based on the object in question. Here, we define $RT_{j,k}$ as the number of triggers k dependent on the jth reaction. Similarly, we define $ET_{j,k}$ as the number of the number of triggers k dependent on the jth event. And finally we define $TT_{j,k}$ as the number of triggers dependent on the current (j) trigger.

1. $\forall\{G_l\}$: if(C_l == true) then
   (a) Execute a_l
2. (SSA Step A) $\forall\{R_j\}$: Let $\alpha_j = f(X_i(t), k_j, r_j)$
3. (SSA Step B) Let $\tau = \dfrac{-ln(\lambda_1)}{\sum_{j=1}^{n} \alpha_j}$
4. E_{Next} : if($T_{\text{Next}} <= t + \tau$ && hasNotExecuted(A_{Next})) then
   (a) Execute A_{Next}
   (b) Let $t = T_{\text{Next}}$
   (c) $\forall\{k\}$ in $ET_{j,k}$: if(C_k == true) then execute a_k
      i. Recurse through trigger dependencies until no conditions are satisfied
   (d) Goto Step 3.
5. (SSA Step C) Let μ satisfy $\displaystyle\sum_{j=1}^{\mu} \alpha_j \ <= \ \lambda_2 \sum_{j=1}^{n} \alpha_j \ < \ \sum_{j=\mu+1}^{n} \alpha_j$
6. (SSA Step D) $\forall\{S_i\}$: Let $X_i(t + \tau) = X_i(t) - r_{\mu,i} + p_{\mu,i}$
7. $\forall\{k\}$ in $RT_{\mu,k}$: if(C_k == true) then execute a_k
   (a) $\forall\{k_2\}$ in TT_{k,k_2} : if(C_{k_2} == true) then execute a_{k_2}
      i. Recurse through trigger dependencies until no conditions are satisfied
8. (SSA Step E) Let $t = t + \tau$
9. (SSA Step F) if $(t < T)$ then Goto 3

Figure 4.1 EEHDM Execution Overview

Chapter 5

Examples and Performance Measurement

Analytically speaking, if α represents the number of propensities calculated during the course of a simulation and T represents the number of trigger conditions checked, their values after execution of the various algorithms discussed can be predicted. Assuming the naïve DM approach produces counts of α and T for the reaction propensities and trigger conditions, a similar approach with the SDM should show a dramatic reduction in the α value while the T value shows no significant change. Similarly, utilizing the EEHDM should show dramatic reductions in both values when compares to the naïve DM approach.

In order to generate performance statistics for the EEHDM, an implementation in Java (v1.6) was written for analysis. This implementation would compare our algorithm with a naïve implementation of events and triggers. Comparison data would also be generated for the Direct Method. These three measurements (i.e. the naïve implementation of events/triggers with the DM, naïve events/triggers with the SDM, and the EEHDM) should give us a clear picture of the performance increase with our algorithm.

The program suite takes as input a file fully describing the model. This file is easily handwritten, or auto-generated utilizing a test program. The program continues by generating a separate Java file that can be compiled and executed on its own to actually run the simulation. In this way, the code that is run as the simulator is procedurally generated by our software to run in the most efficient way possible.

The current form of our ESS code generator is as stated above written in Java. The code that is generated can be either Java or C. While both algorithms show correctness in terms of stochastic simulation, the C code was necessary to generate proper performance statistics as the Java compiler does no optimize code as efficiently as the C compiler does.

5.1 Real Life Analysis

The SIR model along with the Cell Doubling models discussed in Section 3.2 were utilized in generating preliminary performance statistics. By examining Figures 5.1 through 5.4 several conclusions can be drawn. Most notably is that we do not appear to achieve a performance gain with the EEHDM. This is due to the fact that these models are relatively simple and contain very few events and triggers. In the case of Figure 5.2 triggers were not utilized at all (thus their omission from the Figure).

Timing analysis from the SIR and Cell Doubling models also suggests that the overhead in implementing events and triggers leads to a slight performance decrease in execution time for very simple models.

In order to properly evaluate the algorithm for efficiency, models were specifically generated with numerous species, reactions, events, and triggers. This is discussed further in Section 5.4.

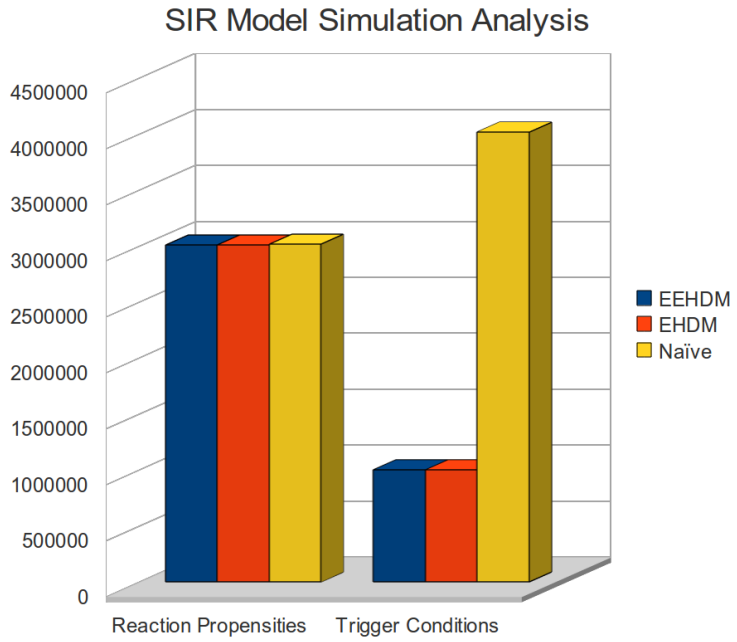

Figure 5.1 SIR Model Simulation Analysis

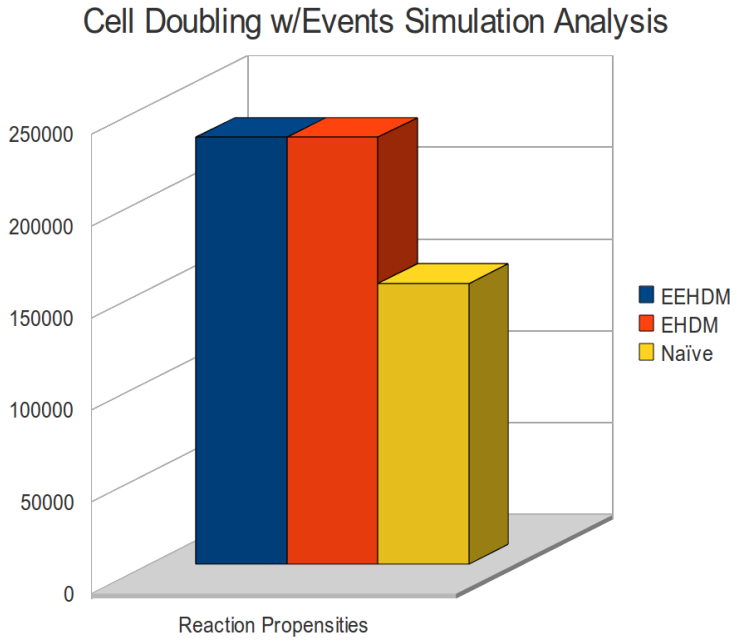

Figure 5.2 Cell Doubling w/Events Model Simulation Analysis

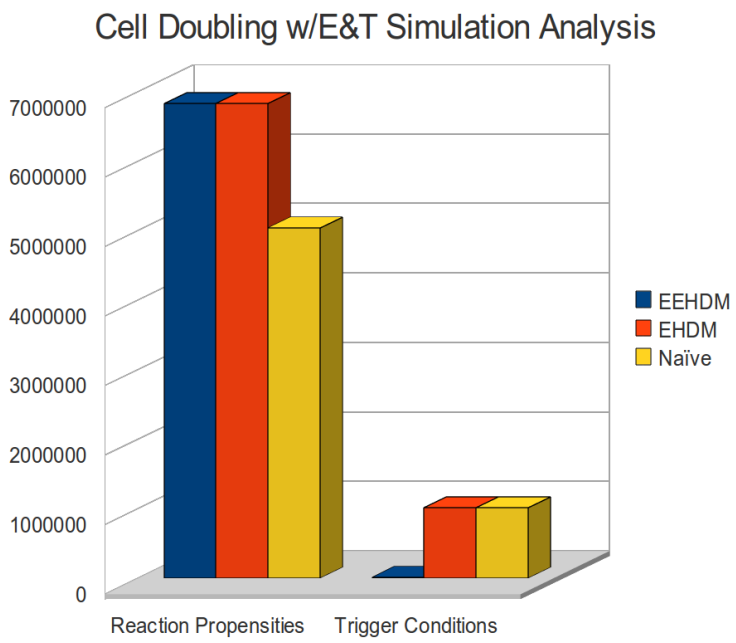

Figure 5.3 Cell Doubling w/Events & Triggers Model Simulation Analysis

5.2 Naïve vs. Efficient Approaches

Once a collection of events/triggers has been generated, implementations that are simply approached can be utilized to gather benchmarking data for the EEHDM. Given the set of triggers (G_l) and their associated actions (a_l), a naïve implementation would simply check all trigger conditions at the start of each iteration until no conditions are satisfied. Once this occurs, execution proceeds as normal.

This inefficient implementation forces a recheck of all triggers at the beginning of each iteration. Recall that the EEHDM only checks and executes triggers when a dependent species has been altered. The EEHDM only checks the necessary triggers based on the dependency graph.

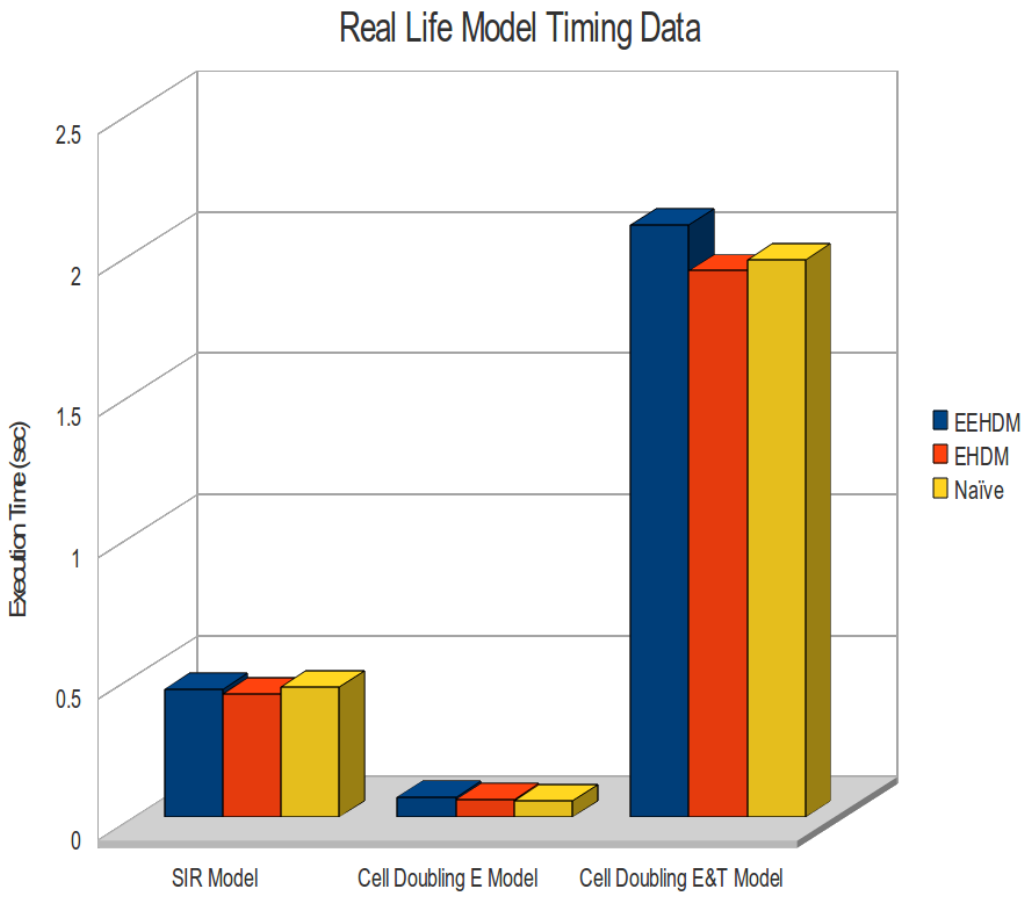

Figure 5.4 Real Life Models Timing Analysis

5.3 Events

The implementation of events, even in the naïve case, can not be accelerated in the same way as triggers were. In fact, their implementation is simply dependent on the current time during simulation. Maintaining the collection of events in a sorted structure requires only one comparison at each iteration.

5.4 Test Model

The model shown in Table 5.1 was used in testing the effectiveness of our algorithm. This model contains a set of species (S_a) along with a set of reactions. The reaction set (R_a) is shown in rows 1 and 2 of Table 5.1. These equations show a simple gene creating a protein at a randomly generated rate between 10 and 100. These proteins decay at a rate randomly generated between 0.5 and 1.5 (row 2).

Table 5.1 EEHDM Test System

#	***Reactions*** (R_a + R_b)	***Rate Constants***
1	$G_a \Rightarrow G_a + S_a$	$k_{a1} = \{10...100\}$
2	$S_a \Rightarrow *$	$k_{a2} = \{0.5...1.5\}$
#	***Trigger Condition*** (C_a)	***Trigger Action*** (a_a)
1	$S_i > \{20...40\}$	$S_j = \frac{S_i}{2}; S_i = 0$
#	***Periodic Event Execution Time*** (T_a)	***Trigger Action*** (A_a)
1	T_k = Every 1000 sim ticks	$S_a = \frac{S_a}{2}$

The statistics generated with this model were based on 50 species, 100 reactions (2× the number of species), 50 periodic events (1 for each species), and 100 triggers.

The events were constructed to represent the splitting of a species population at specific intervals. In this case, this occurs every 1000 ticks of simulation time for each species. The period of the periodic events was adjusted to generate models containing more events. At a period of 1000, 500 actual events will be created and executed in the program (i.e. 50 events * (10000 sim time / 1000 event period). At a period of 100, 5000 actual events will be created (i.e. 50 events * (10000 sim time / 100 event period).

Triggers were constructed in a way to introduce stress reaction dependence. Since the steady state populations of the species in the model fall specifically inside the range tested in the trigger conditions, we are ensured that they will fire often. The triggers check a random species population (S_i) and add half of its population to another random species (S_j) if the condition is satisfied. If this occurs, the population of the originating species is zeroed.

By making these parameters variable in our test generator, we can create specific models to evaluate our algorithm quickly and easily. Model 1 was generated with the base case of 50 species, 100 reactions, 50 periodic events (executing a total of 500 events), and 50 triggers. Models 2 and 3 were adjusted to have a greater model count (the case of generating 5000 actual events discussed earlier).

Tables 5.2 and 5.3 show the breakdown of function calls spent in executing various parts of the model. This data was generated by keeping track of how often events and triggers execute compared to how often reactions execute. In doing this, we will be able to tell how effective our algorithm is when the overall execution time of all of the events and triggers in the system increases.

Table 5.2 Algorithm Comparison Data (Raw Count)

Model	***Reaction Count***	***Event Count***	***Trigger Count***
1	4.932E+7	500	7.715E+5
2	4.607E+7	5000	5.995E+6
3	4.646E+7	5000	1.436E+7

Models 2 and 3 were also tailored such that they execute triggers more frequently than Model 1. This was accomplished by adjusting the condition at which the trigger was executed. The range specified in Table 5.1 for the trigger condition was lowered to accomplish

Table 5.3 Algorithm Comparison Data

Model	*Reaction Count (%)*	*Event Count (%)*	*Trigger Count (%)*
1	98.45	9.98E-4	1.539
2	88.47	9.60E-3	11.513
3	76.38	8.21E-3	23.608

this. This will utilize more of the enhancements specified by the EEHDM and should increase performance.

To illustrate the performance gains achieved by the EEHDM two sets of figures will be presented. The first show the total number of reactions propensities calculated, the total number of trigger conditions checked, and the total number of event conditions checked for each simulation. This is shown in Figures 5.5 through 5.7.

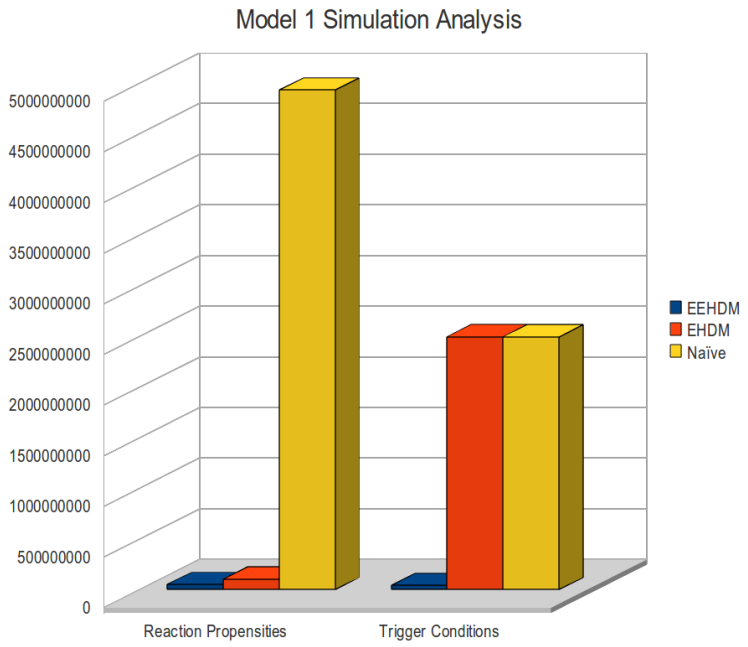

Figure 5.5 Model 1 Simulation Analysis

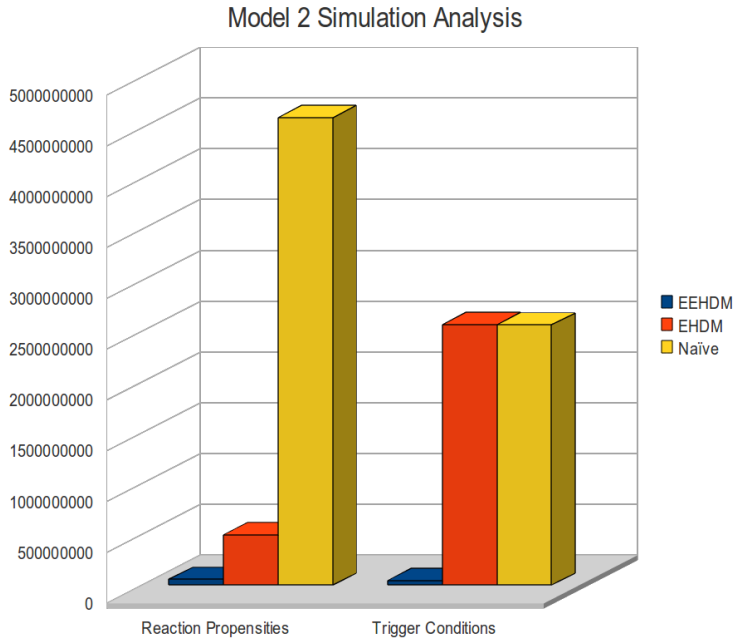

Figure 5.6 Model 2 Simulation Analysis

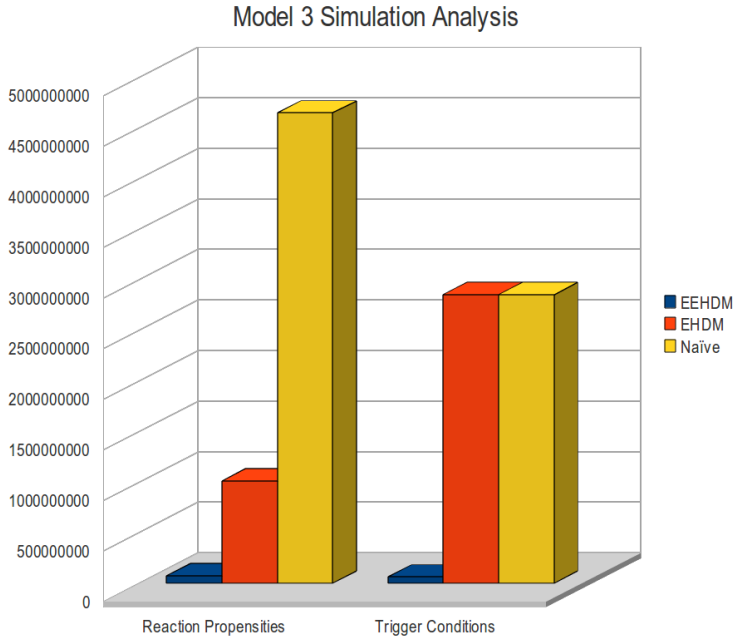

Figure 5.7 Model 3 Simulation Analysis

Table 5.4 Algorithm Performance Comparison Data (Raw Counts)

Parameter	***EEHDM***	***EHDM***	***Naïve***
Model 1			
Reaction Propensities	5.087E+7	1.031E+8	4.930E+9
Trigger Conditions	3.978E+7	2.497E+9	2.496E+9
Event Conditions	4.932E+7	4.932E+7	4.930E+7
Model 2			
Reaction Propensities	5.807E+7	4.982E+8	4.601E+9
Trigger Conditions	4.651E+7	2.564E+9	2.564E+9
Event Conditions	4.608E+7	4.602E+7	4.601E+7
Model 3			
Reaction Propensities	7.519E+7	1.010E+9	4.648E+9
Trigger Conditions	6.246E+7	2.854E+9	2.854E+9
Event Conditions	4.647E+7	4.649E+7	4.648E+7

Figures 5.5 through 5.7 show that when employing the EEHDM over other simulation methods, the improvement on trigger conditions is great. As we progress through the models and examine the impact of executing triggers more often, we notice that we also see a performance improvement in the number of reaction propensities calculated. This is expected as the EEHDM only recalculated reaction propensities based on the dependency graph (with respect to events and triggers). The EHDM recalculates all propensities when a trigger or event fires (as it has no knowledge of the dependencies). Table 5.4 shows the data used to generate the plots in Figures 5.5 through 5.7.

