8-2006

Accelerating Exact Stochastic Simulation of Biochemical Systems

James Michael McCollum
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a dissertation written by James Michael McCollum entitled "Accelerating Exact Stochastic Simulation of Biochemical Systems." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Engineering.

Gregory D. Peterson, Chris D. Cox, Major Professor

We have read this dissertation and recommend its acceptance:

Seddik M. Djouadi, Donald W. Bouldin, Michael L. Simpson

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a dissertation written by James Michael McCollum entitled "Accelerating Exact Stochastic Simulation of Biochemical Systems." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Engineering.

Gregory D. Peterson
Major Professor

Chris D. Cox
Major Professor

We have read this dissertation
and recommend its acceptance:

Seddik M. Djouadi

Donald W. Bouldin

Michael L. Simpson

Acceptance for the Council:

Anne Mayhew
Vice Chancellor and
Dean of Graduate Studies

ACCELERATING EXACT STOCHASTIC SIMULATION OF BIOCHEMICAL SYSTEMS

A Dissertation
Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

James Michael McCollum
August 2006

DEDICATION

This dissertation is dedicated to my wife, Sarah,
for her love and support.

ACKNOWLEDGEMENTS

I wish to thank the many people who helped me complete my Doctorate in Computer Engineering. I would especially like to thank Dr. Peterson and Dr. Cox for co-chairing my committee and Dr. Bouldin, Dr. Simpson, and Dr. Djouadi for serving on my committee.

I would also like to thank the organizations and projects that funded this research, including the NSF/DARPA Bio-SPICE Project, the Keck Futures Initiative, the Department of Energy's Genomes to Life program, the University of Tennessee Center for Environmental Biotechnology, the University of Tennessee Center for Information Technology Research, and the Bodenheimer Fellowship.

Additional, I would like to thank Dr. Yang Cao, Dr. Hong Li, and Dr. Linda Petzold of the University of California Santa Barbara for providing the source code for their implementation of the Optimized Direct Method and the E. coli heat stress response model. I would also like to thank Dr. Chris Myers of the University of Utah for kindly sharing his SBML model of the phage-λ system.

ABSTRACT

The ability to accurately and efficiently simulate computer models of biochemical systems is of growing importance to the molecular biology and pharmaceutical research communities. Exact stochastic simulation is a popular approach for simulating such systems because it properly represents genetic noise and it accurately represents systems with small populations of chemical species. Unfortunately, the computational demands of exact stochastic simulation often limit its applicability. To enable next-generation whole-cell and multi-cell stochastic modeling, advanced tools and techniques must be developed to increase simulation efficiency. This work assesses the applicability of a variety of hardware and software acceleration approaches for exact stochastic simulation including serial algorithm improvements, parallel computing, reconfigurable computing, and cluster computing. Through this analysis, improved simulation techniques for biological systems are explored and evaluated.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 - Introduction

The literature review provided in this chapter is a revised and expanded version of the literature review in a paper published in the journal *Computational Biology and Chemistry* in 2006 by James McCollum, Greg Peterson, Chris Cox, Mike Simpson, and Nagiza Samatova:

McCollum JM, Peterson GD, Cox CD, Simpson ML, Samatova NF, "The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior." *Journal of Computational Biology and Chemistry* 30 (1): 39-49, 2006.

My use of "we" refers to my co-authors and myself. My primary contributions to this paper include (1) a majority of the writing, (2) the formation of the literature review, (3) the implementation and performance analysis of the algorithms, and (4) the development and analysis of the sorting direct method.

1.1. In Silico Biology

Consider the design of a large-scale digital circuit. Initially, specifications are developed to identify the inputs, outputs, function, and performance of the design in a spoken language like English. This description is then translated into a behavioral model of the circuit typically written in a high-level computer hardware description language like SystemC, Verilog, or VHDL. The behavioral model is used as an outline for the development of a high-level structural model, which is then transformed into a logical model, and eventually converted into a low-level layout model that is ready for implementation. This translation process typically uses a combination of previously constructed components (Intellectual Property), automated computational design tools, and manual design entry. At each level of abstraction, simulation tools and design automation tools are used to optimize the structure of the design and validate that the system meets or exceeds the predetermined specifications for performance and manufacturability. Only after the computational models of the system are built and validated will a prototype of the design be manufactured.

The models serve as blueprints for the design. They are a mechanism for documenting and communicating the design between engineers and are used as building blocks for future designs. The computational tools used to support modeling essentially create a virtual digital laboratory for prototyping designs. Without this virtual laboratory, today's multi-million transistor digital systems would be impractical to design due to system complexity, engineering time, and implementation cost. The infusion of computation into the digital design process has revolutionized the field, expanding the capabilities of design engineers.

A similar revolution may soon take place in the field of molecular biology. Computational tools are being developed to help manage, interpret, and model the large

data sets coming from high-throughput biological experiments. These tools can be used to search for common structure or motifs in biochemical systems (Kashtan, 2004; Yeger-Lotem, 2004), to help discover the regulatory mechanisms that control biological system behavior (Strohman, 2003), and for modeling and simulation of the biochemical system (Chickarmane, 2005; Adalsteinsson, 2004; Vass, 2004; Schmidt, 2005). Through this work, uniform languages for developing and communicating biological modeling information are being developed (Hucka, 2003; Llyod, 2004).

The goal of these tools is to provide a framework for biological experiment design, documentation, and validation. Essentially, these tools will create a virtual biological laboratory where *in silico* experimentation can complement or even replace *in vivo* and *in vitro* experimentation. Eventually, computers may not only provide prediction and validation information to the biologist, but will suggest the most relevant set of biological experiments to perform next, maximizing the productivity and efficiency of the biologist.

For this revolution to have the broadest impact, several advances in experimental measurement, computational, and modeling technologies must be achieved. This work specifically addresses the problem of providing a computational simulation platform that accurately and efficiently predicts the behavior of a biological system.

1.2. Deterministic Modeling

The most common mathematical formulation used for biochemical system modeling is a deterministic approach that represents chemical species as continuous valued concentrations. Interactions between chemical species are represented by a system of ordinary differential equations that depend on the instantaneous values of the concentrations.

For example, consider the autoregulated biochemical system given in Figure 1[1]. A gene, which is a section of the DNA that stores the genetic code or recipe for a particular protein, is copied into a molecule called messenger ribonucleic acid (mRNA) through a process called transcription. This mRNA molecule can also degrade over time through a decay process. At a given time *t*, if we let *G(t)* represent the concentration of the gene, and *M(t)* represent the concentration of the mRNA, we can write equation 1-1 to describe the instantaneous change in mRNA concentration over time.

$$\frac{dM}{dt} = k_{tc}G(t) - k_{md}M(t) \qquad (1\text{-}1)$$

In equation 1-1, mRNA molecules are introduced into the system via the transcription process, which depends on the transcription rate k_{tc} and the concentration of the gene

[1] All figures and tables are located in the Appendix.

G(t). mRNA molecules are consumed by the decay process, which depends on the mRNA decay rate k_{md} and the concentration of the mRNA *M(t)*.

In a process called translation, the genetic recipe encoded in the mRNA is decoded to produce a protein. Similar to mRNA, these proteins can degrade through a decay process. Two proteins can bind together to form a protein dimer, which can also disassociate back into two proteins. At a given time *t*, if we let *P(t)* represent the concentration of the protein, and *D(t)* represent the concentration of the dimer, we can describe the instantaneous change in protein concentration over time using equation 1-2.

$$\frac{dP}{dt} = k_{tl}M(t) - k_{pd}P(t) - 2k_{di}P(t)^2 + 2k_{rdi}D(t) \tag{1-2}$$

Introduction of proteins into the system occurs through the translation process, which depends on the translation rate k_{tl} and the current mRNA concentration *M(t)*. Proteins leave the system through decay, which depends on the protein concentration *P(t)* and the decay rate k_{pd}. The third term of equation 1-2 represents the dimerization process, where two proteins are consumed to produce a dimer. Because any two protein molecules can react together to form a dimer, this term depends on $P(t)^2$ along with the dimerization rate k_{di}. The fourth term represents the disassociation of dimer molecules back into proteins and depends on the dimer concentration *D(t)* and the reverse dimerization rate k_{rdi}.