Figure 5.8 shows the execution time of the given models for the various simulation algo-

rithms. The first of which is our EEHDM. The second is the SDM with a simplistic approach toward the implementation of events and triggers. Finally, the third is the DM with the same implementation of events and triggers as the second approach.

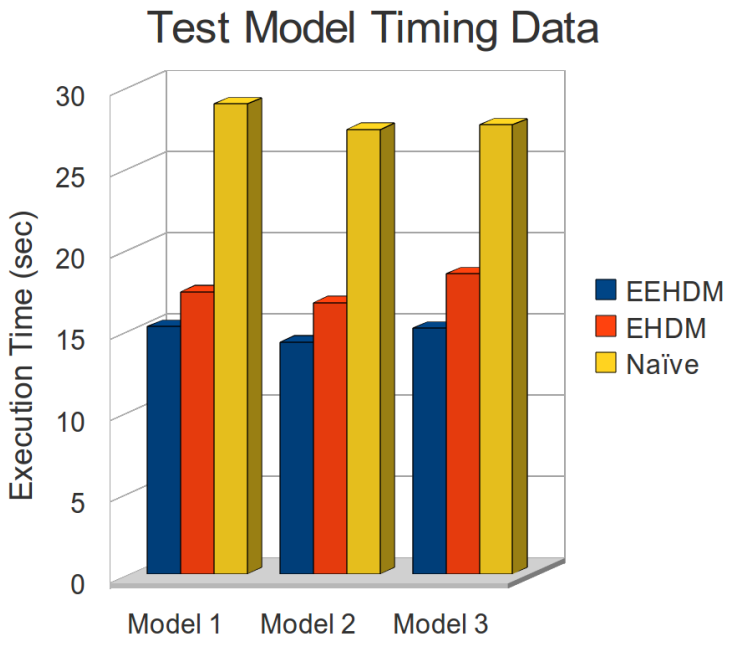

Figure 5.8 Test Model Timing Data

By examining Figure 5.8 we notice that in all cases the EEHDM out performs the other methods. By combining this with the data in Table 5.5 further conclusions can be drawn. Notice that as we proceed through the models (and thus increasing the amount of time we spend executing events and triggers) the factor by which the EEHDM increases performance improves from 1.14x to 1.22x. This data supports the point that as the execution time of events and triggers increases, the EEHDM performs the most efficiently of the 3 implementations. For models that don't execute events and triggers often, the performance falls back to the performance of the SDM (as the EEHDM is an extension to the SDM).

Table 5.5 Algorithm Execution Comparison Data

Model	***EEHDM (sec)***	***EHDM (sec)***	***Naïve (sec)***	***Speedup (x)***
1	15.209	17.345	28.907	1.14
2	14.244	16.677	27.342	1.17
3	15.111	18.453	27.617	1.22

Conclusion

Allowing events and triggers to be added to the SSA simplifies the ways in which models can be described and expands the diversity of models that can be simulated using the SSA. Implementation of events and triggers was demonstrated for the fastest implementation of the SSA, the Sorting Direct Method. The dependencies between events, triggers, and reactions was then exploited for further performance gain. This was verified both analytically and through performance measurements of an implementation in Java. It is the hope of the author that this will significantly expand the applicability of the SSA to a wider variety of ecological, chemical, and biological modeling problems.

Bibliography

[1] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic, *Digital integrated circuits : a design perspective*, 2nd ed., ser. Prentice Hall electronics and VLSI series. Pearson Education, January 2003.

[2] G. Totten, *Modeling and Simulation for Material Selection and Mechanical Design*. New York: Marcel Dekker, 2004.

[3] H. de Jong, "Modeling and simulation of genetic regulatory systems: A literature review," *Journal of Computational Biology*, vol. 9, no. 1, pp. 67–103, January 2002. [Online]. Available: http://dx.doi.org/10.1089/10665270252833208

[4] J. Hasty, D. McMillen, and J. J. Collins, "Engineered gene circuits." *Nature*, vol. 420, no. 6912, pp. 224–230, November 2002. [Online]. Available: http://dx.doi.org/10.1038/nature01257

[5] D. T. Gillespie, "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions," *Journal of Computational Physics*, vol. 22, no. 4, pp. 403–434, December 1976. [Online]. Available: http://dx.doi.org/10.1016/0021-9991(76)90041-3

[6] ——, "Exact stochastic simulation of coupled chemical reactions," *The Journal of Physical Chemistry*, vol. 81, no. 25, pp. 2340–2361, 1977. [Online]. Available: http://dx.doi.org/10.1021/j100540a008

[7] D. W. Austin, M. S. Allen, J. M. McCollum, R. D. Dar, J. R. Wilgus, G. S. Sayler, N. F. Samatova, C. D. Cox, and M. L. Simpson, "Gene network shaping of inherent noise spectra." *Nature*, vol. 439, no. 7076, pp. 608–611, February 2006. [Online]. Available: http://dx.doi.org/10.1038/nature04194

[8] M. A. Gibson and J. Bruck, "Efficient exact stochastic simulation of chemical systems with many species and many channels," *The Journal of Physical Chemistry A*, vol. 104, no. 9, pp. 1876–1889, March 2000. [Online]. Available: http://dx.doi.org/10.1021/jp993732q

[9] Y. Cao, H. Li, and L. Petzold, "Efficient formulation of the stochastic simulation algorithm for chemically reacting systems." *J Chem Phys*, vol. 121, no. 9, pp. 4059–4067, September 2004. [Online]. Available: http://dx.doi.org/10.1063/1.1778376

[10] H. Li and L. Petzold, "Logarithmic direct method for discrete stochastic simulation of chemically reacting systems," Jul. 2006. [Online]. Available: http://www.engineering.ucsb.edu/~cse/Files/ldm0513.pdf

[11] J. M. McCollum, G. D. Peterson, C. D. Cox, M. L. Simpson, and N. F. Samatova, "The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior," *Computational Biology and Chemistry*, vol. 30, no. 1, pp. 39–49, February 2006. [Online]. Available: http://dx.doi.org/10.1016/j.compbiolchem.2005.10.007

[12] Y. Cao, H. Li, and L. Petzold, "Efficient formulation of the stochastic simulation algorithm for chemically reacting systems." *J Chem Phys*, vol. 121, no. 9, pp. 4059–4067, September 2004. [Online]. Available: http://dx.doi.org/10.1063/1.1778376

[13] N. Metropolis and S. Ulam, "The monte carlo method," *Journal of the American Statistical Association*, vol. 44, no. 247, pp. 335–341, 1949. [Online]. Available: http://dx.doi.org/10.2307/2280232

[14] P. S. Swain, M. B. Elowitz, and E. D. Siggia, "Intrinsic and extrinsic contributions to stochasticity in gene expression," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. 20, pp. 12 795–12 800, October 2002. [Online]. Available: http://dx.doi.org/10.1073/pnas.162041399

[15] M. J. Keeling, P. Rohani, and B. T. Grenfell, "Seasonally forced disease dynamics explored as switching between attractors," *Physica D: Nonlinear Phenomena*, vol. 148, no. 3-4, pp. 317 – 335, 2001. [Online]. Available: http://www.sciencedirect.com/science/article/B6TVK-41XMH7H-9/2/a78fbfbda456b5fb9f2e178ce75367bb

[16] P. J. Goss and J. Peccoud, "Quantitative modeling of stochastic systems in molecular biology by using stochastic petri nets." *Proceedings of the National Academy of Sciences of the United States of America*, vol. 95, no. 12, pp. 6750–6755, June 1998. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/9618484

Appendix A

EEHDM Code

This appendix contains the Java source code for the implementation of the EEHDM. The code is arranged inside the package edu.vcu.scl.ess.

A.1 RawModel

The following file reads the model file in to the simulator and populates initial data structures. It then passes the objects to RawmodelCheck where the model is tested for correctness.

```
1  package edu.vcu.csl.ess;
2
3  import java.util.ArrayList;
4  import java.util.Comparator;
5  import java.io.*;
6
7  public class RawModel {
8    public static class Constant {
9      public String name, value, comment, error;
10   }
11
```

```
12   public static class Species  {
13     public String name, population, comment, error;
14   }
15
16   public static class Reaction {
17     public String name, equation, rate, comment, error;
18     public ArrayList<ReactionElement> reactants     = new ArrayList<
         ReactionElement>();
19     public ArrayList<ReactionElement> products      = new ArrayList<
         ReactionElement>();
20     public ArrayList<ReactionElement> delta         = new ArrayList<
         ReactionElement>();
21     public ArrayList<Reaction> affectedReactionList = new ArrayList<Reaction
         >();
22     public ArrayList<Trigger>  affectedTriggerList  = new ArrayList<Trigger>()
         ;
23   }
24
25   public static class Event    {
26     public String name, time, action, comment, error;
27     public ArrayList<Reaction> affectedReactionList = new ArrayList<Reaction
         >();
28     public ArrayList<Trigger>  affectedTriggerList  = new ArrayList<Trigger>()
         ;
29   }
30
31   public static class Periodic {
32     public String name, start, period, end, action, comment, error;
33     public ArrayList<Reaction> affectedReactionList = new ArrayList<Reaction
         >();
```

```
34      public ArrayList<Trigger> affectedTriggerList = new ArrayList<Trigger>()
            ;
35    }
36
37    public static class Trigger {
38      public String name, condition, action, comment, error;
39      public ArrayList<Reaction> affectedReactionList = new ArrayList<Reaction
            >();
40      public ArrayList<Trigger> affectedTriggerList = new ArrayList<Trigger>()
            ;
41    }
42
43    public static class ReactionElement {
44      public int coefficient;
45      public String species;
46    };
47
48    public static class EventSort implements Comparator<Event>{
49      public int compare (Event o1, Event o2){
50        if (Double.parseDouble(o1.time) <
51            Double.parseDouble(o2.time)){
52          return -1;
53        }
54        else if (Double.parseDouble(o1.time) >
55        Double.parseDouble(o2.time)){
56          return 1;
57        }
58        else{
59          return 0;
60        }
```

```
61      }
62    }
63
64    public ArrayList<Constant>   constants  = new ArrayList<Constant>();
65    public ArrayList<Species>    species    = new ArrayList<Species>();
66    public ArrayList<Reaction>   reactions  = new ArrayList<Reaction>();
67    public ArrayList<Event>      events     = new ArrayList<Event>();
68    public ArrayList<Periodic>   periodics  = new ArrayList<Periodic>();
69    public ArrayList<Trigger>    triggers   = new ArrayList<Trigger>();
70    public double                start      = 0.0;
71    public double                end        = 10.0;
72    public double                interval   = 1.0;
73    public int                   seed       = 1;
74    public String                filename   = "out.txt";
75    public ArrayList<String>     errors     = new ArrayList<String>();
76
77    public RawModel() {
78
79    }
80
81    public void write(File file) throws IOException  {
82      PrintStream out = new PrintStream(new FileOutputStream(file));
83
84      out.println("-- RawModel FILE v1.1 --");
85
86      out.println(constants.size());
87      for(int i=0; i<constants.size(); i++) {
88        Constant x = constants.get(i);
89        out.println(x.name.replace('\n','\t'));
90        out.println(x.value.replace('\n','\t'));
```

```
91          out.println(x.comment.replace('\n','\t'));
92        }
93
94        out.println(species.size());
95        for(int i=0; i<species.size(); i++) {
96          Species x = species.get(i);
97          out.println(x.name.replace('\n','\t'));
98          out.println(x.population.replace('\n','\t'));
99          out.println(x.comment.replace('\n','\t'));
100       }
101
102       out.println(reactions.size());
103       for(int i=0; i<reactions.size(); i++) {
104         Reaction x = reactions.get(i);
105         out.println(x.name.replace('\n','\t'));
106         out.println(x.equation.replace('\n','\t'));
107         out.println(x.rate.replace('\n','\t'));
108         out.println(x.comment.replace('\n','\t'));
109       }
110
111       out.println(events.size());
112       for(int i=0; i<events.size(); i++) {
113         Event x = events.get(i);
114         out.println(x.name.replace('\n','\t'));
115         out.println(x.time.replace('\n','\t'));
116         out.println(x.action.replace('\n','\t'));
117         out.println(x.comment.replace('\n','\t'));
118       }
119
120       out.println(periodics.size());
```

```
121     for(int i=0; i<periodics.size(); i++) {
122       Periodic x = periodics.get(i);
123       out.println(x.name.replace('\n','\t'));
124       out.println(x.start.replace('\n','\t'));
125       out.println(x.period.replace('\n','\t'));
126       out.println(x.end.replace('\n','\t'));
127       out.println(x.action.replace('\n','\t'));
128       out.println(x.comment.replace('\n','\t'));
129     }
130
131     out.println(triggers.size());
132     for(int i=0; i<triggers.size(); i++) {
133       Trigger x = triggers.get(i);
134       out.println(x.name.replace('\n','\t'));
135       out.println(x.condition.replace('\n','\t'));
136       out.println(x.action.replace('\n','\t'));
137       out.println(x.comment.replace('\n','\t'));
138     }
139
140     out.println(start);
141     out.println(end);
142     out.println(interval);
143     out.println(seed);
144     out.println(filename);
145
146     out.close();
147   }
148
149   public static RawModel read(File file) throws IOException {
150     BufferedReader in = new BufferedReader(new FileReader(file));
```

```
151     RawModel model = new RawModel();
152     int count;
153
154     if (!in.readLine().equals("-- RawModel FILE v1.1 --"))
155       throw new IOException();
156
157     count = Integer.parseInt(in.readLine());
158     for(int i=0; i<count; i++) {
159       model.addConstant(in.readLine().replace('\t','\n'),
160           in.readLine().replace('\t','\n'),
161           in.readLine().replace('\t','\n'));
162     }
163
164     count = Integer.parseInt(in.readLine());
165     for(int i=0; i<count; i++) {
166       model.addSpecies(in.readLine().replace('\t','\n'),
167           in.readLine().replace('\t','\n'),
168           in.readLine().replace('\t','\n'));
169     }
170
171     count = Integer.parseInt(in.readLine());
172     for(int i=0; i<count; i++) {
173       model.addReaction(in.readLine().replace('\t','\n'),
174           in.readLine().replace('\t','\n'),
175           in.readLine().replace('\t','\n'),
176           in.readLine().replace('\t','\n'));
177     }
178
179     count = Integer.parseInt(in.readLine());
180     for(int i=0; i<count; i++) {
```

```
181         model.addEvent(in.readLine().replace('\t','\n'),
182                 in.readLine().replace('\t','\n'),
183                 in.readLine().replace('\t','\n'),
184                 in.readLine().replace('\t','\n'));
185       }
186
187       count = Integer.parseInt(in.readLine());
188       for(int i=0; i<count; i++) {
189         model.addPeriodic(in.readLine().replace('\t','\n'),
190                 in.readLine().replace('\t','\n'),
191                 in.readLine().replace('\t','\n'),
192                 in.readLine().replace('\t','\n'),
193                 in.readLine().replace('\t','\n'),
194                 in.readLine().replace('\t','\n'));
195       }
196
197       count = Integer.parseInt(in.readLine());
198       for(int i=0; i<count; i++) {
199         model.addTrigger(in.readLine().replace('\t','\n'),
200                 in.readLine().replace('\t','\n'),
201                 in.readLine().replace('\t','\n'),
202                 in.readLine().replace('\t','\n'));
203       }
204
205       model.setStartTime(Double.parseDouble(in.readLine()));
206       model.setEndTime(Double.parseDouble(in.readLine()));
207       model.setPrintInterval(Double.parseDouble(in.readLine()));
208       model.setSeed(Integer.parseInt(in.readLine()));
209       model.setOutputFilename(in.readLine());
210
```

```
211     return model;
212   }
213
214
215   public void addConstant(String name,
216       String value,
217       String comment) {
218     Constant p = new Constant();
219     p.name = name;
220     p.value = value;
221     p.comment = comment;
222     p.error = null;
223     constants.add(p);
224   }
225
226   public void addSpecies(String name,
227       String population,
228       String comment) {
229     Species s = new Species();
230     s.name = name;
231     s.population = population;
232     s.comment = comment;
233     s.error = null;
234     species.add(s);
235   }
236
237   public void addReaction(String name,
238       String equation,
239       String rate,
240       String comment) {
```

```
241     Reaction r = new Reaction();
242     r.name = name;
243     r.equation = equation;
244     r.rate = rate;
245     r.comment = comment;
246     r.error = null;
247     reactions.add(r);
248   }
249
250   public void addEvent(String name,
251       String time,
252       String action,
253       String comment) {
254     Event e = new Event();
255     e.name = name;
256     e.time = time;
257     e.action = action;
258     e.comment = comment;
259     e.error = null;
260     events.add(e);
261   }
262
263   public void addPeriodic(String name,
264       String start,
265       String period,
266       String end,
267       String action,
268       String comment) {
269     Periodic p = new Periodic();
270     p.name = name;
```

```
271     p.start = start;
272     p.period = period;
273     p.end = end;
274     p.action = action;
275     p.comment = comment;
276     p.error = null;
277     periodics.add(p);
278   }
279
280   public void addTrigger(String name,
281       String condition,
282       String action,
283       String comment) {
284     Trigger t = new Trigger();
285     t.name = name;
286     t.condition = condition;
287     t.action = action;
288     t.comment = comment;
289     t.error = null;
290     triggers.add(t);
291   }
292
293   public void setStartTime(double start) {
294     this.start = start;
295   }
296
297   public void setEndTime(double end) {
298     this.end = end;
299   }
300
```

```
301   public void setPrintInterval(double interval) {
302     this.interval = interval;
303   }
304
305   public void setSeed(int seed) {
306     this.seed = seed;
307   }
308
309   public void setOutputFilename(String filename) {
310     this.filename = filename;
311   }
312
313   public boolean run() {
314     return RawModelCheck.isValid(this);
315   }
316
317   public void printErrors() {
318     for(Constant x : constants) {
319       if (x.error != null) {
320         System.out.println("Error: ");
321         System.out.println("Constant " + x.name);
322         System.out.println(x.error);
323         System.out.println();
324       }
325     }
326
327     for(Species x : species) {
328       if (x.error != null) {
329         System.out.println("Error: ");
330         System.out.println("Species " + x.name);
```

```
331           System.out.println(x.error);
332           System.out.println();
333         }
334       }
335
336       for(Reaction x : reactions) {
337         if (x.error != null) {
338           System.out.println("Error: ");
339           System.out.println("Reaction " + x.name);
340           System.out.println(x.error);
341           System.out.println();
342         }
343       }
344
345       for(Event x : events) {
346         if (x.error != null) {
347           System.out.println("Error: ");
348           System.out.println("Event " + x.name);
349           System.out.println(x.error);
350           System.out.println();
351         }
352       }
353
354       for(Periodic x : periodics) {
355         if (x.error != null) {
356           System.out.println("Error: ");
357           System.out.println("Periodic Event " + x.name);
358           System.out.println(x.error);
359           System.out.println();
360         }
```

```
361     }
362
363     for(Trigger x : triggers) {
364       if (x.error != null) {
365         System.out.println("Error: ");
366         System.out.println("Trigger " + x.name);
367         System.out.println(x.error);
368         System.out.println();
369       }
370     }
371
372     for(String x : errors) {
373       System.out.println("Error: ");
374       System.out.println(x);
375       System.out.println();
376     }
377   }
378
379   public static void main(String args[]) throws IOException {
380     if (args.length != 2) {
381       System.out.println("usage: java RawModel <filename> <model_type>");
382       System.exit(1);
383     }
384
385     RawModel model = RawModel.read(new File(args[0]));
386     if (model.run() == false) {
387       model.printErrors();
388     } else {
389       System.out.println("Construction complete...");
390       //        ModelCodeGen.generateOutputFile(model);
```

```
391         //        ModelCodeGen.generateDirectMethodOutputFile(model);
392
393         if(Integer.parseInt(args[1]) == 1){
394           System.out.println("Using EEHDM...");
395           ModelCodeGenC.generateOutputFile(model);
396         }
397         else if(Integer.parseInt(args[1]) == 2){
398           System.out.println("Using Naive EHDM...");
399           ModelCodeGenC.generateNaiveOutputFile(model);
400         }
401         else if(Integer.parseInt(args[1]) == 99){
402           System.out.println("Using Java EEHDM...");
403           ModelCodeGen.generateOutputFile(model);
404         }
405         else{
406           System.out.println("Using Naive DM");
407           ModelCodeGenC.generateDirectMethodOutputFile(model);
408         }
409       }
410     }
411 }
```