In a process called regulation, molecules can bind to genes and induce (accelerate) or repress (halt) the transcription process of the gene. This is the mechanism by which the cells control the production of mRNA and therefore protein. In the example in Figure 1, the dimer molecule is self-regulating in that it can bind to the gene and repress the transcription process. Eventually the dimer frees itself from the gene and the production of mRNA molecules resumes. If we allow *R(t)* to represent the concentration of the repressed gene, we can write equation 1-3 to represent the instantaneous change in dimer concentration.

$$\frac{dD}{dt} = k_{di}P(t)^2 - k_{rdi}D(t) - k_{bind}D(t)G(t) + k_{unbind}R(t) \tag{1-3}$$

The first two terms of equation 1-3 are used to represent the protein dimerization process. The third term defines the rate at which the regulation process consumes dimer molecules and depends on the current gene concentration *G(t)*, the current dimer concentration *D(t)*, and the binding rate k_{bind}. Dimer molecules are reintroduced into the system when they unbind from the gene as shown in the fourth term of equation 1-3. We can now write equations 1-4 and 1-5 to represent the instantaneous change of gene and repressed gene concentrations.

$$\frac{dG}{dt} = k_{unbind}R(t) - k_{bind}D(t)G(t) \tag{1-4}$$

$$\frac{dR}{dt} = k_{bind} D(t) G(t) - k_{unbind} R(t) \tag{1-5}$$

Equations 1-1 through 1-5 and initial conditions for each of the concentrations fully describe the biochemical system given in Figure 1 using the deterministic approach. Note, this system can be simplified using the relation given in equation 1-6.

$$G(t) = G(0) + R(0) - R(t) \tag{1-6}$$

Stiff differential equation solvers like those found in ODEPack (Hindmarsh, 2001) or Matlab (Shampine, 1997) can be used to generate numerical solutions for the deterministic approach and their execution time is typically short. Matlab code for the deterministic model of this system is given in the Appendix. A plot of the numerical solution generated by Matlab is given in Figure 2.

While deterministic solutions are sufficiently accurate to capture the dynamics of systems with relatively large populations of the various chemical species, biological cells with length scales on the order of microns often contain populations of certain species that are under 100 molecules. In such cases, deterministic solutions may not sufficiently or accurately describe the evolving biochemical system.

Furthermore, stochastic fluctuations of molecular populations (noise) may result in phenotypic variation among a population of genetically identical cells exposed to uniform environmental conditions. Cells have evolved to either tolerate, or in some cases exploit, the noisy intracellular environment resulting from these stochastic fluctuations. These possibilities have stimulated a body research on the dynamics and role of noise in regulatory networks in recent years (Rao, 2002; Kærn, 2005). Proper modeling of these situations requires the use of a stochastic formulation.

Evidence of the inaccuracy of the deterministic approach can be observed in Figure 3, which shows the evolving gene concentration from our Matlab simulation of the Figure 1 model. The gene concentration starts at 1.0 molecules per cell volume and eventually reaches a steady state value just under 0.40 molecules per cell volume. The result of 0.40 molecules per cell volume has little physical meaning within the context of the biological system, because the gene in the underlying physical system can only have one of two discrete states: repressed or expressed. This value 0.40 represents only an average concentration of the gene, thus the gene is repressed approximately 60% of the time and is expressed 40% of the time. To more accurately model the dynamics of these systems, a stochastic formulation that models the system using discrete-valued chemical populations must be employed.

1.3. Stochastic Modeling

To more accurately model systems affected by small populations or noise, we can describe a biochemical system using a stochastic formulation that represents species as discrete-valued populations and chemical interactions (reactions) as random processes (Gillespie, 1977; Kot, 2001). The dynamics of such a formulation can be described by the Chemical Master Equation (CME) (Gillespie, 1992), an equation that describes how the probability of being in each discrete system state (different combinations of species population values) changes with time. The following is a derivation of the CME.

We begin by expressing a biochemical system model as a set of species with initial populations and a set of chemical reactions with rate constants. Let the M molecular species in the model be represented by a vector $<S>=\{S_1,\ldots,S_M\}$ with species populations $<X(t)>=\{X_1(t),\ldots,X_M(t)\}$. Let the N chemical reactions in the system be represented by a vector $<R>=\{R_1,\ldots,R_N\}$. Each reaction R_i is defined by a stochastic rate constant k_i and a stoichiometry, represented as vectors of M reactant coefficients $<r_i>=\{r_{i1},\ldots,r_{iM}\}$ and M product coefficients $<p_i>=\{p_{i1},\ldots,p_{iM}\}$. Figure 4 describes the autoregulated biochemical model from Figure 1 in this form.