RawModel.java

A.2 RawModelCheck

The following file tests the model read in for validity. If the model is valid, it passes the objects to the parser where the remainder of the necessary data structures are populated.

```
1  package edu.vcu.csl.ess;
2
3  import edu.vcu.csl.ess.RawModel;
4  import edu.vcu.csl.ess.RawModel.Reaction;
5
6  import java.util.ArrayList;
7
8  class RawModelCheck {
9    public static boolean isValid(RawModel model) {
10     int errorCount = 0;
11
12     errorCount += checkForSpecies(model);
13     errorCount += checkTimeParameters(model);
14     errorCount += checkForValidAndUniqueNames(model);
15     errorCount += checkReactionEquations(model);
16     errorCount += checkEventTimes(model);
17
18     return (errorCount == 0);
19   }
20
21   public static int checkTimeParameters(RawModel model) {
22     int errorCount = 0;
23
24     if (model.start < 0.0) {
25       model.errors.add("Start time is negative.");
26       errorCount++;
27     }
28
29     if (model.end <= 0.0) {
30       model.errors.add("End time must be greater than zero.");
```

```
31        errorCount++;
32      }
33
34      if (model.end <= model.start) {
35        model.errors.add("End time must be greater than the start time.");
36        errorCount++;
37      }
38
39      if (model.interval <= 0.0) {
40        model.errors.add("Print interval must be greater than zero.");
41        errorCount++;
42      }
43
44      return errorCount;
45    }
46
47    public static int checkForSpecies(RawModel model) {
48      int errorCount = 0;
49
50      if (model.species.size() == 0) {
51        model.errors.add("No species exist in the model.");
52        errorCount++;
53      }
54
55      return errorCount;
56    }
57
58    public static int checkForValidAndUniqueNames(RawModel model) {
59      ArrayList<String> names = new ArrayList<String>();
60      int errorCount = 0;
```

```
61
62      for (RawModel.Constant s : model.constants) {
63        if (!JavaCodeParser.isValidName(s.name)) {
64          errorCount++;
65          s.error = "Invalid name.";
66        } else {
67          if (isStringInArrayList(s.name,names)) {
68            errorCount++;
69            s.error = "Duplicate name.";
70          } else {
71            names.add(s.name);
72          }
73        }
74      }
75
76      for (RawModel.Species s : model.species) {
77        if (!JavaCodeParser.isValidName(s.name)) {
78          errorCount++;
79          s.error = "Invalid name.";
80        } else {
81          if (isStringInArrayList(s.name,names)) {
82            errorCount++;
83            s.error = "Duplicate name.";
84          } else {
85            names.add(s.name);
86          }
87        }
88      }
89
90      for (RawModel.Reaction s : model.reactions) {
```

```
91        if (!JavaCodeParser.isValidName(s.name)) {
92          errorCount++;
93          s.error = "Invalid name.";
94        } else {
95          if (isStringInArrayList(s.name,names)) {
96            errorCount++;
97            s.error = "Duplicate name.";
98          } else {
99            names.add(s.name);
100         }
101       }
102     }
103
104     for (RawModel.Event s : model.events) {
105       if (!JavaCodeParser.isValidName(s.name)) {
106         errorCount++;
107         s.error = "Invalid name.";
108       } else {
109         if (isStringInArrayList(s.name,names)) {
110           errorCount++;
111           s.error = "Duplicate name.";
112         } else {
113           names.add(s.name);
114         }
115       }
116     }
117
118     for (RawModel.Periodic s : model.periodics) {
119       if (!JavaCodeParser.isValidName(s.name)) {
120         errorCount++;
```

```
121         s.error = "Invalid name.";
122       } else {
123         if (isStringInArrayList(s.name,names)) {
124           errorCount++;
125           s.error = "Duplicate name.";
126         } else {
127           names.add(s.name);
128         }
129       }
130     }
131
132     for (RawModel.Trigger s : model.triggers) {
133       if (!JavaCodeParser.isValidName(s.name)) {
134         errorCount++;
135         s.error = "Invalid name.";
136       } else {
137         if (isStringInArrayList(s.name,names)) {
138           errorCount++;
139           s.error = "Duplicate name.";
140         } else {
141           names.add(s.name);
142         }
143       }
144     }
145
146     return errorCount;
147   }
148
149   public static int checkReactionEquations(RawModel model) {
150     int errorCount = 0;
```

```
151
152     for (RawModel.Reaction r : model.reactions) {
153       ReactionParser.parse(r);
154       if (r.error == null) {
155         for(RawModel.ReactionElement e : r.reactants) {
156           boolean valid = false;
157           for(RawModel.Species s : model.species) {
158             if (s.name.compareTo(e.species) == 0) {
159               valid = true;
160               break;
161             }
162           }
163           if (!valid) {
164             r.error = "Unknown reactant '" + e.species + "'.";
165             errorCount++;
166           }
167         }
168         for(RawModel.ReactionElement e : r.products) {
169           boolean valid = false;
170           for(RawModel.Species s : model.species) {
171             if (s.name.compareTo(e.species) == 0) {
172               valid = true;
173               break;
174             }
175           }
176           if (!valid) {
177             r.error = "Unknown product '" + e.species + "'.";
178             errorCount++;
179           }
180         }
```

```
181       }
182     }
183
184     return errorCount;
185   }
186
187   public static int checkEventTimes(RawModel model) {
188     int errorCount = 0;
189
190     for (RawModel.Event e : model.events) {
191       if (Double.parseDouble(e.time) < 0) {
192         e.error = "Event time must be greater than 0.";
193         errorCount++;
194       }
195     }
196
197     for (RawModel.Periodic p : model.periodics){
198       if (Double.parseDouble(p.period) <= 0){
199         p.error = "Period must be greater than 0.";
200         errorCount++;
201       }
202       if (Double.parseDouble(p.start) < 0){
203         p.error = "Start time must not be negative.";
204         errorCount++;
205       }
206       if (Double.parseDouble(p.end) < 0) {
207         p.error = "End time must be greater than 0.";
208         errorCount++;
209       }
210       if (Double.parseDouble(p.end) < Double.parseDouble(p.start)) {
```

```
211             p.error = "Start time must be less than end time.";
212             errorCount++;
213           }
214         }
215
216         return errorCount;
217       }
218
219       public static boolean isStringInArrayList(String str, ArrayList<String> list
              ) {
220         for(String s : list) {
221           if (str.compareTo(s) == 0) {
222             return true;
223           }
224         }
225         return false;
226       }
227     }
```

RawModelCheck.java

A.3 ReactionParser

The following file parses the remainder of the data into their specified objects and passes the data to our final validity checker.

```
1  package edu.vcu.csl.ess;
2
3  import java.io.BufferedReader;
```

```
4  import java.io.BufferedWriter;
5  import java.io.ByteArrayInputStream;
6  import java.io.ByteArrayOutputStream;
7  import java.io.File;
8  import java.io.FileInputStream;
9  import java.io.FileWriter;
10 import java.io.IOException;
11 import java.io.InputStreamReader;
12 import java.io.ObjectInputStream;
13 import java.io.ObjectOutputStream;
14 import java.io.PrintWriter;
15 import java.util.ArrayList;
16 import java.util.Properties;
17 import java.util.regex.Matcher;
18 import java.util.regex.Pattern;
19
20 import edu.vcu.csl.ess.RawModel.*;
21
22 public class ReactionParser {
23
24   /**
25    * Parses the reaction equations out into the variables necessary for the
          simulator to run.
26    * @param r The current reaction to parse.
27    */
28   public static boolean parse(Reaction r){
29     String[] reaction = r.equation.split("=>");
30
31     if (reaction.length > 2){
32       r.error = "Too many => arrows";
```

```
33        return false;
34      }
35      else if(reaction.length == 1){
36        r.error = "No yields arrow (=>) found.";
37        return false;
38      }
39      //Split on + sign and parse individual elements
40
41      //Parse reactants (reaction[0])
42      String[] terms = reaction[0].trim().split("\\+");//Hack avoids dangling
            meta character error
43      boolean temp = checkAndAddElements(r, terms, false);
44      if(!temp){return temp;}
45
46      //Parse products (reaction[1])
47      terms = reaction[1].trim().split("\\+");
48      temp = checkAndAddElements(r, terms, true);
49      if(!temp){return temp;}
50
51      return true;
52
53    }
54
55    private static boolean checkAndAddElements(Reaction r, String[] terms,
          boolean trueIfProducts){
56
57      boolean goBack = false; //Hack var to break out of function
58
59      int subStringLength;
60
```

```
61      for (String t : terms){
62        t=t.trim();
63        subStringLength = 0;
64
65        //Increment through first term until we don't see a digit. (Extract
              coefficient)
66        for (char c : t.toCharArray()){
67          if (!Character.isDigit(c)){break;}
68          subStringLength++;
69        }
70
71        //Finish extracting coefficient
72        int coefficient = 0;
73        if (subStringLength == 0){coefficient = 1;} //Handles case when variable
              does not have a digit coefficient
74        else{
75          try {
76            coefficient = Integer.parseInt(t.substring(0,subStringLength));
77          } catch (NumberFormatException e) {
78            r.error = "Invalid coefficient.";
79            return false;
80          }
81        }
82
83        //Throws error for ++ condition
84        if(t.trim().length() == 0){
85          r.error = "Too many + signs";
86          return false;
87        }
88
```

```
89        //The rest of the string is therefore the species name
90        String varName = t.substring(subStringLength,t.length()).trim();
91
92        if (varName.equals("*")){break;}
93
94        //Check to see if species name is legal per SBML reqs
95        if(!JavaCodeParser.isValidName(varName)){
96          r.error = "Invalid name.";
97        }
98
99        //Check to see is var already exists to increment existing coefficient
100       if (trueIfProducts){
101
102         for(ReactionElement r1 : r.products){
103           if(r1.species.equals(varName)){
104             r1.coefficient += coefficient;
105             goBack = true;
106           }
107         }
108
109       }
110       else{
111
112         for(ReactionElement r1 : r.reactants){
113           if(r1.species.equals(varName)){
114             r1.coefficient += coefficient;
115             goBack = true;
116           }
117         }
118
```

```
119         }
120         //Will be assigned true if a species already exists in the current
                reaction
121         if (goBack){break;}
122
123         ReactionElement temp = new ReactionElement();
124         temp.coefficient = coefficient;
125         temp.species = varName;
126
127         if (trueIfProducts){
128           r.products.add(temp);
129         }
130         else{
131           r.reactants.add(temp);
132         }
133
134       }
135       return true;
136     }
137
138     //  private static boolean checkVarName(Reaction r, String name){
139     //
140     //    Matcher matcher = Pattern.compile("[^a-zA-Z0-9_]").matcher(name);
141     //
142     //    if(matcher.find()){
143     //      r.error = "Illegal species name " + name + ".  Legal characters are
              a-z, A-Z, 0-9 and underscore.";
144     //      return false;
145     //    }
146     //    return true;
```

```
147   //
148   //  }
149   //End Class
150 }
```

ReactionParser.java

A.4 JavaCodeParser

The following file tests each object to ensure no variable in the model is a reserved keyword in Java.

```
1  package edu.vcu.csl.ess;
2
3  import java.util.ArrayList;
4  import java.io.*;
5
6  public class JavaCodeParser {
7    public static boolean isAlpha(char c) {
8      return ((c >= 'a') && (c <= 'z')) || ((c >= 'A') && (c <= 'Z'));
9    }
10
11   public static boolean isDigit(char c) {
12     return ((c >= '0') && (c <= '9'));
13   }
14
15   public static boolean isUnderscore(char c) {
16     return (c == '_');
17   }
```

```
18
19  public static final String keywords[]
20                                      = {"abstract", "default", "if",
                                           "private",    "this",
21    "boolean",  "do",       "implements", "protected",  "throw",
22    "break",    "double",   "import",     "public",     "throws",
23    "byte",     "else",     "instanceof", "return",     "transient",
24    "case",     "extends",  "int",        "short",      "try",
25    "catch",    "final",    "interface",  "static",     "void",
26    "char",     "finally",  "long",       "strictfp",   "volatile",
27    "class",    "float",    "native",     "super",      "while",
28    "const",    "for",      "new",        "switch",
29    "continue", "goto",     "package",    "synchronized",
30    "true",     "false",    "null"};
31
32  public static boolean isKeyword(String str) {
33    for(int i=0; i<keywords.length; i++) {
34      if (keywords[i].equals(str))
35        return true;
36    }
37    return false;
38  }
39
40  public static boolean isValidName(String str) {
41    if (str == null)
42      return false;
43    if (str.length() == 0)
44      return false;
45    if (!isAlpha(str.charAt(0))) {
46      return false;
```

```
47      }
48      for(int i=1; i<str.length(); i++) {
49        if (!(isAlpha(str.charAt(i)) ||
50            isUnderscore(str.charAt(i)) ||
51            isDigit(str.charAt(i)))) {
52          return false;
53        }
54      }
55      if (isKeyword(str)) {
56        return false;
57      }
58      return true;
59    }
60
61    public static ArrayList<String> getNamesFromCode(String code) {
62      ArrayList<String> strs = new ArrayList<String>();
63      int i = 0;
64      while(i < code.length()) {
65        char c = code.charAt(i);
66        if (isAlpha(c) || isUnderscore(c)) {
67          String str = "" + c;
68          i++;
69          while (i < code.length()) {
70            c = code.charAt(i);
71            if (isDigit(c) || isAlpha(c) || isUnderscore(c)) {
72              str += c;
73              i++;
74            } else {
75              break;
76            }
```

```
77          }
78
79          boolean found = false;
80          for(int j=0; j<strs.size(); j++) {
81            if (strs.get(j).equals(str)) {
82              found = true;
83              break;
84            }
85          }
86          if ((!found) && (!isKeyword(str)))
87            strs.add(str);
88        } else {
89          i++;
90        }
91      }
92
93      return strs;
94    }
95
96    public static void main(String args[]) throws IOException {
97      BufferedReader in = new BufferedReader(new FileReader(args[0]));
98      String temp = in.readLine();
99      String str = "";
100     while (temp != null) {
101       str += temp + "\n";
102       temp = in.readLine();
103     }
104     in.close();
105     ArrayList<String> strs = getNamesFromCode(str);
106     for(int i=0; i<strs.size(); i++) {
```

```
107         System.out.println(strs.get(i));
108       }
109     }
110 }
```