First, we consider reaction R_1 in Figure 4. This reaction defines that in the presence of a single gene molecule, mRNA molecules are introduced into the system at a rate k_{tc}. Let the function $E(R_i,\Delta t)$ represent the number of times we execute a reaction R_i during the time interval $(t,t+\Delta t]$. We assume that the probability of executing reaction R_1 in a short time Δt, given that the current gene population is one ($X_G(t) = 1$), is given by the following equations.

$$\Pr\{E(R_1,\Delta t)=1 \mid X_G(t)=1\}=k_{tc}\Delta t+o(\Delta t) \tag{1-7}$$

$$\Pr\{E(R_1,\Delta t)>1 \mid X_G(t)=1\}=o(\Delta t) \tag{1-8}$$

$$\Pr\{E(R_1,\Delta t)=0 \mid X_G(t)=1\}=1-k_{tc}\Delta t+o(\Delta t) \tag{1-9}$$

The $o(\Delta t)$ term in this equation represents higher order terms of Δt and becomes negligible as Δt becomes decreasingly small.

$$\lim_{\Delta t\to 0}\frac{o(\Delta t)}{\Delta t}=0 \tag{1-10}$$

Now consider the presence of two gene molecules in the system ($X_G(t) = 2$). These gene molecules will cause the introduction of mRNA molecules into the system independently, therefore the following equation describes the probability of executing R_1 once during Δt in this situation.

$$\begin{aligned}&\Pr\{E(R_1,\Delta t)=1 \mid X_G(t)=2\}= \\ &\quad \Pr\{E(R_1,\Delta t)=1 \mid X_G(t)=1\}\Pr\{E(R_1,\Delta t)=0 \mid X_G(t)=1\}+ \\ &\quad \Pr\{E(R_1,\Delta t)=0 \mid X_G(t)=1\}\Pr\{E(R_1,\Delta t)=1 \mid X_G(t)=1\}\end{aligned} \tag{1-11}$$

The first term represents the probability of executing one reaction using the first gene molecule and no reactions using the second gene molecule. The second term represents the probability of executing no reactions using the first gene molecule and one reaction using the second. Substituting equations 1-7 and 1-9 into this equation yields the following simplification.

$$\begin{aligned}&\Pr\{E(R_1,\Delta t)=1 \mid X_G(t)=2\}= \\ &\quad (k_{tc}\Delta t+o(\Delta t))(1-k_{tc}\Delta t+o(\Delta t))+(1-k_{tc}\Delta t+o(\Delta t))(k_{tc}\Delta t+o(\Delta t))\end{aligned} \tag{1-12}$$

$$\Pr\{E(R_1,\Delta t)=1 \mid X_G(t)=2\}=2k_{tc}\Delta t+o(\Delta t) \tag{1-13}$$

Expanding this equation to account for N gene molecules yields the following equations.

$$\begin{aligned}&\Pr\{E(R_1,\Delta t)=1 \mid X_G(t)=N\}= \\ &\quad \sum_{i=1}^{N}\left(\Pr\{E(R_1,\Delta t)=1 \mid X_G(t)=1\}\right)\left(\Pr\{E(R_1,\Delta t)=0 \mid X_G(t)=1\}\right)^{N-1}\end{aligned} \tag{1-14}$$

$$\Pr\{E(R_1,\Delta t)=1 \mid X_G(t)=N\}=\sum_{i=1}^{N}(k_{tc}\Delta t+o(\Delta t))(1-k_{tc}\Delta t+o(\Delta t))^{N-1} \tag{1-15}$$

$$\Pr\{E(R_1,\Delta t)=1 \mid X_G(t)=N\}=k_{tc}N\Delta t+o(\Delta t) \tag{1-16}$$

We further simplify this equation to write a general equation for executing R_1 during Δt given the current species populations $\langle X(t)\rangle$.

$$\Pr\{E(R_1,\Delta t)=1 \mid \langle X(t)\rangle\}=k_{tc}X_G(t)\Delta t+o(\Delta t) \tag{1-17}$$

Equation 1-9 can also be generalized for the presence of multiple gene molecules based on the fact that individual gene molecules generate mRNA molecules independently.

$$\Pr\{E(R_1,\Delta t)=0 \mid X_G(t)=N\}=\Pr\{E(R_1,\Delta t)=0 \mid X_G(t)=1\}^N \tag{1-18}$$

$$\Pr\{E(R_1,\Delta t)=0 \mid X_G(t)=N\}=(1-k_{tc}\Delta t+o(\Delta t))^N \tag{1-19}$$

$$\Pr\{E(R_1,\Delta t)=0 \mid X_G(t)=N\}=1-k_{tc}N\Delta t+o(\Delta t) \tag{1-20}$$

$$\Pr\left\{E(R_1,\Delta t)=0 \mid \left\langle X(t)\right\rangle\right\}=1-k_{tc}X_G(t)\Delta t+o(\Delta t) \tag{1-21}$$