JavaCodeParser.java

A.5 DependencyGen

The following file determines the dependencies between each model element (i.e. reactions, events, and triggers) in the system.

```
1  package edu.vcu.csl.ess;
2
3  import java.util.ArrayList;
4
5  import edu.vcu.csl.ess.RawModel.*;
6
7
8  class DependencyGen {
9    public static void calculateReactionDeltas(RawModel model) {
10
11     // For each reaction, compute the species delta = products - reactants.
12     for (Reaction reaction : model.reactions) {
13
14       // Subtract the reactants from the delta.
15       for (ReactionElement reactant : reaction.reactants) {
16         ReactionElement temp = new ReactionElement();
17         temp.species          = reactant.species;
```

```
18          temp.coefficient      = -reactant.coefficient;
19          reaction.delta.add(temp);
20        }
21
22        // Add the products to the delta.
23        for (ReactionElement product : reaction.products) {
24          boolean found = false;
25          for (ReactionElement delta : reaction.delta) {
26            if (delta.species.compareTo(product.species) == 0) {
27              found = true;
28              delta.coefficient += product.coefficient;
29              break;
30            }
31          }
32          if (!found) {
33            ReactionElement temp = new ReactionElement();
34            temp.species         = product.species;
35            temp.coefficient     = product.coefficient;
36            reaction.delta.add(temp);
37          }
38        }
39
40        // Eliminate all zero coefficient items.
41        for (ReactionElement delta : reaction.delta) {
42          if (delta.coefficient == 0) {
43            reaction.delta.remove(delta);
44          }
45        }
46      }
47    }
```

```
48
49  public static void calculateModelDependencies(RawModel model) {
50
51    // For each reaction, determine the reactions and triggers
52    // that depend on the execution of reaction.
53    for (Reaction reaction : model.reactions) {
54
55      // Look for dependent reactions
56      for (Reaction reaction2 : model.reactions) {
57        boolean affects = false;
58        for (ReactionElement element : reaction.delta) {
59          for(ReactionElement element2 : reaction2.reactants) {
60            if (element.species.compareTo(element2.species) == 0) {
61              affects = true;
62              break;
63            }
64          }
65        }
66        if (affects)
67          reaction.affectedReactionList.add(reaction2);
68      }
69
70      // Look for dependent triggers
71      for (Trigger trigger : model.triggers) {
72        ArrayList<String> conditionNames =
73          JavaCodeParser.getNamesFromCode(trigger.condition);
74        boolean affects = false;
75        for (ReactionElement delta : reaction.delta) {
76          for (String conditionName : conditionNames) {
77            if (conditionName.compareTo(delta.species) == 0) {
```

```
78                affects = true;
79                break;
80              }
81            }
82          }
83          if (affects)
84            reaction.affectedTriggerList.add(trigger);
85        }
86      }
87
88      // For each event, determine the reactions and triggers
89      // that depend on that event.
90      for (Event event : model.events) {
91        ArrayList<String> actionNames =
92          JavaCodeParser.getNamesFromCode(event.action);
93
94        for (Reaction reaction : model.reactions) {
95          boolean affects = false;
96          for (String actionName : actionNames) {
97            for (ReactionElement reactant : reaction.reactants) {
98              if (reactant.species.compareTo(actionName) == 0) {
99                affects = true;
100               break;
101             }
102           }
103           if (actionName.compareTo(reaction.name) == 0) {
104             affects = true;
105           }
106         }
107         if (affects)
```

```
108             event.affectedReactionList.add(reaction);
109         }
110
111         for (Trigger trigger : model.triggers) {
112           ArrayList<String> conditionNames =
113             JavaCodeParser.getNamesFromCode(trigger.condition);
114           boolean affects = false;
115           for (String actionName : actionNames) {
116             for (String conditionName : conditionNames) {
117               if (actionName.compareTo(conditionName) == 0) {
118                 affects = true;
119                 break;
120               }
121             }
122             if (actionName.compareTo(trigger.name) == 0) {
123               affects = true;
124               break;
125             }
126           }
127           if (affects)
128             event.affectedTriggerList.add(trigger);
129         }
130       }
131
132       // For each periodic, determine the reactions and triggers
133       // that depend on that event.
134       for (Periodic event : model.periodics) {
135         ArrayList<String> actionNames =
136           JavaCodeParser.getNamesFromCode(event.action);
137
```

```
138       for (Reaction reaction : model.reactions) {
139         boolean affects = false;
140         for (String actionName : actionNames) {
141           for (ReactionElement reactant : reaction.reactants) {
142             if (reactant.species.compareTo(actionName) == 0) {
143               affects = true;
144               break;
145             }
146           }
147           if (actionName.compareTo(reaction.name) == 0) {
148             affects = true;
149           }
150         }
151         if (affects)
152           event.affectedReactionList.add(reaction);
153       }
154
155       for (Trigger trigger : model.triggers) {
156         ArrayList<String> conditionNames =
157           JavaCodeParser.getNamesFromCode(trigger.condition);
158         boolean affects = false;
159         for (String actionName : actionNames) {
160           for (String conditionName : conditionNames) {
161             if (actionName.compareTo(conditionName) == 0) {
162               affects = true;
163               break;
164             }
165           }
166           if (actionName.compareTo(trigger.name) == 0) {
167             affects = true;
```

```
168             break;
169           }
170         }
171         if (affects)
172           event.affectedTriggerList.add(trigger);
173       }
174     }
175
176     // For each trigger, determine the reactions and triggers
177     // that depend on that event.
178     for (Trigger trigger : model.triggers) {
179       ArrayList<String> actionNames =
180         JavaCodeParser.getNamesFromCode(trigger.action);
181
182       for (Reaction reaction : model.reactions) {
183         boolean affects = false;
184         for (String actionName : actionNames) {
185           for (ReactionElement reactant : reaction.reactants) {
186             if (reactant.species.compareTo(actionName) == 0) {
187               affects = true;
188               break;
189             }
190           }
191           if (actionName.compareTo(reaction.name) == 0) {
192             affects = true;
193           }
194         }
195         if (affects)
196           trigger.affectedReactionList.add(reaction);
197       }
```

```
198
199         for (Trigger trigger2 : model.triggers) {
200           ArrayList<String> conditionNames =
201             JavaCodeParser.getNamesFromCode(trigger2.condition);
202           boolean affects = false;
203           for (String actionName : actionNames) {
204             for (String conditionName : conditionNames) {
205               if (actionName.compareTo(conditionName) == 0) {
206                 affects = true;
207                 break;
208               }
209             }
210             if (actionName.compareTo(trigger2.name) == 0) {
211               affects = true;
212               break;
213             }
214           }
215           if (affects)
216             trigger.affectedTriggerList.add(trigger2);
217         }
218       }
219     }
220 }
```