In a similar manner, equation 1-8 can be generalized.

$$\Pr\left\{E(R_1,\Delta t)>1 \mid \left\langle X(t)\right\rangle\right\}=o(\Delta t) \tag{1-22}$$

Considering R_2 in Figure 4, this reaction defines that individual mRNA molecules leave the system at a rate k_{tc}. These molecules will decay independently, therefore using the same procedure given above, we can write the following probability equations for R_2.

$$\Pr\left\{E(R_2,\Delta t)=1 \mid \left\langle X(t)\right\rangle\right\}=k_{md}X_M(t)\Delta t+o(\Delta t) \tag{1-23}$$

$$\Pr\left\{E(R_2,\Delta t)>1 \mid \left\langle X(t)\right\rangle\right\}=o(\Delta t) \tag{1-24}$$

$$\Pr\left\{E(R_2,\Delta t)=0 \mid \left\langle X(t)\right\rangle\right\}=1-k_{md}X_M(t)\Delta t+o(\Delta t) \tag{1-25}$$

In fact, the probability functions for all of the individual reactions will have the same general form.

$$\Pr\left\{E(R_i,\Delta t)=1 \mid \left\langle X(t)\right\rangle\right\}=\alpha_i\left(\left\langle X(t)\right\rangle\right)\Delta t+o(\Delta t) \tag{1-26}$$

$$\Pr\left\{E(R_i,\Delta t)>1 \mid \left\langle X(t)\right\rangle\right\}=o(\Delta t) \tag{1-27}$$

$$\Pr\left\{E(R_i,\Delta t)=0 \mid \left\langle X(t)\right\rangle\right\}=1-\alpha_i\left(\left\langle X(t)\right\rangle\right)\Delta t+o(\Delta t) \tag{1-28}$$

The function α_i is called the "propensity" function. Its value is the product of the reaction rate constant and the number of possible ways the reaction can occur based on the current reactant populations. The propensity functions for the autoregulated biochemical model given in Figure 4 are provided below.

$$R_1: \quad \alpha_1=k_{tc}X_G(t) \tag{1-29}$$

$$R_2: \quad \alpha_2=k_{md}X_M(t) \tag{1-30}$$

$$R_3: \quad \alpha_3=k_{tl}X_M(t) \tag{1-31}$$

$$R_4: \quad \alpha_4=k_{pd}X_P(t) \tag{1-32}$$

$$R_5: \quad \alpha_5=k_{di}\frac{X_P(t)\left(X_P(t)-1\right)}{2} \tag{1-33}$$

$$R_6: \quad \alpha_6 = k_{rdi} X_D(t) \tag{1-34}$$

$$R_7: \quad \alpha_7 = k_{bind} X_D(t) X_G(t) \tag{1-35}$$

$$R_8: \quad \alpha_8 = k_{unbind} X_R(t) \tag{1-36}$$

For reactions with a single reactant like R_1, R_2, R_3, R_4, R_6, and R_8, the number of ways that the reaction can occur is equal to the current reactant population, thus equations 1-29, 1-30, 1-31, 1-32, 1-34, and 1-36 all share the same propensity function form. For reactions with two different reactants like R_7, any molecule of the first reactant species can react with any molecule of the second reactant species, therefore we take the product of the two species populations when finding the propensity as shown in equation 1-35. For dimerization reactions like R_5, where the reactants consist of two molecules of the same species, one molecule of the current species can react with any other molecule of that species, but not with itself, so we arrive at equation 1-33.

Now suppose we want to write an equation that describes the probability executing one of any of the reaction channels during a short time Δt given the current species populations $\langle X(t) \rangle$. Because our probability functions depend only on the current state of the system and the probability of first reaction events are independent, we can combine the probability functions to form the following equations.

$$\begin{aligned} &\Pr\left\{\sum_{i=1}^{N} E(R_i, \Delta t) = 1 \mid \langle X(t) \rangle\right\} = \\ &\quad \sum_{i=1}^{N}\left[\Pr\left\{E(R_i, \Delta t) = 1 \mid \langle X(t) \rangle\right\} \prod_{j=1}^{N, j\neq i} \Pr\left\{E(R_j, \Delta t) = 0 \mid \langle X(t) \rangle\right\}\right] \end{aligned} \tag{1-37}$$

$$\begin{aligned} &\Pr\left\{\sum_{i=1}^{N} E(R_i, \Delta t) = 1 \mid \langle X(t) \rangle\right\} = \\ &\quad \sum_{i=1}^{N}\left[\left(\alpha_i\left(\langle X(t) \rangle\right)\Delta t + o(\Delta t)\right) \prod_{j=1}^{N, j\neq i}\left(1 - \alpha_j\left(\langle X(t) \rangle\right)\Delta t + o(\Delta t)\right)\right] \end{aligned} \tag{1-38}$$

$$\Pr\left\{\sum_{i=1}^{N} E(R_i, \Delta t) = 1 \mid \langle X(t) \rangle\right\} = \sum_{i=1}^{N} \alpha_i\left(\langle X(t) \rangle\right)\Delta t + o(\Delta t) \tag{1-39}$$

A similar procedure can be used to find the probability of executing no reactions.

$$\Pr\left\{\sum_{i=1}^{N} E(R_i,\Delta t)=0 \mid \langle X(t)\rangle\right\}=\prod_{j=1}^{N}\Pr\left\{E(R_j,\Delta t)=0 \mid \langle X(t)\rangle\right\} \tag{1-40}$$

$$\Pr\left\{\sum_{i=1}^{N} E(R_i,\Delta t)=0 \mid \langle X(t)\rangle\right\}=\prod_{j=1}^{N}\left(1-\alpha_j\left(\langle X(t)\rangle\right)\Delta t+o(\Delta t)\right) \tag{1-41}$$

$$\Pr\left\{\sum_{i=1}^{N} E(R_i,\Delta t)=0 \mid \langle X(t)\rangle\right\}=1-\sum_{i=1}^{N}\alpha_i\left(\langle X(t)\rangle\right)\Delta t+o(\Delta t) \tag{1-42}$$

Let us now define a new function *F(<X>,t)* to describe the probability that the model's species populations will have a particular value *<X>* at time t.

$$F\left(\langle X\rangle,t\right)=\Pr\left\{X(t)=\langle X\rangle\right\} \tag{1-43}$$

If at time *t*, our species populations are *<X(t)>* and the next reaction we execute is R_i at time $t+\tau$, then new species populations are determined by decrementing the reactants species and incrementing the product species as shown in the following equation.

$$\langle X(t+\tau)\rangle=\langle X(t)\rangle-\langle r_i\rangle+\langle p_i\rangle \tag{1-44}$$

We can now write an expression for *F(<X>,t+Δt).*

$$\begin{aligned} &F(\langle X\rangle,t+\Delta t)= \\ &\quad F(\langle X\rangle,t)\Pr\left\{\sum_{i=1}^{N} E(R_i,\Delta t)=0 \mid \langle X\rangle\right\}+ \\ &\quad \sum_{i=1}^{N} F\left(\langle X\rangle+\langle r_i\rangle-\langle p_i\rangle,t\right)\Pr\left\{E(R_i,\Delta t) \mid \langle X\rangle+\langle r_i\rangle-\langle p_i\rangle\right\}+o(\Delta t) \end{aligned} \tag{1-45}$$

In words, this equation says that the probability of being in state *<X>* at time $t+\Delta t$ is:

1. The probability of remaining in the state given that the system is already in that state.

2. The probability of moving into the state from another state through the execution of a single reaction.

3. Higher order terms.

This equation can then be simplified to the following equation.

$$F(\langle X\rangle, t+\Delta t) = F(\langle X\rangle, t)\left(1-\sum_{i=1}^{N}\alpha_i(\langle X\rangle)\Delta t\right)+ \sum_{i=1}^{N}\left[F(\langle X\rangle+\langle r_i\rangle-\langle p_i\rangle, t)(\alpha_i(\langle X\rangle+\langle r_i\rangle-\langle p_i\rangle)\Delta t)\right]+o(\Delta t) \quad (1\text{-}46)$$

Rearranging equation 1-46 gives us the following equation.

$$\frac{F(\langle X\rangle, t+\Delta t)-F(\langle X\rangle, t)}{\Delta t} = \sum_{i=1}^{N}\left[F(\langle X\rangle+\langle r_i\rangle-\langle p_i\rangle, t)(\alpha_i(\langle X\rangle+\langle r_i\rangle-\langle p_i\rangle))\right]- \sum_{i=1}^{N}F(\langle X\rangle, t)\alpha_i(\langle X\rangle)+\frac{o(\Delta t)}{\Delta t} \quad (1\text{-}47)$$