DependencyGen.java

A.6 ModelCodeGenC

The following file takes the data parsed and verified earlier and generates a C file that can be compiled and run to execute the simulation.

```
1  package edu.vcu.csl.ess;
2
3  import java.io.BufferedWriter;
4  import java.io.FileWriter;
5  import java.io.IOException;
6  import java.io.PrintWriter;
7  import java.util.ArrayList;
8  import java.util.Collections;
9
10 import edu.vcu.csl.ess.RawModel.*;
11
12 public class ModelCodeGenC {
13   public static void output(PrintWriter out, int indentCount, String str) {
14     for(int i=0; i<indentCount; i++)
15       out.print("  ");
16     out.println(str);
17   }
18
19   public static void generateHeaderHeader(PrintWriter out) {
20 //     output(out,0,"package proc;");
21 //     output(out,0,"import java.io.BufferedWriter;");
22 //     output(out,0,"import java.io.FileWriter;");
23 //     output(out,0,"import java.io.PrintWriter;");
24 //     output(out,0,"import java.io.IOException;");
25 //     output(out,0,"import java.util.Random;");
26 //     output(out,0,"");
```

```
27
28      output(out,0,"#include <stdio.h>");
29      output(out,0,"#include <stdlib.h>");
30      output(out,0,"#include <math.h>");
31      output(out,0,"#define true 1");
32      output(out,0,"#define false 0");
33    }
34
35
36
37    public static void generateHeaderMain(PrintWriter out,RawModel model) {
38  //      output(out,0,"package proc;");
39  //      output(out,0,"import java.io.BufferedWriter;");
40  //      output(out,0,"import java.io.FileWriter;");
41  //      output(out,0,"import java.io.PrintWriter;");
42  //      output(out,0,"import java.io.IOException;");
43  //      output(out,0,"import java.util.Random;");
44  //      output(out,0,"");
45
46      output(out,0,"#include \""+ model.filename +".h\"");
47    }
48
49    public static void generateGlobalSimVarDeclarations(PrintWriter out,
50        RawModel model,
51        String dir) {
52      output(out,1,"//Random simulator var declarations");
53      output(out,1,"double __currentTime = 0.0;");
54      output(out,1,"double __nextPrintTime = 0.0;");
55      output(out,1,"double __props[" + model.reactions.size() + "];");
56      output(out,1,"int __props_size = " + model.reactions.size() + ";");
```

```
57
58      output(out,1,"int __reactionSearchOrder[" + model.reactions.size() + "];")
            ;
59      output(out,1,"double __propensityTotal;");
60      output(out,1,"double __oldPropensityTotal;");
61      output(out,1,"double __nextReactionTime;");
62      output(out,1,"int __lastExecutedReactionIndex;");
63      output(out,1,"int __nextEventIndex = 0;");
64      output(out,1,"double __oldProp = 0.0;");
65
66      output(out,0,"");
67    }
68
69
70    public static void generateExecuteReactionFunctions(PrintWriter out,
          PrintWriter header, RawModel model) {
71      for(Reaction reaction : model.reactions) {
72
73        //Prototype
74        output(header,1,"void __reaction_execute_" + reaction.name + "();");
75
76        output(out,1,"void __reaction_execute_" +
77            reaction.name + "() {");
78        for(ReactionElement delta : reaction.delta)
79          output(out,2,"" + delta.species + " += " + delta.coefficient + ";");
80        output(out,2,"__" + reaction.name + "_count++;");
81        for (Reaction r : reaction.affectedReactionList)
82          output(out,2,"__reaction_recalculate_propensity_" + r.name + "();");
83        output(out,2,"__trigger_check_" + reaction.name + "();");
84        output(out,2,"__magnitudeRecalcCheck();");
```

```
85        output(out,1,"}");
86        output(out,0,"");
87      }
88    }
89
90    public static void generateNaiveExecuteReactionFunctions(PrintWriter out,
          PrintWriter header, RawModel model) {
91      for(Reaction reaction : model.reactions) {
92
93        //Prototype
94        output(header,1,"void __reaction_execute_" + reaction.name + "();");
95
96        output(out,1,"void __reaction_execute_" +
97            reaction.name + "() {");
98        for(ReactionElement delta : reaction.delta)
99          output(out,2,"" + delta.species + " += " + delta.coefficient + ";");
100       output(out,2,"__" + reaction.name + "_count++;");
101       for (Reaction r : reaction.affectedReactionList)
102         output(out,2,"__reaction_recalculate_propensity_" + r.name + "();");
103 //        output(out,2,"__trigger_check_" + reaction.name + "();");
104       output(out,2,"__magnitudeRecalcCheck();");
105       output(out,1,"}");
106       output(out,0,"");
107     }
108   }
109
110   public static void generateDMExecuteReactionFunctions(PrintWriter out,
        RawModel model) {
111     for(Reaction reaction : model.reactions) {
112       output(out,1,"void __reaction_execute_" +
```

```
113             reaction.name + "() {");
114         for(ReactionElement delta : reaction.delta)
115           output(out,2,"" + delta.species + " += " + delta.coefficient + ";");
116         output(out,2,"__" + reaction.name + "_count++;");
117         output(out,1,"}");
118         output(out,0,"");
119       }
120     }
121
122     public static void generateReactionPropensityFunctions(PrintWriter out,
          PrintWriter header, RawModel model) {
123       for(Reaction reaction : model.reactions) {
124         output(out,1,"double __reaction_calculate_propensity_" +
125             reaction.name + "() {");
126         //Prototype in header
127         output(header,1,"double __reaction_calculate_propensity_" +
128             reaction.name + "();");
129
130         output(out,1,"__reaction_propensity_count++;");
131
132         String str = "return " + reaction.name;
133
134         for(ReactionElement reactant : reaction.reactants) {
135           for(int i=0; i<reactant.coefficient; i++)  {
136             if (i == 0) {
137               str += "*" + reactant.species;
138             } else {
139               str += "*(" + reactant.species + "-" + i + ")/" + (i+1);
140             }
141           }
```

```
142         }
143         str += ";";
144         output(out,2,str);
145         output(out,1,"}");
146         output(out,0,"");
147       }
148
149       for(Reaction reaction : model.reactions) {
150         output(out,1,"void __reaction_recalculate_propensity_" +
151             reaction.name + "() {");
152         int index = model.reactions.indexOf(reaction);
153         output(out,2,"__propensityTotal -= __props[" + index + "];");
154         output(out,2,"__props[" + index + "] = __reaction_calculate_propensity_"
                +
155             reaction.name + "();");
156         output(out,2,"__propensityTotal += __props[" + index + "];");
157         output(out,1,"}");
158         output(out,0,"");
159       }
160
161     }
162
163     private static String getTriggerString(RawModel model,
164         ArrayList<Trigger> dependencies) {
165       String str = "";
166       for(Trigger t : model.triggers) {
167         if (dependencies.contains(t))
168           str += "1";
169         else {
170           str += "0";
```

```
171         }
172       }
173       return str;
174     }
175
176     private static int getTriggerStringIndex(ArrayList<String> triggerStrings,
177         String triggerString) {
178       int index = -1;
179       for(int i=0; i<triggerStrings.size(); i++) {
180         if (triggerString.compareTo(triggerStrings.get(i)) == 0) {
181           index = i;
182           break;
183         }
184       }
185       if (index == -1) {
186         index = triggerStrings.size();
187         triggerStrings.add(triggerString);
188       }
189       return index;
190     }
191
192     private static void generateTriggerCheckFunction(PrintWriter out,PrintWriter
          header,
193         RawModel model,
194         ArrayList<Trigger> dependencies,
195         ArrayList<String> triggerStrings,
196         String name) {
197       String triggerString = getTriggerString(model,dependencies);
198       int triggerStringIndex = getTriggerStringIndex(triggerStrings,
            triggerString);
```

```
199
200     output(header,1,"void __trigger_check_" + name + "();");
201
202     output(out,1,"void __trigger_check_" + name + "() {");
203     output(out,2,"__trigger_madness_" + triggerStringIndex + "();");
204     output(out,1,"}");
205     output(out,0,"");
206   }
207
208   public static void generateTriggerFunctions(PrintWriter out, PrintWriter
        header, RawModel model) {
209     ArrayList<String> triggerStrings = new ArrayList<String>();
210     int dependenceCount = 0;
211
212     for(Trigger t : model.triggers) {
213
214       //Prototype
215       output(header,1,"int __trigger_condition_" + t.name + "();");
216
217       output(out,1,"int __trigger_condition_" + t.name + "() {");
218       output(out,2,"__trigger_condition_checked_count++;");
219       output(out,2,"return " + t.name + " && (" + t.condition + ");");
220       output(out,1,"}");
221       output(out,0,"");
222     }
223
224     for(Trigger t : model.triggers) {
225
226       //Prototype
227       output(header,1,"void __trigger_action_" + t.name + "();");
```

```
228
229         output(out,1,"void __trigger_action_" + t.name + "() {");
230         output(out,2,"" + t.action);
231         output(out,2,"__" + t.name + "_count++;");
232         for (Reaction r : t.affectedReactionList)
233           output(out,2,"__reaction_recalculate_propensity_" + r.name + "();");
234         output(out,2,"__magnitudeRecalcCheck();");
235         output(out,1,"}");
236         output(out,0,"");
237       }
238
239       //Generate the all trigger string
240       String allTriggerString = "";
241       for(Trigger r : model.triggers)
242         allTriggerString += "1";
243       getTriggerStringIndex(triggerStrings,allTriggerString);
244
245       int reactionTriggerCount = 0;
246       int eventTriggerCount = 0;
247       int periodicTriggerCount = 0;
248       int triggerTriggerCount = 0;
249
250       for(Reaction r : model.reactions){
251         generateTriggerCheckFunction(out, header, model, r.affectedTriggerList,
252             triggerStrings, r.name);
253         reactionTriggerCount += r.affectedTriggerList.size();
254       }
255       for(Event r : model.events){
256         generateTriggerCheckFunction(out, header, model, r.affectedTriggerList,
257             triggerStrings, r.name);
```

```
258         eventTriggerCount += r.affectedTriggerList.size();
259       }
260       for(Periodic r : model.periodics){
261         generateTriggerCheckFunction(out, header, model, r.affectedTriggerList,
262             triggerStrings, r.name);
263         periodicTriggerCount += r.affectedTriggerList.size();
264       }
265       for(Trigger r : model.triggers){
266         generateTriggerCheckFunction(out, header, model, r.affectedTriggerList,
267             triggerStrings, r.name);
268         triggerTriggerCount += r.affectedTriggerList.size();
269       }
270
271       System.out.println("----MODEL CHARACTERIZATION DATA----");
272
273       System.out.println("Reactions affecting triggers = " +
              reactionTriggerCount);
274       System.out.println("Events affecting triggers    = " + eventTriggerCount);
275       System.out.println("Periodics affecting triggers = " +
              periodicTriggerCount);
276       System.out.println("Triggers affecting triggers  = " + triggerTriggerCount
              );
277       System.out.println("\n                            Total  = " + (
              reactionTriggerCount + eventTriggerCount + periodicTriggerCount +
              triggerTriggerCount));
278
279       int tempn = model.reactions.size() + model.events.size() + model.triggers.
              size() + model.periodics.size();
280
281       System.out.println("\n                       Model size = " + tempn);
```

```
282
283     System.out.println("----MODEL CHARACTERIZATION DATA----\n");
284
285     //After this loop, triggerStrings has been populated with all possible
            dependencies based on
286     //the affectedTriggerList of each collection.
287
288     //Code that limits the depth of these functions in the call stack with
            need to be added here
289     //as well as above in the generateTriggerCheckFunction method.
290
291     for(int l = 0; l < triggerStrings.size(); l++){ //Fixes exception, logic
            error?
292 //    for(String triggerString : triggerStrings) {
293
294       String triggerString = triggerStrings.get(l);
295
296       int triggerStringIndex = triggerStrings.indexOf(triggerString);
297       char triggerStringArray[] = triggerString.toCharArray();
298
299 //       System.out.println("TS = " + triggerString + "(" + triggerStringIndex
        +")");
300
301       output(header,1,"void __trigger_madness_" + triggerStringIndex + "();");
302
303       output(out,1,"void __trigger_madness_" + triggerStringIndex + "() {");
304       for(int i=0; i<triggerString.length(); i++) {
305         if (triggerString.charAt(i) == '1') {
306
307           dependenceCount++;
```

```
308
309             //Insert depth checking code
310             //And/or interconnected-ness checking
311
312             output(out,2,"if (__trigger_condition_" + model.triggers.get(i).name
                    + "()) {");
313             output(out,3,"__trigger_action_" + model.triggers.get(i).name + "();
                    ");
314             String temp = getTriggerString(model,model.triggers.get(i).
                    affectedTriggerList);
315             char tempArray[] = temp.toCharArray();
316             for(int j=0; j<triggerString.length(); j++) {
317               if (triggerStringArray[j] == '1')
318                 tempArray[j] = '1';
319             }
320             temp = new String(tempArray);
321             //This line may modify triggerStrings
322             //causes ConcurrentModificationException under new for loop notation
                    .
323             int tempIndex = getTriggerStringIndex(triggerStrings,temp);
324             //
325             output(out,3,"__trigger_madness_" + tempIndex + "();");
326             output(out,3,"return;");
327             output(out,2,"}");
328             triggerStringArray[i] = '0';
329           }
330         }
331         output(out,1,"}");
332         output(out,0,"");
333       }
```

```
334
335     //This verifies that dependency has nothing to do with stack overflow
336     System.out.println("Dependency count = " + dependenceCount);
337     System.out.println("Trigger count = " + triggerStrings.size());
338   }
339
340   public static void generateCalculateAllReactionPropensitiesFunction(
          PrintWriter out, RawModel model) {
341
342     output(out,1,"void __calculate_all_reaction_propensities() {");
343
344     output(out,2,"__propensityTotal = 0;");
345     for(int i=0; i<model.reactions.size(); i++) {
346       output(out,2,"__props[" + i + "] = " +
347           "__reaction_calculate_propensity_" + model.reactions.get(i).name + "
                ();");
348       output(out,2,"__propensityTotal += __props[" + i + "];");
349     }
350
351     output(out,2,"__oldPropensityTotal = __propensityTotal;");
352     output(out,1,"}");
353     output(out,0,"");
354   }
355
356
357   public static void generateConstantDeclarations(PrintWriter out, RawModel
          model) {
358     output(out,1,"// Constants");
359     for(Constant c : model.constants)
360       output(out,1,"double " + c.name + " = " + c.value + ";");
```

```
361     output(out,0,"");
362   }
363
364   public static void generateSpeciesDeclarations(PrintWriter out, RawModel
          model) {
365     output(out,1,"// Species");
366     for(Species sp : model.species)
367       output(out,1,"double " + sp.name + " = " + sp.population + ";");
368     output(out,0,"");
369   }
370
371   public static void generateRateConstantDeclarations(PrintWriter out,
          RawModel model) {
372     output(out,1,"//Rate constant declarations");
373     for(Reaction r : model.reactions)
374       output(out,1,"double " + r.name + " = "  + r.rate + ";");
375     output(out,0,"");
376   }
377
378   public static void generateCountDeclarations(PrintWriter out, RawModel model
          ) {
379     output(out,1,"// Count declarations");
380     for(Reaction r : model.reactions)
381       output(out,1,"int __" + r.name + "_count = "  + "0;");
382     output(out,1,"int __reaction_total_count = "  + "0;");
383
384     for(Event e : model.events)
385       output(out,1,"int __" + e.name + "_count = "  + "0;");
386     output(out,1,"int __event_total_count = "  + "0;");
387
```

```
388     for(Periodic p : model.periodics)
389       output(out,1,"int __" + p.name + "_count = " + "0;");
390     output(out,1,"int __periodic_total_count = " + "0;");
391
392     for(Trigger t : model.triggers)
393       output(out,1,"int __" + t.name + "_count = " + "0;");
394     output(out,1,"int __trigger_total_count = " + "0;");
395
396     output(out,1,"int __total_model_count = " + "0;");
397
398     output(out,1,"long int __reaction_propensity_count = " + "0;");
399     output(out,1,"long int __trigger_condition_checked_count = " + "0;");
400     output(out,1,"long int __event_condition_checked_count = " + "0;");
401
402     output(out,0,"");
403   }
404
405   public static void generateTotals(PrintWriter out, int indentCount,RawModel
        model) {
406
407     for(Reaction r : model.reactions)
408       output(out,indentCount,"__reaction_total_count += __" + r.name + "
            _count;");
409
410     for(Periodic p : model.periodics)
411       output(out,indentCount,"__periodic_total_count += __" + p.name + "
            _count;");
412
413     for(Trigger t : model.triggers)
414       output(out,indentCount,"__trigger_total_count += __" + t.name + "_count
```

```
        ;");
415
416     output(out,indentCount,"__total_model_count = __reaction_total_count +
            __periodic_total_count + __trigger_total_count;");
417
418     output(out,indentCount,"printf(\"Total reaction count              = %d \\t
            (%f)\\r\\n\",__reaction_total_count,(float)__reaction_total_count/(
            float)__total_model_count);");
419     output(out,indentCount,"printf(\"Total periodic count              = %d \\t
            (%f)\\r\\n\",__periodic_total_count,(float)__periodic_total_count/(
            float)__total_model_count);");
420     output(out,indentCount,"printf(\"Total trigger count               = %d \\t
            (%f)\\r\\n\",__trigger_total_count,(float)__trigger_total_count/(float
            )__total_model_count);");
421
422     output(out,indentCount,"printf(\"Reaction propensity count         = %ld \\r
            \\n\",__reaction_propensity_count);");
423     output(out,indentCount,"printf(\"Trigger condition checked count = %ld \\r
            \\n\",__trigger_condition_checked_count);");
424     output(out,indentCount,"printf(\"Event condition checked           = %ld \\r
            \\n\",__event_condition_checked_count);");
425
426     output(out,indentCount,"");
427   }
428
429   public static void generateEnableDeclarations(PrintWriter out, RawModel
        model) {
430     output(out,1,"// Enable declarations");
431     for(Event e : model.events)
432       output(out,1,"int " + e.name + " = true;");
```

```
433     for(Periodic p : model.periodics)
434       output(out,1,"int " + p.name + " = true;");
435     for(Trigger t : model.triggers)
436       output(out,1,"int " + t.name + " = true;");
437     output(out,0,"");
438   }
439
440   public static void generateOutputFunction(PrintWriter out, RawModel model) {
441     output(out,1,"void __output(double time) {");
442 //    output(out,2,"__out.print(time);");
443     output(out,2,"fprintf(__out,\"%.10f\",time);");
444
445     for (Species s : model.species)
446 //      output(out,2,"__out.print(\",\"+" + s.name + ");");
447       output(out,2,"fprintf(__out,\",%.10f\"," + s.name +");");
448     for (Reaction r : model.reactions)
449 //      output(out,2,"__out.print(\",\"+__" + r.name + "_count);");
450       output(out,2,"fprintf(__out,\",%d\",__" + r.name + "_count);");
451     for (Event e : model.events)
452 //      output(out,2,"__out.print(\",\"+__" + e.name + "_count);");
453       output(out,2,"fprintf(__out,\",%d\",__" + e.name + "_count);");
454     for (Periodic p : model.periodics)
455 //      output(out,2,"__out.print(\",\"+__" + p.name + "_count);");
456       output(out,2,"fprintf(__out,\",%d\",__" + p.name + "_count);");
457     for (Trigger t : model.triggers)
458 //      output(out,2,"__out.print(\",\"+__" + t.name + "_count);");
459       output(out,2,"fprintf(__out,\",%d\",__" + t.name + "_count);");
460
461 //    output(out,2,"__out.println();");
462     output(out,2,"fprintf(__out,\"\\r\\n\");");
```

```
463       output(out,1,"}");
464       output(out,1,"");
465     }
466
467     public static void generateOutputHeaderFunction(PrintWriter out, RawModel
            model) {
468       output(out,1,"void __output_header() {");
469 //      output(out,2,"__out.print(\"time\");");
470       output(out,2,"fprintf(__out,\"time\");");
471
472       for (Species s : model.species)
473 //        output(out,2,"__out.print(\",\" + s.name + \"\");");
474         output(out,2,"fprintf(__out,\",\" + s.name + \"\");");
475       for (Reaction r : model.reactions)
476 //        output(out,2,"__out.print(\",\" + r.name + \"\");");
477         output(out,2,"fprintf(__out,\",\" + r.name + \"\");");
478       for (Event e : model.events)
479 //        output(out,2,"__out.print(\",\" + e.name + \"\");");
480         output(out,2,"fprintf(__out,\",\" + e.name + \"\");");
481       for (Periodic p : model.periodics)
482 //        output(out,2,"__out.print(\",\" + p.name + \"\");");
483         output(out,2,"fprintf(__out,\",\" + p.name + \"\");");
484       for (Trigger t : model.triggers)
485 //        output(out,2,"__out.print(\",\" + t.name + \"\");");
486         output(out,2,"fprintf(__out,\",\" + t.name + \"\");");
487 //      output(out,2,"__out.println();");
488       output(out,2,"fprintf(__out,\"\\r\\n\");");
489       output(out,1,"}");
490       output(out,1,"");
491     }
```

```
492
493   public static void generateMainStart(PrintWriter out) {
494     output(out,1,"int main(int argc, char** argv){");
495     output(out,0,"");
496   }
497
498   public static void generateMainInitPrintWriter(PrintWriter out, String dir,
          RawModel model) {
499
500     output(out,2,"__out = fopen(\"" + model.filename + ".csv\",\"wt\");");
501
502     output(out,2,"srand ( " + model.seed + ");");
503
504 //     output(out,2,"__out = new PrintWriter(new BufferedWriter(new FileWriter
        (\""
505 //         + dir + model.filename + ".csv\")));");
506     output(out,0,"");
507   }
508
509   public static void generateMainOutputHeader(PrintWriter out) {
510     output(out,2,"__output_header();");
511     output(out,0,"");
512   }
513
514   public static void generateMainInitReactionSearchOrder(PrintWriter out,
          RawModel model) {
515     output(out,2,"int __j;");
516     output(out,2,"for(__j=0; __j<" + model.reactions.size() + "; __j++)");
517     output(out,3,"__reactionSearchOrder[__j] = __j;");
518     output(out,0,"");
```

```
519   }
520
521   public static void generateMainCalculateReactionPropensities(PrintWriter out
          ) {
522     output(out,2,"__calculate_all_reaction_propensities();");
523     output(out,0,"");
524   }
525
526   public static void generateIllgalStateCheckFunction(PrintWriter out,
          RawModel model){
527     output(out,1,"void __illegalStateCheck() {");
528
529     for(Species sp : model.species){
530       output(out,2,"if (" + sp.name + "< 0) {");
531       output(out,3,"printf(\"ILLEGAL STATE @ %.10f\",__currentTime);");
532       output(out,3,"printf(\"%s has a negative population\",\"" + sp.name + "
              \");");
533       output(out,3,"fclose(__out);");
534       output(out,3,"exit(1);");
535       output(out,2,"}");
536     }
537     for(Reaction r : model.reactions){
538       output(out,2,"if (" + r.name + "< 0) {");
539       output(out,3,"printf(\"ILLEGAL STATE @ %.10f\",__currentTime);");
540       output(out,3,"printf(\"%s has a negative rate constant\",\"" + r.name +
              "\");");
541       output(out,3,"fclose(__out);");
542       output(out,3,"exit(1);");
543       output(out,2,"}");
544     }
```

```
545       output(out,1,"}");
546       output(out,1,"");
547     }
548
549     public static void generateMagnitudeRecalcCheckFunction(PrintWriter out,
          RawModel model) {
550       output(out,1,"void __magnitudeRecalcCheck(){");
551       output(out,2,"if(__propensityTotal < (__oldPropensityTotal / 2) ||");
552       output(out,2,"    __propensityTotal > (2 * __oldPropensityTotal)) {");
553       output(out,3,"__calculate_all_reaction_propensities();");
554       output(out,2,"}");
555       output(out,1,"}");
556       output(out,0,"");
557     }
558
559     public static void generateExecuteEventFunctions(PrintWriter out,
          PrintWriter header,RawModel model) {
560       for(Event e : model.events) {
561
562         //Prototype
563         output(header,1,"void __event_execute_" + e.name + "();");
564
565         output(out,1,"void __event_execute_" + e.name + "() {");
566         output(out,2,"if (!" + e.name + ") return;");
567         output(out,2,e.action);
568         output(out,2,"__" + e.name + "_count++;");
569         for (Reaction r : e.affectedReactionList)
570           output(out,2,"__reaction_recalculate_propensity_" + r.name + "();");
571         output(out,2,"__trigger_check_" + e.name + "();");
572         output(out,2,"__magnitudeRecalcCheck();");
```

```
573         output(out,1,"}");
574         output(out,0,"");
575       }
576
577       for(Periodic e : model.periodics) {
578
579         //Prototype
580         output(header,1,"void __event_execute_" + e.name + "();");
581
582         output(out,1,"void __event_execute_" + e.name + "() {");
583         output(out,2,"if (!" + e.name + ") return;");
584         output(out,2,e.action);
585         output(out,2,"__" + e.name + "_count++;");
586         for (Reaction r : e.affectedReactionList)
587           output(out,2,"__reaction_recalculate_propensity_" + r.name + "();");
588         output(out,2,"__trigger_check_" + e.name + "();");
589         output(out,2,"__magnitudeRecalcCheck();");
590         output(out,1,"}");
591         output(out,0,"");
592       }
593     }
594
595     public static void generateDMExecuteEventFunctions(PrintWriter out,
          PrintWriter header,RawModel model) {
596       for(Event e : model.events) {
597
598         //Prototype
599         output(header,1,"void __event_execute_" + e.name + "();");
600
601         output(out,1,"void __event_execute_" + e.name + "() {");
```

```
602         output(out,2,"if (!" + e.name + ") return;");
603         output(out,2,e.action);
604         output(out,2,"__" + e.name + "_count++;");
605         for (Reaction r : e.affectedReactionList)
606           output(out,2,"__reaction_recalculate_propensity_" + r.name + "();");
607         output(out,1,"}");
608         output(out,0,"");
609       }
610
611       for(Periodic e : model.periodics) {
612
613         //Prototype
614         output(header,1,"void __event_execute_" + e.name + "();");
615
616         output(out,1,"void __event_execute_" + e.name + "() {");
617         output(out,2,"if (!" + e.name + ") return;");
618         output(out,2,e.action);
619         output(out,2,"__" + e.name + "_count++;");
620         for (Reaction r : e.affectedReactionList)
621           output(out,2,"__reaction_recalculate_propensity_" + r.name + "();");
622         output(out,1,"}");
623         output(out,0,"");
624       }
625     }
626
627     public static void generateExecuteNextEventFunction(PrintWriter out,
          RawModel model) {
628       ArrayList<Event> sortedEventList = new ArrayList<Event>();
629
630       for(Event e : model.events)
```

```
631         sortedEventList.add(e);
632
633       for(Periodic p : model.periodics) {
634         double time = Double.parseDouble(p.start);
635         double end = Double.parseDouble(p.end);
636
637         while (time <= end){
638           Event e = new Event();
639           e.name = p.name;
640           e.time = Double.toString(time);
641           e.action = p.action;
642           e.comment = p.comment;
643           sortedEventList.add(e);
644           time += Double.parseDouble(p.period);
645         }
646       }
647
648       Collections.sort(sortedEventList, new EventSort());
649
650       //TODO Figure out C syntax for positive infinity (-log (0.0))
651
652       if (sortedEventList.size() == 0) {
653         output(out,1,"double __nextEventTime = INFINITY;");
654       } else {
655         output(out,1,"double __nextEventTime = " + sortedEventList.get(0).time +
                 ";");
656       }
657
658       output(out,1,"void __executeNextEvent() {");
659       output(out,2,"switch(__nextEventIndex) {");
```

```
660
661     for(int i=0; i<sortedEventList.size(); i++) {
662       output(out,3,"case " + i + ":");
663       output(out,4,"if (" + sortedEventList.get(i).name + "){");
664       output(out,5,"__event_execute_" + sortedEventList.get(i).name + "();");
665       output(out,4,"}");
666       if (i+1 < sortedEventList.size()) {
667         output(out,4,"__nextEventTime = " + sortedEventList.get(i+1).time + ";
              ");
668       } else {
669         output(out,4,"__nextEventTime = INFINITY;");
670       }
671       output(out,4,"break;");
672     }
673     output(out,2,"}");
674     output(out,2,"__nextEventIndex++;");
675     output(out,1,"}");
676     output(out,0,"");
677   }
678
679   public static void generateExecuteNextReactionFunction(PrintWriter out,
        RawModel model){
680     output(out,1,"int __selectReactionToExecute() {");
681     output(out,2,"double __selector;");
682     output(out,2,"int __i;");
683     output(out,2,"int __reactionIndex;");
684     output(out,2,"while (true){");
685     output(out,3,"__selector = ((double)rand())/((double)RAND_MAX) *
          __propensityTotal;");
686     output(out,3,"for(__i=0;__i<__props_size;__i++){");
```

```
687     output(out,4,"__reactionIndex = __reactionSearchOrder[__i];");
688     output(out,4,"__selector -= __props[__reactionIndex];");
689     output(out,4,"if (__selector <= 0) return __i;");
690     output(out,3,"}");
691     output(out,2,"}");
692     output(out,1,"}");
693     output(out,0,"");
694
695
696     output(out,1,"void __executeNextReaction(){");
697     output(out,2,"int __i = __selectReactionToExecute();");
698     output(out,2,"switch(__reactionSearchOrder[__i]){");
699     for(int i=0;i<model.reactions.size();i++){
700       output(out,3,"case " + i + ":");
701       output(out,4,"  __reaction_execute_" + model.reactions.get(i).name + "()
            ;");
702       output(out,4,"break;");
703     }
704     output(out,2,"}");
705     output(out,0,"");
706
707     //Bubble up executed reaction
708     output(out,2,"if (__i != 0){");
709     output(out,3,"__lastExecutedReactionIndex = __reactionSearchOrder[__i];");
710     output(out,3,"__reactionSearchOrder[__i] = __reactionSearchOrder[__i-1];")
          ;
711     output(out,3,"__reactionSearchOrder[__i-1] = __lastExecutedReactionIndex;"
          );
712     output(out,2,"} else {");
713     output(out,3,"__lastExecutedReactionIndex = 0;}");
```

```
714     output(out,1,"}");
715     output(out,0,"");
716   }
717
718   public static void generateDMExecuteNextReactionFunction(PrintWriter out,
          RawModel model){
719     output(out,1,"int __selectReactionToExecute() {");
720     output(out,2,"double __selector;");
721     output(out,2,"int __i;");
722     output(out,2,"int __reactionIndex;");
723     output(out,2,"while (true){");
724     output(out,3,"__selector = ((double)rand())/((double)RAND_MAX) *
            __propensityTotal;");
725     output(out,3,"for(__i=0;__i<__props_size;__i++){");
726     output(out,4,"__selector -= __props[__i];");
727     output(out,4,"if (__selector <= 0) return __i;");
728     output(out,3,"}");
729     output(out,2,"}");
730     output(out,1,"}");
731     output(out,0,"");
732
733
734     output(out,1,"static void __executeNextReaction(){");
735     output(out,2,"int __i = __selectReactionToExecute();");
736     output(out,2,"switch(__i){");
737     for(int i=0;i<model.reactions.size();i++){
738       output(out,3,"case " + i + ":");
739       output(out,4,"  __reaction_execute_" + model.reactions.get(i).name + "()
            ;");
740       output(out,4,"break;");
```

```
741         }
742         output(out,2,"}");
743         output(out,0,"");
744 
745         output(out,1,"}");
746         output(out,0,"");
747     }
748 
749     // TODO: Make all generated files into parameters
750 
751     public static void generateOutputFile(RawModel model){
752 
753 
754         //Calls to finish setting up the model
755         DependencyGen.calculateReactionDeltas(model);
756         DependencyGen.calculateModelDependencies(model);
757 
758         String dir = "proc/";
759 
760         try {
761             PrintWriter procClass = new PrintWriter(new BufferedWriter(new
                    FileWriter(dir + model.filename + ".c")));
762             PrintWriter procClassHeader = new PrintWriter(new BufferedWriter(new
                    FileWriter(dir + model.filename + ".h")));
763             generateHeaderMain(procClass,model);
764             generateHeaderHeader(procClassHeader);
765 //            output(procClass,0,"public class " + model.filename + " {");
766 
767             output(procClassHeader,0,"FILE *__out;");
768 
```

```
769         generateGlobalSimVarDeclarations(procClass, model, dir);
770         generateConstantDeclarations(procClass,model);
771         generateSpeciesDeclarations(procClass,model);
772         generateRateConstantDeclarations(procClass,model);
773         generateCountDeclarations(procClass,model);
774         generateEnableDeclarations(procClass,model);
775
776         generateOutputHeaderFunction(procClass,model);
777         generateOutputFunction(procClass,model);
778         generateIllgalStateCheckFunction(procClass,model);
779         generateReactionPropensityFunctions(procClass,procClassHeader,model);
780         generateCalculateAllReactionPropensitiesFunction(procClass,model);
781         generateMagnitudeRecalcCheckFunction(procClass,model);
782         generateExecuteEventFunctions(procClass,procClassHeader,model);
783         generateExecuteReactionFunctions(procClass,procClassHeader,model);
784         generateExecuteNextEventFunction(procClass,model);
785         generateExecuteNextReactionFunction(procClass,model);
786         generateTriggerFunctions(procClass,procClassHeader,model);
787
788         generateMainStart(procClass);
789         generateMainInitPrintWriter(procClass, dir, model);
790         generateMainOutputHeader(procClass);
791         generateMainInitReactionSearchOrder(procClass,model);
792         generateMainCalculateReactionPropensities(procClass);
793
794         output(procClass,2,"// Check all triggers at startup.");
795         output(procClass,2,"__trigger_madness_0();");
796
797         output(procClass,2,"while(true) {");
798
```

```
799         output(procClass,3,"__nextReactionTime = __currentTime-(log(((double)
                rand())/((double)RAND_MAX)))/__propensityTotal;");
800         output(procClass,0,"");
801
802         output(procClass,3,"__event_condition_checked_count++;");
803         output(procClass,3,"if (__nextEventTime < __nextReactionTime) {");
804         output(procClass,4,"__currentTime = __nextEventTime;");
805         output(procClass,3,"} else {");
806         output(procClass,4,"__currentTime = __nextReactionTime;");
807         output(procClass,3,"}");
808         output(procClass,0,"");
809
810         output(procClass,3,"while (__currentTime > __nextPrintTime) {");
811         output(procClass,4,"__output(__nextPrintTime);");
812         output(procClass,4,"__nextPrintTime += " + model.interval + ";");
813         output(procClass,4,"if (__nextPrintTime >= " + model.end + ") {");
814         generateTotals(procClass,4,model);
815         output(procClass,5,"fclose(__out);");
816         output(procClass,5,"exit(0);");
817         output(procClass,4,"}");
818         output(procClass,3,"}");
819         output(procClass,0,"");
820
821         output(procClass,3,"if (__currentTime == __nextEventTime) {");
822         output(procClass,4,"__executeNextEvent();");
823         output(procClass,3,"} else {");
824         output(procClass,4,"__executeNextReaction();");
825         output(procClass,3,"}");
826         output(procClass,0,"");
827
```

```
828         output(procClass,3,"__illegalStateCheck();");
829         output(procClass,2,"}");
830         output(procClass,0,"");
831 
832         output(procClass,1,"}");
833 //          output(procClass,0,"}");
834 
835         procClass.close();
836         procClassHeader.close();
837 
838 
839       } catch (IOException e) {
840         e.printStackTrace();
841         System.exit(1);
842       }
843     }
844 
845     public static void generateNaiveOutputFile(RawModel model){
846       //Calls to finish setting up the model
847       DependencyGen.calculateReactionDeltas(model);
848       DependencyGen.calculateModelDependencies(model);
849 
850       String dir = "proc/";
851 
852       try {
853         PrintWriter procClass = new PrintWriter(new BufferedWriter(new
                FileWriter(dir + model.filename + ".c")));
854         PrintWriter procClassHeader = new PrintWriter(new BufferedWriter(new
                FileWriter(dir + model.filename + ".h")));
855         generateHeaderMain(procClass,model);
```

```
856         generateHeaderHeader(procClassHeader);
857 //          output(procClass,0,"public class " + model.filename + " {");
858 
859         output(procClassHeader,0,"FILE *__out;");
860 
861         generateGlobalSimVarDeclarations(procClass, model,dir);
862         generateConstantDeclarations(procClass,model);
863         generateSpeciesDeclarations(procClass,model);
864         generateRateConstantDeclarations(procClass,model);
865         generateCountDeclarations(procClass,model);
866         generateEnableDeclarations(procClass,model);
867 
868         generateOutputHeaderFunction(procClass,model);
869         generateOutputFunction(procClass,model);
870         generateIllgalStateCheckFunction(procClass,model);
871         generateReactionPropensityFunctions(procClass,procClassHeader,model);
872         generateCalculateAllReactionPropensitiesFunction(procClass,model);
873         generateMagnitudeRecalcCheckFunction(procClass,model);
874         generateDMExecuteEventFunctions(procClass,procClassHeader,model);
875         generateNaiveExecuteReactionFunctions(procClass,procClassHeader,model);
876         generateExecuteNextEventFunction(procClass,model);
877         generateExecuteNextReactionFunction(procClass,model);
878 //          generateTriggerFunctions(procClass,procClassHeader,model);
879 
880         generateMainStart(procClass);
881         generateMainInitPrintWriter(procClass, dir, model);
882         generateMainOutputHeader(procClass);
883         generateMainInitReactionSearchOrder(procClass,model);
884         generateMainCalculateReactionPropensities(procClass);
885 
```

```
886         output(procClass,2,"// Check all triggers at startup.");
887 //        output(procClass,2,"__trigger_madness_0();");
888
889         output(procClass,2,"while(true) {");
890
891         // Check all triggers until they are all satisfied.
892
893         output(procClass,3,"int triggerMadnessDone = false;");
894         output(procClass,3,"int triggerFired = false;");
895         output(procClass,3,"while(!triggerMadnessDone) {");
896         output(procClass,4,"triggerMadnessDone = true;");
897
898         // check each trigger condition.
899         // if true...
900         //   fire action..
901         //   increment trig counter...
902         //   triggerMadnessDone = false
903
904         for(Trigger t : model.triggers){
905           output(procClass,4,"__trigger_condition_checked_count++;");
906           output(procClass,4,"if(" + t.condition + "){");
907           output(procClass,5,t.action);
908           output(procClass,5,"__" + t.name + "_count++;");
909           output(procClass,5,"triggerFired = true;");
910           output(procClass,5,"triggerMadnessDone = false;");
911           output(procClass,4,"}");
912         }
913
914
915         output(procClass,3,"}");
```

```
916
917         output(procClass,3,"if(triggerFired){");
918         generateMainCalculateReactionPropensities(procClass);
919         output(procClass,3,"}");
920
921         output(procClass,3,"__nextReactionTime = __currentTime-(log(((double)
                rand())/((double)RAND_MAX)))/__propensityTotal;");
922         output(procClass,0,"");
923
924         output(procClass,3,"__event_condition_checked_count++;");
925         output(procClass,3,"if (__nextEventTime < __nextReactionTime) {");
926         output(procClass,4,"__currentTime = __nextEventTime;");
927         output(procClass,3,"} else {");
928         output(procClass,4,"__currentTime = __nextReactionTime;");
929         output(procClass,3,"}");
930         output(procClass,0,"");
931
932         output(procClass,3,"while (__currentTime > __nextPrintTime) {");
933         output(procClass,4,"__output(__nextPrintTime);");
934         output(procClass,4,"__nextPrintTime += " + model.interval + ";");
935         output(procClass,4,"if (__nextPrintTime >= " + model.end + ") {");
936         generateTotals(procClass,4,model);
937         output(procClass,5,"fclose(__out);");
938         output(procClass,5,"exit(0);");
939         output(procClass,4,"}");
940         output(procClass,3,"}");
941         output(procClass,0,"");
942
943         output(procClass,3,"if (__currentTime == __nextEventTime) {");
944         output(procClass,4,"__executeNextEvent();");
```

```
945         output(procClass,3,"} else {");
946         output(procClass,4,"__executeNextReaction();");
947         output(procClass,3,"}");
948         output(procClass,0,"");
949 
950         output(procClass,3,"__illegalStateCheck();");
951         output(procClass,2,"}");
952         output(procClass,0,"");
953 
954         output(procClass,1,"}");
955 //        output(procClass,0,"}");
956 
957         procClass.close();
958         procClassHeader.close();
959 
960 
961       } catch (IOException e) {
962         e.printStackTrace();
963         System.exit(1);
964       }
965     }
966 
967     public static void generateDirectMethodOutputFile(RawModel model){
968 
969 
970       //Calls to finish setting up the model
971       DependencyGen.calculateReactionDeltas(model);
972 //      DependencyGen.calculateModelDependencies(model);
973 
974       String dir = "proc/";
```

```
975
976     try {
977       PrintWriter procClass = new PrintWriter(new BufferedWriter(new
              FileWriter(dir + model.filename + ".c")));
978       PrintWriter procClassHeader = new PrintWriter(new BufferedWriter(new
              FileWriter(dir + model.filename + ".h")));
979       generateHeaderMain(procClass,model);
980       generateHeaderHeader(procClassHeader);
981 //        output(procClass,0,"public class " + model.filename + " {");
982
983       output(procClassHeader,1,"FILE *__out;");
984
985       generateGlobalSimVarDeclarations(procClass, model,dir);
986       generateConstantDeclarations(procClass,model);
987       generateSpeciesDeclarations(procClass,model);
988       generateRateConstantDeclarations(procClass,model);
989       generateCountDeclarations(procClass,model);
990       generateEnableDeclarations(procClass,model);
991
992       generateOutputHeaderFunction(procClass,model);
993       generateOutputFunction(procClass,model);
994       generateIllgalStateCheckFunction(procClass,model);
995
996       generateReactionPropensityFunctions(procClass,procClassHeader,model);
997       generateCalculateAllReactionPropensitiesFunction(procClass,model);
998 /*        generateMagnitudeRecalcCheckFunction(procClass,model); */
999       generateDMExecuteEventFunctions(procClass,procClassHeader,model);
1000      generateDMExecuteReactionFunctions(procClass,model);
1001      generateExecuteNextEventFunction(procClass,model);
1002      generateDMExecuteNextReactionFunction(procClass,model);
```

```
1003 //        generateTriggerFunctions(procClass, procClassHeader, model);
1004
1005         generateMainStart(procClass);
1006         generateMainInitPrintWriter(procClass, dir, model);
1007         generateMainOutputHeader(procClass);
1008
1009
1010         output(procClass,2,"while(true) {");
1011
1012         // Check all triggers until they are all satisfied.
1013
1014         output(procClass,3,"int triggerMadnessDone = false;");
1015         output(procClass,3,"while(!triggerMadnessDone) {");
1016         output(procClass,4,"triggerMadnessDone = true;");
1017
1018         // check each trigger condition.
1019         // if true...
1020         //   fire action..
1021         //   increment trig counter...
1022         //   triggerMadnessDone = false
1023
1024         for(Trigger t : model.triggers){
1025           output(procClass,4,"__trigger_condition_checked_count++;");
1026           output(procClass,4,"if(" + t.condition + "){");
1027           output(procClass,5,t.action);
1028           output(procClass,5,"__" + t.name + "_count++;");
1029           output(procClass,5,"triggerMadnessDone = false;");
1030           output(procClass,4,"}");
1031         }
1032
```

```
1033
1034        output(procClass,3,"}");
1035
1036        output(procClass,3,"__calculate_all_reaction_propensities();");
1037
1038        output(procClass,3,"__nextReactionTime = __currentTime-(log(((double)
                rand())/((double)RAND_MAX)))/__propensityTotal;");
1039        output(procClass,0,"");
1040
1041        output(procClass,3,"if (__nextEventTime < __nextReactionTime) {");
1042        output(procClass,4,"__currentTime = __nextEventTime;");
1043        output(procClass,3,"} else {");
1044        output(procClass,4,"__currentTime = __nextReactionTime;");
1045        output(procClass,3,"}");
1046        output(procClass,0,"");
1047
1048        output(procClass,3,"while (__currentTime > __nextPrintTime) {");
1049        output(procClass,4,"__output(__nextPrintTime);");
1050        output(procClass,4,"__nextPrintTime += " + model.interval + ";");
1051        output(procClass,4,"if (__nextPrintTime >= " + model.end + ") {");
1052        generateTotals(procClass,4,model);
1053        output(procClass,5,"fclose(__out);");
1054        output(procClass,5,"exit(0);");
1055        output(procClass,4,"}");
1056        output(procClass,3,"}");
1057        output(procClass,0,"");
1058
1059        // Execute either the event or the reaction - WITHOUT DEPENDENDCY GRAPH
1060
1061
```

```
1062        output(procClass,3,"__event_condition_checked_count++;");
1063        output(procClass,3,"if (__currentTime == __nextEventTime) {");
1064        output(procClass,4,"__executeNextEvent();");
1065        output(procClass,3,"} else {");
1066        output(procClass,4,"__executeNextReaction();");
1067        output(procClass,3,"}");
1068        output(procClass,0,"");
1069
1070
1071
1072        output(procClass,3,"__illegalStateCheck();");
1073        output(procClass,2,"}");
1074        output(procClass,0,"");
1075
1076        output(procClass,1,"}");
1077 //        output(procClass,0,"}");
1078
1079        procClass.close();
1080        procClassHeader.close();
1081
1082
1083      } catch (IOException e) {
1084        e.printStackTrace();
1085        System.exit(1);
1086      }
1087    }
1088
1089
1090 }
```