Taking the limit as Δt approaches zero gives us the Chemical Master Equation.

$$\frac{dF(\langle X\rangle, t)}{dt} = \sum_{i=1}^{N}\left[F(\langle X\rangle+\langle r_i\rangle-\langle p_i\rangle, t)(\alpha_i(\langle X\rangle+\langle r_i\rangle-\langle p_i\rangle))\right]-\sum_{i=1}^{N}F(\langle X\rangle, t)\alpha_i(\langle X\rangle) \quad (1\text{-}48)$$

1.4. Exact Stochastic Simulation

For most biochemical system models, including the relatively simple model given in Figure 1, the number of discrete system states is infinite, thus the CME is an infinite set of ordinary differential equations making it difficult to solve analytically. In such cases, it is common to predict the behavior of the CME using a Monte Carlo simulation technique first developed by Gillespie called the Stochastic Simulation Algorithm (SSA) (Gillespie, 1977). The function of the SSA is to predict a potential time evolution of the species population for a given time period. It does this by predicting the execution time of each individual reaction event for the given system. A stochastic simulation of the autoregulated biochemical model given in Figure 4 is provided in Figure 5 and Figure 6.

Gillespie's original versions of the SSA are the First Reaction Method (FRM) and the Direct Method (DM) (Gillespie, 1976; Gillespie, 1977). Several mathematically equivalent algorithms have been proposed to help relieve some of the computational demands of the SSA. These algorithms include the Next Reaction Method (NRM) by Gibson (2000) and the Optimized Direct Method (ODM) by Cao (2004).

Each version of the SSA consists of several stages.

1. Initialization – Read the model file and initialize data structures.
2. Propensity Calculation – Calculate the propensity function for each reaction.
3. Reaction Time Generation – Estimate the occurrence time of the next reaction.
4. Reaction Selection – Select which reaction will occur next.
5. Reaction Execution – Update the current simulation time and species variables to reflect the execution of the selected reaction.
6. Termination – If the simulation time has not yet reached the desired end time, go to stage 2.

A summary of the differences between algorithms is given in Figure 7. Each of these algorithms is discussed in detail in the following sections.

1.5. The First Reaction Method

Based on equations 1-26 and 1-28, the probability of executing a reaction R_i at time $t+\tau_i$ given that no other reactions occur during this time period is given by the following equation (Gillespie, 1976).

$$\Pr\left\{R_i @ t+\tau_i \mid \langle X(t)\rangle\right\} d\tau = \alpha_i\left(\langle X(t)\rangle\right) e^{-\alpha_i\left(\langle X(t)\rangle\right)\tau_i} d\tau \tag{1-49}$$

Solving for the cumulative distribution function, we find the following equation.

$$\Pr\left\{R_i @ [t, t+\tau_i] \mid \langle X(t)\rangle\right\} = 1 - e^{-\alpha_i\left(\langle X(t)\rangle\right)\tau_i} \tag{1-50}$$

We can then solve this equation for τ_i and use a uniformly distributed random number U to generate potential estimates for τ_i.

$$\tau_i = \frac{-\ln(1-U)}{\alpha_i\left(\langle X(t)\rangle\right)} = \frac{-\ln(U)}{\alpha_i\left(\langle X(t)\rangle\right)} \tag{1-51}$$

The variable τ_i represents a potential time that reaction R_i will occur given that all of the other reactions did not occur. By generating potential times for each reaction, we can generate an estimate of when and what reaction will occur next by selecting the smallest of the generated potential times. This is the basis for the First Reaction Method (FRM), which is outlined in Figure 8.