A.7 ModelCodeGen

This file generates a similar output file to that of ModelCodeGenC only in this case Java code is generated as opposed to C code.

```
1  package edu.vcu.csl.ess;
2
3  import java.io.BufferedWriter;
4  import java.io.FileWriter;
5  import java.io.IOException;
6  import java.io.PrintWriter;
7  import java.util.ArrayList;
8  import java.util.Collections;
9
10 import edu.vcu.csl.ess.RawModel.*;
11
12 public class ModelCodeGen {
13   public static void output(PrintWriter out, int indentCount, String str) {
14     for(int i=0; i<indentCount; i++)
15       out.print("  ");
16     out.println(str);
17   }
18
19   public static void generateHeader(PrintWriter out) {
20     output(out,0,"package proc;");
21     output(out,0,"import java.io.BufferedWriter;");
22     output(out,0,"import java.io.FileWriter;");
23     output(out,0,"import java.io.PrintWriter;");
24     output(out,0,"import java.io.IOException;");
25     output(out,0,"import java.util.Random;");
26     output(out,0,"");
```

```
27    }
28
29    public static void generateGlobalSimVarDeclarations(PrintWriter out,
30        RawModel model,
31        String dir) {
32      output(out,1,"//Random simulator var declarations");
33      output(out,1,"static double __currentTime = 0.0;");
34      output(out,1,"static double __nextPrintTime = 0.0;");
35      output(out,1,"static double __props[] = new double[" + model.reactions.
            size() + "];");
36      output(out,1,"static int[] __reactionSearchOrder = new int[" + model.
            reactions.size() + "];");
37      output(out,1,"static double __propensityTotal;");
38      output(out,1,"static double __oldPropensityTotal;");
39      output(out,1,"static double __nextReactionTime;");
40      output(out,1,"static int __lastExecutedReactionIndex;");
41      output(out,1,"static int __nextEventIndex = 0;");
42      output(out,1,"static double __oldProp = 0.0;");
43      output(out,1,"static Random __generator = new Random(" + model.seed + ");"
            );
44      output(out,1,"static PrintWriter __out;");
45      //Trigger stack overflow fix:
46      output(out,1,"static int __triggersFired = 0;");
47      //End trigger stack overflow fix
48      output(out,0,"");
49    }
50
51
52    public static void generateExecuteReactionFunctions(PrintWriter out,
        RawModel model) {
```

```
53      for(Reaction reaction : model.reactions) {
54        output(out,1,"public static void __reaction_execute_" +
55            reaction.name + "() {");
56        for(ReactionElement delta : reaction.delta)
57          output(out,2,"" + delta.species + " += " + delta.coefficient + ";");
58        output(out,2,"__" + reaction.name + "_count++;");
59        for (Reaction r : reaction.affectedReactionList)
60          output(out,2,"__reaction_recalculate_propensity_" + r.name + "();");
61        output(out,2,"__trigger_check_" + reaction.name + "();");
62        output(out,2,"__magnitudeRecalcCheck();");
63        output(out,1,"}");
64        output(out,0,"");
65      }
66    }
67
68    public static void generateDMExecuteReactionFunctions(PrintWriter out,
        RawModel model) {
69      for(Reaction reaction : model.reactions) {
70        output(out,1,"public static void __reaction_execute_" +
71            reaction.name + "() {");
72        for(ReactionElement delta : reaction.delta)
73          output(out,2,"" + delta.species + " += " + delta.coefficient + ";");
74        output(out,2,"__" + reaction.name + "_count++;");
75        output(out,1,"}");
76        output(out,0,"");
77      }
78    }
79
80    public static void generateReactionPropensityFunctions(PrintWriter out,
        RawModel model) {
```

```
81      for(Reaction reaction : model.reactions) {
82        output(out,1,"public static double __reaction_calculate_propensity_" +
83            reaction.name + "() {");
84        String str = "return " + reaction.name;
85
86        for(ReactionElement reactant : reaction.reactants) {
87          for(int i=0; i<reactant.coefficient; i++) {
88            if (i == 0) {
89              str += "*" + reactant.species;
90            } else {
91              str += "*(" + reactant.species + "-" + i + ")/" + (i+1);
92            }
93          }
94        }
95        str += ";";
96        output(out,2,str);
97        output(out,1,"}");
98        output(out,0,"");
99      }
100
101     for(Reaction reaction : model.reactions) {
102       output(out,1,"public static void __reaction_recalculate_propensity_" +
103           reaction.name + "() {");
104       int index = model.reactions.indexOf(reaction);
105       output(out,2,"__propensityTotal -= __props[" + index + "];");
106       output(out,2,"__props[" + index + "] = __reaction_calculate_propensity_"
              +
107           reaction.name + "();");
108       output(out,2,"__propensityTotal += __props[" + index + "];");
109       output(out,1,"}");
```

```
110         output(out,0,"");
111       }
112
113     }
114
115     private static String getTriggerString(RawModel model,
116         ArrayList<Trigger> dependencies) {
117       String str = "";
118       for(Trigger t : model.triggers) {
119         if (dependencies.contains(t))
120           str += "1";
121         else {
122           str += "0";
123         }
124       }
125       return str;
126     }
127
128     private static int getTriggerStringIndex(ArrayList<String> triggerStrings,
129         String triggerString) {
130       int index = -1;
131       for(int i=0; i<triggerStrings.size(); i++) {
132         if (triggerString.compareTo(triggerStrings.get(i)) == 0) {
133           index = i;
134           break;
135         }
136       }
137       if (index == -1) {
138         index = triggerStrings.size();
139         triggerStrings.add(triggerString);
```

```
140     }
141     return index;
142   }
143
144   private static void generateTriggerCheckFunction(PrintWriter out,
145       RawModel model,
146       ArrayList<Trigger> dependencies,
147       ArrayList<String> triggerStrings,
148       String name) {
149     String triggerString = getTriggerString(model,dependencies);
150     int triggerStringIndex = getTriggerStringIndex(triggerStrings,
          triggerString);
151
152     output(out,1,"public static void __trigger_check_" + name + "() {");
153     output(out,2,"__trigger_madness_" + triggerStringIndex + "();");
154     output(out,1,"}");
155     output(out,0,"");
156   }
157
158   public static void generateTriggerFunctions(PrintWriter out, RawModel model)
        {
159     ArrayList<String> triggerStrings = new ArrayList<String>();
160     int dependenceCount = 0;
161
162     //Generate the all trigger string
163     String allTriggerString = "";
164     for(Trigger r : model.triggers)
165       allTriggerString += "1";
166     getTriggerStringIndex(triggerStrings,allTriggerString);
167
```

```
168
169     for(Reaction r : model.reactions)
170       generateTriggerCheckFunction(out, model, r.affectedTriggerList,
171           triggerStrings, r.name);
172     for(Event r : model.events)
173       generateTriggerCheckFunction(out, model, r.affectedTriggerList,
174           triggerStrings, r.name);
175     for(Periodic r : model.periodics)
176       generateTriggerCheckFunction(out, model, r.affectedTriggerList,
177           triggerStrings, r.name);
178     for(Trigger r : model.triggers)
179       generateTriggerCheckFunction(out, model, r.affectedTriggerList,
180           triggerStrings, r.name);
181
182     //After this loop, triggerStrings has been populated with all possible
          dependencies based on
183     //the affectedTriggerList of each collection.
184
185     //Code that limits the depth of these functions in the call stack with
          need to be added here
186     //as well as above in the generateTriggerCheckFunction method.
187
188     for(int l = 0; l < triggerStrings.size(); l++){ //Fixes exception, logic
          error?
189 //    for(String triggerString : triggerStrings) {
190
191       String triggerString = triggerStrings.get(l);
192
193       int triggerStringIndex = triggerStrings.indexOf(triggerString);
194       char triggerStringArray[] = triggerString.toCharArray();
```

```
195
196 //      System.out.println("TS = " + triggerString + "(" + triggerStringIndex
        +")");
197
198       output(out,1,"public static void __trigger_madness_" +
              triggerStringIndex + "() {");
199       for(int i=0; i<triggerString.length(); i++) {
200         if (triggerString.charAt(i) == '1') {
201
202           dependenceCount++;
203
204           //Insert depth checking code
205           //And/or interconnected-ness checking
206
207           output(out,2,"if (__trigger_condition_" + model.triggers.get(i).name
                  + "()) {");
208           output(out,3,"__trigger_action_" + model.triggers.get(i).name + "();
                 ");
209           String temp = getTriggerString(model,model.triggers.get(i).
                 affectedTriggerList);
210           char tempArray[] = temp.toCharArray();
211           for(int j=0; j<triggerString.length(); j++) {
212             if (triggerStringArray[j] == '1')
213               tempArray[j] = '1';
214           }
215           temp = new String(tempArray);
216           //This line may modify triggerStrings
217           //causes ConcurrentModificationException under new for loop notation
                 .
218           int tempIndex = getTriggerStringIndex(triggerStrings,temp);
```

```
219             //
220             output(out,3,"__trigger_madness_" + tempIndex + "();");
221             output(out,3,"return;");
222             output(out,2,"}");
223             triggerStringArray[i] = '0';
224           }
225         }
226         output(out,1,"}");
227         output(out,0,"");
228       }
229
230       //This verifies that dependency has nothing to do with stack overflow
231       System.out.println("Dependency count = " + dependenceCount);
232       System.out.println("Trigger count = " + triggerStrings.size());
233
234       for(Trigger t : model.triggers) {
235         output(out,1,"public static boolean __trigger_condition_" + t.name + "()
              {");
236         output(out,2,"return " + t.name + " && (" + t.condition + ");");
237         output(out,1,"}");
238         output(out,0,"");
239       }
240
241       for(Trigger t : model.triggers) {
242         output(out,1,"public static void __trigger_action_" + t.name + "() {");
243         output(out,2,"" + t.action);
244         output(out,2,"__" + t.name + "_count++;");
245         for (Reaction r : t.affectedReactionList)
246           output(out,2,"__reaction_recalculate_propensity_" + r.name + "();");
247         output(out,2,"__magnitudeRecalcCheck();");
```

```
248         output(out,1,"}");
249         output(out,0,"");
250       }
251     }
252
253     public static void generateCalculateAllReactionPropensitiesFunction(
          PrintWriter out, RawModel model) {
254
255       output(out,1,"public static void __calculate_all_reaction_propensities() {
            ");
256
257       output(out,2,"__propensityTotal = 0;");
258       for(int i=0; i<model.reactions.size(); i++) {
259         output(out,2,"__props[" + i + "] = " +
260             "__reaction_calculate_propensity_" + model.reactions.get(i).name + "
                ();");
261         output(out,2,"__propensityTotal += __props[" + i + "];");
262       }
263
264       output(out,2,"__oldPropensityTotal = __propensityTotal;");
265       output(out,1,"}");
266       output(out,0,"");
267     }
268
269
270     public static void generateConstantDeclarations(PrintWriter out, RawModel
          model) {
271       output(out,1,"// Constants");
272       for(Constant c : model.constants)
273         output(out,1,"static double " + c.name + " = " + c.value + ";");
```

```
274     output(out,0,"");
275   }
276
277   public static void generateSpeciesDeclarations(PrintWriter out, RawModel
          model) {
278     output(out,1,"// Species");
279     for(Species sp : model.species)
280       output(out,1,"static double " + sp.name + " = " + sp.population + ";");
281     output(out,0,"");
282   }
283
284   public static void generateRateConstantDeclarations(PrintWriter out,
          RawModel model) {
285     output(out,1,"//Rate constant declarations");
286     for(Reaction r : model.reactions)
287       output(out,1,"static double " + r.name + " = "  + r.rate + ";");
288     output(out,0,"");
289   }
290
291   public static void generateCountDeclarations(PrintWriter out, RawModel model
          ) {
292     output(out,1,"// Count declarations");
293     for(Reaction r : model.reactions)
294       output(out,1,"static int __" + r.name + "_count = "  + "0;");
295     for(Event e : model.events)
296       output(out,1,"static int __" + e.name + "_count = "  + "0;");
297     for(Periodic p : model.periodics)
298       output(out,1,"static int __" + p.name + "_count = "  + "0;");
299     for(Trigger t : model.triggers)
300       output(out,1,"static int __" + t.name + "_count = "  + "0;");
```

```
301     output(out,0,"");
302   }
303
304   public static void generateEnableDeclarations(PrintWriter out, RawModel
          model) {
305     output(out,1,"// Enable declarations");
306     for(Event e : model.events)
307       output(out,1,"static boolean " + e.name + " = true;");
308     for(Periodic p : model.periodics)
309       output(out,1,"static boolean " + p.name + " = true;");
310     for(Trigger t : model.triggers)
311       output(out,1,"static boolean " + t.name + " = true;");
312     output(out,0,"");
313   }
314
315   public static void generateOutputFunction(PrintWriter out, RawModel model) {
316     output(out,1,"public static void __output(double time) {");
317     output(out,2,"__out.print(time);");
318     for (Species s : model.species)
319       output(out,2,"__out.print(\",\"+" + s.name + ");");
320     for (Reaction r : model.reactions)
321       output(out,2,"__out.print(\",\"+__" + r.name + "_count);");
322     for (Event e : model.events)
323       output(out,2,"__out.print(\",\"+__" + e.name + "_count);");
324     for (Periodic p : model.periodics)
325       output(out,2,"__out.print(\",\"+__" + p.name + "_count);");
326     for (Trigger t : model.triggers)
327       output(out,2,"__out.print(\",\"+__" + t.name + "_count);");
328     output(out,2,"__out.println();");
329     output(out,1,"}");
```

```
330     output(out,1,"");
331   }
332
333   public static void generateOutputHeaderFunction(PrintWriter out, RawModel
          model) {
334     output(out,1,"public static void __output_header() {");
335     output(out,2,"__out.print(\"time\");");
336     for (Species s : model.species)
337       output(out,2,"__out.print(\"," + s.name + "\");");
338     for (Reaction r : model.reactions)
339       output(out,2,"__out.print(\"," + r.name + "\");");
340     for (Event e : model.events)
341       output(out,2,"__out.print(\"," + e.name + "\");");
342     for (Periodic p : model.periodics)
343       output(out,2,"__out.print(\"," + p.name + "\");");
344     for (Trigger t : model.triggers)
345       output(out,2,"__out.print(\"," + t.name + "\");");
346     output(out,2,"__out.println();");
347     output(out,1,"}");
348     output(out,1,"");
349   }
350
351   public static void generateMainStart(PrintWriter out) {
352     output(out,1,"public static void main(String args[]) throws Exception {");
353     output(out,0,"");
354   }
355
356   public static void generateMainInitPrintWriter(PrintWriter out, String dir,
          RawModel model) {
357     output(out,2,"__out = new PrintWriter(new BufferedWriter(new FileWriter(\"
```

```
        "
358          + dir + model.filename + ".csv\")));");
359      output(out,0,"");
360    }
361
362    public static void generateMainOutputHeader(PrintWriter out) {
363      output(out,2,"__output_header();");
364      output(out,0,"");
365    }
366
367    public static void generateMainInitReactionSearchOrder(PrintWriter out,
         RawModel model) {
368      output(out,2,"for(int __j=0; __j<" + model.reactions.size() + "; __j++)");
369      output(out,3,"__reactionSearchOrder[__j] = __j;");
370      output(out,0,"");
371    }
372
373    public static void generateMainCalculateReactionPropensities(PrintWriter out
         ) {
374      output(out,2,"__calculate_all_reaction_propensities();");
375      output(out,0,"");
376    }
377
378    public static void generateIllgalStateCheckFunction(PrintWriter out,
         RawModel model){
379      output(out,1,"public static void __illegalStateCheck() {");
380
381      for(Species sp : model.species){
382        output(out,2,"if (" + sp.name + "< 0) {");
383        output(out,3,"System.out.print(\"ILLEGAL STATE @ \");");
```

```
384         output(out,3,"System.out.print(\"\" + __currentTime + \" : \");");
385         output(out,3,"System.out.println(\"" + sp.name + " has negative
                population.\");");
386         output(out,3,"__out.close();");
387         output(out,3,"System.exit(1);");
388         output(out,2,"}");
389       }
390       for(Reaction r : model.reactions){
391         output(out,2,"if (" + r.name + "< 0) {");
392         output(out,3,"System.out.print(\"ILLEGAL STATE @ \");");
393         output(out,3,"System.out.print(\"\" + __currentTime + \" : \");");
394         output(out,3,"System.out.println(\"" + r.name + " has negative rate
                constant.\");");
395         output(out,3,"__out.close();");
396         output(out,3,"System.exit(1);");
397         output(out,2,"}");
398       }
399       output(out,1,"}");
400       output(out,1,"");
401     }
402
403     public static void generateMagnitudeRecalcCheckFunction(PrintWriter out,
          RawModel model) {
404       output(out,1,"static void __magnitudeRecalcCheck(){");
405       output(out,2,"if(__propensityTotal < (__oldPropensityTotal / 2) ||");
406       output(out,2,"    __propensityTotal > (2 * __oldPropensityTotal)) {");
407       output(out,3,"__calculate_all_reaction_propensities();");
408       output(out,2,"}");
409       output(out,1,"}");
410       output(out,0,"");
```

```
411   }
412
413   public static void generateExecuteEventFunctions(PrintWriter out, RawModel
          model) {
414     for(Event e : model.events) {
415       output(out,1,"static void __event_execute_" + e.name + "() {");
416       output(out,2,"if (!" + e.name + ") return;");
417       output(out,2,e.action);
418       output(out,2,"__" + e.name + "_count++;");
419       for (Reaction r : e.affectedReactionList)
420         output(out,2,"__reaction_recalculate_propensity_" + r.name + "();");
421       output(out,2,"__trigger_check_" + e.name + "();");
422       output(out,2,"__magnitudeRecalcCheck();");
423       output(out,1,"}");
424       output(out,0,"");
425     }
426
427     for(Periodic e : model.periodics) {
428       output(out,1,"static void __event_execute_" + e.name + "() {");
429       output(out,2,"if (!" + e.name + ") return;");
430       output(out,2,e.action);
431       output(out,2,"__" + e.name + "_count++;");
432       for (Reaction r : e.affectedReactionList)
433         output(out,2,"__reaction_recalculate_propensity_" + r.name + "();");
434       output(out,2,"__trigger_check_" + e.name + "();");
435       output(out,2,"__magnitudeRecalcCheck();");
436       output(out,1,"}");
437       output(out,0,"");
438     }
439   }
```

```
440
441   public static void generateDMExecuteEventFunctions(PrintWriter out, RawModel
          model) {
442     for(Event e : model.events) {
443       output(out,1,"static void __event_execute_" + e.name + "() {");
444       output(out,2,"if (!" + e.name + ") return;");
445       output(out,2,e.action);
446       output(out,2,"__" + e.name + "_count++;");
447       for (Reaction r : e.affectedReactionList)
448         output(out,2,"__reaction_recalculate_propensity_" + r.name + "();");
449       output(out,1,"}");
450       output(out,0,"");
451     }
452
453     for(Periodic e : model.periodics) {
454       output(out,1,"static void __event_execute_" + e.name + "() {");
455       output(out,2,"if (!" + e.name + ") return;");
456       output(out,2,e.action);
457       output(out,2,"__" + e.name + "_count++;");
458       for (Reaction r : e.affectedReactionList)
459         output(out,2,"__reaction_recalculate_propensity_" + r.name + "();");
460       output(out,1,"}");
461       output(out,0,"");
462     }
463   }
464
465   public static void generateExecuteNextEventFunction(PrintWriter out,
          RawModel model) {
466     ArrayList<Event> sortedEventList = new ArrayList<Event>();
467
```

```
468     for(Event e : model.events)
469       sortedEventList.add(e);
470
471     for(Periodic p : model.periodics) {
472       double time = Double.parseDouble(p.start);
473       double end = Double.parseDouble(p.end);
474
475       while (time <= end){
476         Event e = new Event();
477         e.name = p.name;
478         e.time = Double.toString(time);
479         e.action = p.action;
480         e.comment = p.comment;
481         sortedEventList.add(e);
482         time += Double.parseDouble(p.period);
483       }
484     }
485
486     Collections.sort(sortedEventList, new EventSort());
487
488     if (sortedEventList.size() == 0) {
489       output(out,1,"static double __nextEventTime = Double.POSITIVE_INFINITY;"
              );
490     } else {
491       output(out,1,"static double __nextEventTime = " + sortedEventList.get(0)
              .time + ";");
492     }
493
494     output(out,1,"public static void __executeNextEvent() {");
495     output(out,2,"switch(__nextEventIndex) {");
```

```
496
497     for(int i=0; i<sortedEventList.size(); i++) {
498       output(out,3,"case " + i + ":");
499       output(out,4,"if (" + sortedEventList.get(i).name + "){");
500       output(out,5,"__event_execute_" + sortedEventList.get(i).name + "();");
501       output(out,4,"}");
502       if (i+1 < sortedEventList.size()) {
503         output(out,4,"__nextEventTime = " + sortedEventList.get(i+1).time + ";
              ");
504       } else {
505         output(out,4,"__nextEventTime = Double.POSITIVE_INFINITY;");
506       }
507       output(out,4,"break;");
508     }
509     output(out,2,"}");
510     output(out,2,"__nextEventIndex++;");
511     output(out,1,"}");
512     output(out,0,"");
513   }
514
515   public static void generateExecuteNextReactionFunction(PrintWriter out,
        RawModel model){
516     output(out,1,"static int __selectReactionToExecute() {");
517     output(out,2,"double __selector;");
518     output(out,2,"int __reactionIndex;");
519     output(out,2,"while (true){");
520     output(out,3,"__selector = __generator.nextDouble() * __propensityTotal;")
          ;
521     output(out,3,"for(int __i=0;__i<__props.length;__i++){");
522     output(out,4,"__reactionIndex = __reactionSearchOrder[__i];");
```

```
523     output(out,4,"__selector -= __props[__reactionIndex];");
524     output(out,4,"if (__selector <= 0) return __i;");
525     output(out,3,"}");
526     output(out,2,"}");
527     output(out,1,"}");
528     output(out,0,"");
529
530
531     output(out,1,"static void __executeNextReaction(){");
532     output(out,2,"int __i = __selectReactionToExecute();");
533     output(out,2,"switch(__reactionSearchOrder[__i]){");
534     for(int i=0;i<model.reactions.size();i++){
535       output(out,3,"case " + i + ":");
536       output(out,4,"  __reaction_execute_" + model.reactions.get(i).name + "()
              ;");
537       output(out,4,"break;");
538     }
539     output(out,2,"}");
540     output(out,0,"");
541
542     //Bubble up executed reaction
543     output(out,2,"if (__i != 0){");
544     output(out,3,"__lastExecutedReactionIndex = __reactionSearchOrder[__i];");
545     output(out,3,"__reactionSearchOrder[__i] = __reactionSearchOrder[__i-1];")
          ;
546     output(out,3,"__reactionSearchOrder[__i-1] = __lastExecutedReactionIndex;"
          );
547     output(out,2,"} else {");
548     output(out,3,"__lastExecutedReactionIndex = 0;}");
549     output(out,1,"}");
```

```
550     output(out,0,"");
551   }
552
553   public static void generateDMExecuteNextReactionFunction(PrintWriter out,
        RawModel model){
554     output(out,1,"static int __selectReactionToExecute() {");
555     output(out,2,"double __selector;");
556     output(out,2,"int __reactionIndex;");
557     output(out,2,"while (true){");
558     output(out,3,"__selector = __generator.nextDouble() * __propensityTotal;")
          ;
559     output(out,3,"for(int __i=0;__i<__props.length;__i++){");
560     output(out,4,"__selector -= __props[__i];");
561     output(out,4,"if (__selector <= 0) return __i;");
562     output(out,3,"}");
563     output(out,2,"}");
564     output(out,1,"}");
565     output(out,0,"");
566
567
568     output(out,1,"static void __executeNextReaction(){");
569     output(out,2,"int __i = __selectReactionToExecute();");
570     output(out,2,"switch(__i){");
571     for(int i=0;i<model.reactions.size();i++){
572       output(out,3,"case " + i + ":");
573       output(out,4,"  __reaction_execute_" + model.reactions.get(i).name + "()
            ;");
574       output(out,4,"break;");
575     }
576     output(out,2,"}");
```

```
577     output(out,0,"");
578
579     output(out,1,"}");
580     output(out,0,"");
581   }
582
583   // TODO: Make all generated files into parameters
584
585   public static void generateOutputFile(RawModel model){
586
587
588     //Calls to finish setting up the model
589     DependencyGen.calculateReactionDeltas(model);
590     DependencyGen.calculateModelDependencies(model);
591
592     String dir = "proc/";
593
594     try {
595       PrintWriter procClass = new PrintWriter(new BufferedWriter(new
              FileWriter(dir + model.filename + ".java")));
596       generateHeader(procClass);
597       output(procClass,0,"public class " + model.filename + " {");
598
599       generateGlobalSimVarDeclarations(procClass, model,dir);
600       generateConstantDeclarations(procClass,model);
601       generateSpeciesDeclarations(procClass,model);
602       generateRateConstantDeclarations(procClass,model);
603       generateCountDeclarations(procClass,model);
604       generateEnableDeclarations(procClass,model);
605
```

```
606         generateOutputHeaderFunction(procClass,model);
607         generateOutputFunction(procClass,model);
608         generateIllgalStateCheckFunction(procClass,model);
609         generateReactionPropensityFunctions(procClass,model);
610         generateCalculateAllReactionPropensitiesFunction(procClass,model);
611         generateMagnitudeRecalcCheckFunction(procClass,model);
612         generateExecuteEventFunctions(procClass,model);
613         generateExecuteReactionFunctions(procClass,model);
614         generateExecuteNextEventFunction(procClass,model);
615         generateExecuteNextReactionFunction(procClass,model);
616         generateTriggerFunctions(procClass,model);
617
618         generateMainStart(procClass);
619         generateMainInitPrintWriter(procClass, dir, model);
620         generateMainOutputHeader(procClass);
621         generateMainInitReactionSearchOrder(procClass,model);
622         generateMainCalculateReactionPropensities(procClass);
623
624         output(procClass,2,"// Check all triggers at startup.");
625         output(procClass,2,"__trigger_madness_0();");
626
627         output(procClass,2,"while(true) {");
628
629         output(procClass,3,"__nextReactionTime = __currentTime-(Math.log(
                __generator.nextDouble()))/__propensityTotal;");
630         output(procClass,0,"");
631
632         output(procClass,3,"if (__nextEventTime < __nextReactionTime) {");
633         output(procClass,4,"__currentTime = __nextEventTime;");
634         output(procClass,3,"} else {");
```

```
635         output(procClass,4,"__currentTime = __nextReactionTime;");
636         output(procClass,3,"}");
637         output(procClass,0,"");
638
639         output(procClass,3,"while (__currentTime > __nextPrintTime) {");
640         output(procClass,4,"__output(__nextPrintTime);");
641         output(procClass,4,"__nextPrintTime += " + model.interval + ";");
642         output(procClass,4,"if (__nextPrintTime >= " + model.end + ") {");
643         output(procClass,5,"__out.close();");
644         output(procClass,5,"System.exit(0);");
645         output(procClass,4,"}");
646         output(procClass,3,"}");
647         output(procClass,0,"");
648
649         output(procClass,3,"if (__currentTime == __nextEventTime) {");
650         output(procClass,4,"__executeNextEvent();");
651         output(procClass,3,"} else {");
652         output(procClass,4,"__executeNextReaction();");
653         output(procClass,3,"}");
654         output(procClass,0,"");
655
656         output(procClass,3,"__illegalStateCheck();");
657         output(procClass,2,"}");
658         output(procClass,0,"");
659
660         output(procClass,1,"}");
661         output(procClass,0,"}");
662
663         procClass.close();
664
```

```
665
666     } catch (IOException e) {
667       e.printStackTrace();
668       System.exit(1);
669     }
670   }
671
672   public static void generateDirectMethodOutputFile(RawModel model){
673
674
675     //Calls to finish setting up the model
676     DependencyGen.calculateReactionDeltas(model);
677 //     DependencyGen.calculateModelDependencies(model);
678
679     String dir = "proc/";
680
681     try {
682       PrintWriter procClass = new PrintWriter(new BufferedWriter(new
              FileWriter(dir + model.filename + ".java")));
683       generateHeader(procClass);
684       output(procClass,0,"public class " + model.filename + " {");
685
686       generateGlobalSimVarDeclarations(procClass, model,dir);
687       generateConstantDeclarations(procClass,model);
688       generateSpeciesDeclarations(procClass,model);
689       generateRateConstantDeclarations(procClass,model);
690       generateCountDeclarations(procClass,model);
691       generateEnableDeclarations(procClass,model);
692
693       generateOutputHeaderFunction(procClass,model);
```

```
694         generateOutputFunction(procClass,model);
695         generateIllgalStateCheckFunction(procClass,model);
696
697         generateReactionPropensityFunctions(procClass,model);
698         generateCalculateAllReactionPropensitiesFunction(procClass,model);
699 /*          generateMagnitudeRecalcCheckFunction(procClass,model); */
700         generateDMExecuteEventFunctions(procClass,model);
701         generateDMExecuteReactionFunctions(procClass,model);
702         generateExecuteNextEventFunction(procClass,model);
703         generateDMExecuteNextReactionFunction(procClass,model);
704 /*          generateTriggerFunctions(procClass,model);*/
705
706         generateMainStart(procClass);
707         generateMainInitPrintWriter(procClass, dir, model);
708         generateMainOutputHeader(procClass);
709
710
711         output(procClass,2,"while(true) {");
712
713         // Check all triggers until they are all satisfied.
714
715         output(procClass,3,"boolean triggerMadnessDone = false;");
716         output(procClass,3,"while(!triggerMadnessDone) {");
717         output(procClass,4,"triggerMadnessDone = true;");
718
719         // check each trigger condition.
720         // if true...
721         //   fire action..
722         //   increment trig counter...
723         //   triggerMadnessDone = false
```

```
724
725         for(Trigger t : model.triggers){
726           output(procClass,4,"if(" + t.condition + "){");
727           output(procClass,5,t.action);
728           output(procClass,5,"__" + t.name + "_count++;");
729           output(procClass,5,"triggerMadnessDone = false;");
730           output(procClass,4,"}");
731         }
732
733
734         output(procClass,3,"}");
735
736         output(procClass,3,"__calculate_all_reaction_propensities();");
737
738         output(procClass,3,"__nextReactionTime = __currentTime-(Math.log(
                __generator.nextDouble()))/__propensityTotal;");
739         output(procClass,0,"");
740
741         output(procClass,3,"if (__nextEventTime < __nextReactionTime) {");
742         output(procClass,4,"__currentTime = __nextEventTime;");
743         output(procClass,3,"} else {");
744         output(procClass,4,"__currentTime = __nextReactionTime;");
745         output(procClass,3,"}");
746         output(procClass,0,"");
747
748         output(procClass,3,"while (__currentTime > __nextPrintTime) {");
749 //        output(procClass,4,"__output(__nextPrintTime);");
750         output(procClass,4,"__nextPrintTime += " + model.interval + ";");
751         output(procClass,4,"if (__nextPrintTime >= " + model.end + ") {");
752         output(procClass,5,"__out.close();");
```

```
753         output(procClass,5,"System.exit(0);");
754         output(procClass,4,"}");
755         output(procClass,3,"}");
756         output(procClass,0,"");
757 
758         // Execute either the event or the reaction - WITHOUT DEPENDENDCY GRAPH
759 
760 
761         output(procClass,3,"if (__currentTime == __nextEventTime) {");
762         output(procClass,4,"__executeNextEvent();");
763         output(procClass,3,"} else {");
764         output(procClass,4,"__executeNextReaction();");
765         output(procClass,3,"}");
766         output(procClass,0,"");
767 
768 
769 
770         output(procClass,3,"__illegalStateCheck();");
771         output(procClass,2,"}");
772         output(procClass,0,"");
773 
774         output(procClass,1,"}");
775         output(procClass,0,"}");
776 
777         procClass.close();
778 
779 
780       } catch (IOException e) {
781         e.printStackTrace();
782         System.exit(1);
```

```
783       }
784     }
785
786
787 }
```

ModelCodeGen.java

A.8 TestGenerator

The following file generates test models that can be used as input to the simulator.

```
1  package edu.vcu.csl.ess;
2
3  import java.io.File;
4  import java.io.FileOutputStream;
5  import java.io.IOException;
6  import java.io.PrintStream;
7  import java.util.Random;
8  import java.util.Vector;
9
10 public class TestGenerator {
11
12   public class SpeciesPair{
13
14     private int s1;
15     private int s2;
16
17     SpeciesPair(int i, int j){
```

```
18        s1 = i;s2 = j;
19      }
20
21      public int getS1(){return s1;}
22      public int getS2(){return s2;}
23
24    }
25
26    private int species_count;
27    private int reaction_count;
28    private int random_reaction_count;
29    private int event_count;
30    private int periodics_count;
31    private int trigger_count;
32
33    private double start_time;
34    private double end_time;
35
36    private Random generator;
37    private Vector<SpeciesPair> randomPairs;
38
39    public TestGenerator(int speciesCount,int reactionCount,int
          randomReactionCount,
40        int eventCount,int periodicsCount,int triggerCount,double start, double
              end){
41
42      species_count = speciesCount;
43      reaction_count = reactionCount;
44      random_reaction_count = randomReactionCount;
45      event_count = eventCount;
```

```
46      periodics_count = periodicsCount;
47      trigger_count = triggerCount;
48
49      start_time = start;
50      end_time = end;
51
52      randomPairs = new Vector<SpeciesPair>();
53
54  //      generator = new Random(1);    //Works
55  //      generator = new Random(2);
56  //      generator = new Random(8);
57
58      generator = new Random(777); //Broken
59  //      generator = new Random(3);
60  //      generator = new Random(4);
61  //      generator = new Random(5);
62  //      generator = new Random(6);
63  //      generator = new Random(7);
64  //      generator = new Random(9);
65
66    }
67
68    public void write(String fileName) throws IOException  {
69
70      //Generate random pairs
71
72      int rand1;
73      int rand2;
74
75      for(int i=0;i<random_reaction_count;i++){
```

```
76
77          rand1 = 0;
78          rand2 = 0;
79
80          boolean okPair = false;
81          SpeciesPair sp = null;
82
83          while (!okPair){
84            rand1 = (int)(generator.nextDouble()*species_count);
85            rand2 = (int)(generator.nextDouble()*species_count);
86
87            sp = new SpeciesPair(rand1,rand2);
88
89            okPair = true;
90            for (SpeciesPair s : randomPairs){
91              if(s.s1 == rand1 && s.s2 == rand2){
92                okPair = false;
93              }
94            }
95
96          }
97
98          randomPairs.add(sp);
99
100       }
101
102       File f = new File(fileName);
103       PrintStream out = new PrintStream(new FileOutputStream(f));
104
105       out.println("-- RawModel FILE v1.1 --");
```

```
106     //Constants count
107     //  Name
108     //  Value
109     //  Comment
110     out.println("0"); //We won't be using constants
111
112
113     //Species count(s)
114     //  Species name
115     //  Initial pop.
116     //  Comment
117
118 //     out.println(species_count*2 + (int)Math.pow(species_count,2));
119     out.println(species_count*2 + randomPairs.size());
120
121     for(int i=0;i<species_count;i++){
122
123       out.println("s" + i); //Name
124       out.println("0"); //Intial pop.
125       out.println(""); //Comment
126
127       out.println("g" + i); //Name
128       out.println("1"); //Intial pop.
129       out.println(""); //Comment
130
131     }
132
133 //     for(int i=0;i<species_count;i++){
134     for(SpeciesPair sp : randomPairs){
135       out.println("g" + sp.s1 + "s" + sp.s2);
```

```
136         out.println("0"); //Intial pop.
137         out.println(""); //Comment
138       }
139
140       //Reaction count(s)
141       //  Name
142       //  Equation
143       //  Propensity
144       //  Comment
145
146       //Set 1
147       //Every reaction moves the corresponding species forward (w/ wraparound)
148
149       out.println(reaction_count*2 + random_reaction_count*2);
150       for(int i=0;i<reaction_count;i++){
151
152         out.println("r" + i); //Name
153 //        out.println("s" + i + " => " + "s" + ((i+1)%species_count)); //
      Equation
154         out.println("g" + i + " => g" + i + " + s" + i);
155         out.println(10 + generator.nextDouble()*90); //Rate constant
156         out.println("Set 1"); //Comment
157
158         out.println("rd" +i); //Name
159         out.println("s" + i + " => *"); //Equation
160         out.println(0.5 + generator.nextDouble()*1); //Rate constant
161         out.println("Set 1 (Death)"); //Comment
162       }
163
164       //Set 2
```

```
165     //Reactions shift random species around
166
167 //     for(int i=0;i<random_reaction_count;i++){
168     for(SpeciesPair sp : randomPairs){
169
170       out.println("rand" + randomPairs.indexOf(sp));  //Name (+ reactions
              already listed)
171       out.println("g" + sp.getS1() + " + s" + sp.getS2()  +" => g" + sp.getS1
              () + "s" + sp.getS2());  //Equation
172       out.println((0.1+ generator.nextDouble()*1)/10);  //Rate constant
173       out.println("Set 2"); //Comment
174
175       out.println("randback" + randomPairs.indexOf(sp));  //Name (+ reactions
              already listed)
176       out.println("g" + sp.getS1() + "s" + sp.getS2()  + " => g" + sp.getS1()
             + " + s" + sp.getS2()); //Equation
177       out.println((0.1+ generator.nextDouble()*1)*10);  //Rate constant
178       out.println("Set 2 (Back)");  //Comment
179
180     }
181
182     //Event count(s)
183     //  Name
184     //  Time
185     //  Action
186     //  Comment
187
188     int rand3;
189
190     out.println(event_count);
```

```
191     for(int i=0;i<event_count;i++){
192
193       rand1 = 0;
194       rand2 = 0;
195       rand3 = 0;
196
197       while ((rand1 == rand2) || (rand2 == rand3) || (rand1 == rand3)){
198         rand1 = (int)(generator.nextDouble()*species_count);
199         rand2 = (int)(generator.nextDouble()*species_count);
200         rand3 = (int)(generator.nextDouble()*species_count);
201       }
202
203       out.println("e" + i); //Name
204       out.println((generator.nextDouble() * (end_time - start_time)) +
              start_time);
205       out.println("s" + rand2 + " += s" + rand1 + " / 2;" +
206           "s" + rand3 + " += s" + rand1 + " - (s" + rand1 + " / 2);" +
207           "s" + rand1 + " = 0;"); //Intial pop.
208       out.println(""); //Comment
209     }
210
211     //Periodics count(s)
212     //  Name
213     //  Start
214     //  Period
215     //  End
216     //  Action
217     //  Comment
218
219     double periodic_start_time;
```

```
220     double periodic_end_time;
221     int desired_executions = 10;
222
223 //     out.println(periodics_count);
224 //     for(int i=0;i<periodics_count;i++){
225 //
226 //         periodic_start_time = (generator.nextDouble() * (end_time - start_time
    )) + start_time;
227 //         periodic_end_time = (generator.nextDouble() * (end_time -
    periodic_start_time)) + periodic_start_time;
228 //
229 //         out.println("p" + i); //Name
230 //         out.println(periodic_start_time); //Start
231 //         out.println((periodic_end_time - periodic_start_time) /
    desired_executions);  //Period
232 //         out.println(periodic_end_time); //End
233 //         out.println("s" + (int)(generator.nextDouble()*species_count) + " =
    0");  //Action
234 //         out.println("");  //Comment
235 //     }
236
237     //Binomial cell division (all species)
238 //     out.println(species_count);
239 //     for(int i=0; i < species_count; i++){
240 //         out.println("p" + i); //Name
241 //         out.println(start_time);  //Start
242 //         out.println(1000);  //Period
243 //         out.println(end_time);  //End
244 //         out.println("s" + i + " = (int)(s" +i + " / 2);");  //Action
245 //         out.println("");  //Comment
```

```
246 //     }
247
248     //Random # of periodic cell divisions
249     rand1 = (int)(generator.nextDouble()*species_count);
250
251     out.println(periodics_count);
252     for(int i=0; i < periodics_count; i++){
253       out.println("p" + i); //Name
254       out.println(start_time);  //Start
255       out.println((int)((end_time - start_time)/100));  //Period
256       out.println(end_time);  //End
257       out.println("s" + rand1 + " = (int)(s" + rand1 + " / 2);"); //Action
258       out.println("");  //Comment
259     }
260
261     //Trigger count(s)
262     //  Name
263     //  Condition
264     //  Action
265     //  Comment
266
267     int species_number;
268
269     out.println(trigger_count);
270     for(int i=0;i<trigger_count;i++){
271
272       rand1 = 0;
273       rand2 = 0;
274
275       while (rand1 == rand2){
```

```
276             rand1 = (int)(generator.nextDouble()*species_count);
277             rand2 = (int)(generator.nextDouble()*species_count);
278         }
279 
280         out.println("t" + i); //Name
281 //          out.println("s" + rand1 + " > " + (20 + generator.nextDouble()*20));
        //Condition
282         out.println("s" + rand1 + " > " + (1 + generator.nextDouble()*1));  //
              Condition
283         out.println("s" + rand2 + " += (int)(s" + rand1 + " / 2);" +
284             "s" + rand1 + " = 0;"); //Action
285         out.println("");  //Comment
286     }
287 
288     //Sim Options
289     //  Start
290     //  End
291     //  Interval
292     //  Seed
293     //  Sim file name
294 
295     int desired_data_points = 1000;
296 
297     out.println(start_time);
298     out.println(end_time);
299     out.println((end_time - start_time) / desired_data_points);
300     out.println(777);
301     out.println("Auto");
302 
303     out.close();
```

```
304     }
305
306     /**
307      * @param args
308      */
309     public static void main(String[] args) {
310
311 //      int species    = 3;
312 //      int reactions    = 3;
313 //      int randoms    = 3;
314 //      int events     = 0;
315 //      int periodics    = 0;
316 //      int triggers  = 0;
317 //
318 //      double start  = 0.0;
319 //      double end    = 1000.0;
320
321 //Torture test block:
322 //      int species    = 100;
323 //      int reactions    = 100;
324 //      int randoms    = 10;
325 //      int events     = 0;
326 //      int periodics    = 1;
327 ////      int periodics    = 0;
328 //      int triggers  = 50;
329 ////      int triggers  = 0;
330 //
331 //      double start  = 0.0;
332 //      double end    = 10000.0;
333
```

```
334     //New test block
335     int species   = 50;
336     int reactions   = 50;
337     int randoms   = 0;
338     int events    = 0;
339     int periodics   = 50;
340 //    int periodics   = 0;
341     int triggers  = 50;
342 //    int triggers  = 0;
343
344     double start  = 0.0;
345     double end    = 10000.0;
346
347 //    System.out.println((new Random()).nextDouble());
348
349     //TestGenerator tg = new TestGenerator(3,2,3,0,0,0.0,100);
350     TestGenerator tg = new TestGenerator(species,reactions,randoms,events,
            periodics,triggers,start,end);
351     try {
352       tg.write("test.abc");
353       System.out.println("Test file successfully generated! :)");
354     } catch (IOException e) {
355       // TODO Auto-generated catch block
356       e.printStackTrace();
357     }
358
359   }
360
361 }
```

A.9 Single Execution Script (C)

The following file compiles the current Java code and launches a simulation based on the argument passed to the script. It executes a single simulation of the EEHDM (in C) and also generates statistics (if necessary). _c

```
1  _c
2  #!/bin/bash
3
4  rm proc/* tmp/brad.jpg
5  javac -d bin src/edu/vcu/csl/ess/*.java
6  java -Xms128m -Xmx1024m -cp bin edu.vcu.csl.ess.RawModel $1
7  javac -g:none -O proc/*.java
8  time java proc.Auto
9  #java proc.Raw
10 cd scripts
11 ./pic ../proc/Auto.csv ../tmp/brad.jpg
12 #./pic ../proc/Raw.csv ../tmp/brad.jpg
13 eog ../tmp/brad.jpg
14 cd .._c
```

go_c

A.10 Single Execution Script (Java)

The following file compiles the current Java code and launches a simulation based on the arguement passed to the script. It executes a single simulation of the EEHDM (in Java) and also generates statistics (if necessary).

```
1  #!/bin/bash
2
3  rm proc/* tmp/brad.jpg
4  javac -d bin src/edu/vcu/csl/ess/*.java
5  java -Xms128m -Xmx1024m -cp bin edu.vcu.csl.ess.RawModel $1
6  javac -g:none -O proc/*.java
7  time java proc.Auto
8  #java proc.Raw
9  cd scripts
10 ./pic ../proc/Auto.csv ../tmp/brad.jpg
11 #./pic ../proc/Raw.csv ../tmp/brad.jpg
12 eog ../tmp/brad.jpg
13 cd ..
```

go

A.11 Mass Execution Script (C)

The following file compiles the current Java code and launches a simulation based on the arguement passed to the script. It executes 3 separate simulations for each ESS method (EEHDM, EHDM, and naïve DM). Timing data is output to the console for these 3 simulations.

```
1  #!/bin/bash
2
3  #Method 1
4  rm proc/* tmp/brad.jpg
5  javac -d bin src/edu/vcu/csl/ess/*.java
6  java -Xms128m -Xmx1024m -cp bin edu.vcu.csl.ess.RawModel $1 1
```

```
7  cd proc
8  echo -e "Compiling...\n"
9  time gcc -O3 -lm Auto.c 2> compile_test
10 cat compile_test
11
12 echo -e "Running...\n"
13 time ./a.out
14 time ./a.out
15 time ./a.out
16
17 cd ..
18
19 #Method 1
20 rm proc/* tmp/brad.jpg
21 javac -d bin src/edu/vcu/csl/ess/*.java
22 java -Xms128m -Xmx1024m -cp bin edu.vcu.csl.ess.RawModel $1 2
23 cd proc
24 echo -e "Compiling...\n"
25 time gcc -O3 -lm Auto.c 2> compile_test
26 cat compile_test
27
28 echo -e "Running...\n"
29 time ./a.out
30 time ./a.out
31 time ./a.out
32
33 cd ..
34
35 #Method 1
36 rm proc/* tmp/brad.jpg
```

```
37 javac -d bin src/edu/vcu/csl/ess/*.java
38 java -Xms128m -Xmx1024m -cp bin edu.vcu.csl.ess.RawModel $1 3
39 cd proc
40 echo -e "Compiling...\n"
41 time gcc -O3 -lm Auto.c 2> compile_test
42 cat compile_test
43
44 echo -e "Running...\n"
45 time ./a.out
46 time ./a.out
47 time ./a.out
48
49 cd ..
50
51 #Run R Script
52 #cd ../scripts
53 ##./pic ../proc/Auto.csv ../tmp/brad.jpg
54 #./pic ../proc/Raw.csv ../tmp/brad.jpg
55 #eog ../tmp/brad.jpg
56
57 cd ..
```

mass_test