For a potential simulation of the model given in Figure 4 using the FRM, we would begin by reading the model file and initializing data structures. The initial values for the species populations are reflective of a single gene with no mRNA, protein, or dimer

molecules yet introduced into the system. This is just one of many possible initial values these species populations could take.

$$\text{CurrentTime} = 0$$
$$\langle X \rangle = \{G=1, M=0, P=0, D=0, R=0\}$$
$$\langle R \rangle = \{R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8\}$$

In step 2, we would calculate propensities for each reaction based on the current species populations, according to equations 1-29 through 1-36.

$$\langle Prop \rangle = \{k_{tc}X_G,\ k_{md}X_M,\ k_{tl}X_M,\ k_{pd}X_P,\ k_{di}X_P(X_P-1)/2,\ k_{rdi}X_D,\ k_{bind}X_DX_P,\ k_{unbind}X_R\}$$

$$\langle Prop \rangle = \{10,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0\}$$

In step 3, we generate potential times by generating unifom random numbers for each of the reactions and scaling them to exponentially distributed random numbers using equation 1-51. The function `rand()` returns a uniformly distributed random number from 0 to 1.

```
For I = 1..N
     T[I] = -ln(rand())/ Prop[I]
End For
```

In this example, we get the following values for the potential reaction occurrence times.

$$\langle T \rangle = \{0.22,\ \infty,\ \infty,\ \infty,\ \infty,\ \infty,\ \infty,\ \infty\}$$

In step 4, we select the reaction with the smallest putative time, which in this example is R_1. This would set both the selected reaction and the minimum time.

$$\text{SelRxn} = 1$$
$$\text{MinTime} = 0.22$$

In step 5, we update the species populations by subtracting the selected reaction's reactants and adding the selected reaction's products.

$$\langle X \rangle = \langle X \rangle - \langle r_{sel} \rangle + \langle p_{sel} \rangle$$
$$\langle X \rangle = \{1,0,0,0,0\} - \{1,0,0,0,0\} + \{1,1,0,0,0\}$$
$$\langle X \rangle = \{1,1,0,0,0\}$$

We also update the current time to reflect the execution of the selected reaction.

$$\text{CurrentTime} = \text{CurrentTime} + \text{MinTime}$$
$$\text{CurrentTime} = 0 + 0.22 = 0.22$$

We now check to see if our end time is reached. If not, we repeat the process.

First, we calculate each reaction propensity.

$$\langle Prop \rangle = \{k_{tc}X_G,\ k_{md}X_M,\ k_{tl}X_M,\ k_{pd}X_P,\ k_{di}X_P(X_P-1)/2,\ k_{rdi}X_D,\ k_{bind}X_DX_P,\ k_{unbind}X_R\}$$

$$\langle Prop \rangle = \{10,\ 1,\ 10,\ 0,\ 0,\ 0,\ 0,\ 0\}$$

We draw random samples for each reaction to estimate potential reaction occurrence times.

$$\langle T \rangle = \{0.33,\ 1.26,\ 0.44,\ \infty,\ \infty,\ \infty,\ \infty,\ \infty\}$$

We select the reaction with the smallest potential occurrence time and update our species population and time variables to reflect the execution of the selected reaction.

$$SelRxn = 1$$
$$MinTime = 0.33$$

$$\langle X \rangle = \langle X \rangle - \langle r_{sel} \rangle + \langle p_{sel} \rangle$$
$$\langle X \rangle = \{1,1,0,0,0\} - \{1,0,0,0,0\} + \{1,1,0,0,0\}$$
$$\langle X \rangle = \{1,2,0,0,0\}$$

$$CurrentTime = CurrentTime + MinTime$$
$$CurrentTime = 0.22 + 0.33 = 0.55$$

The process is repeated until the desired end time is reached.

1.6. The Direct Method

Because each iteration of the FRM main loop (steps 2-6) requires the generation of N exponentially distributed random numbers, the computation time per step of the algorithm is high and the scalability of the algorithm to models with many reaction channels is limited. To overcome this problem, Gillespie (1977) designed the Direct Method (DM) that requires the generation of only two random numbers per iteration regardless of the model size.

Based on equations 1-39 and 1-42, the probability of executing a reaction R_i at time $t+\tau_i$ is given by the following equation (Gillespie, 1976).

$$\Pr\{R_i @ t+\tau_i \mid \langle X(t)\rangle\} d\tau = \alpha_i\left(\langle X(t)\rangle\right) e^{-\sum_{j=1}^{N}\alpha_j\left(\langle X(t)\rangle\right)\tau_i} d\tau \tag{1-52}$$

We can therefore generate system reaction times by solving the cumulative distribution function for τ_i.

$$\tau_i = \frac{-\ln(U)}{\sum_{j=1}^{N}\alpha_j\left(\langle X(t)\rangle\right)} \tag{1-53}$$

We can also select reactions based on the following probability distribution.

$$\Pr\left\{E(R_i,\Delta t)=1 \mid \langle X(t)\rangle, \sum_{j=1}^{N, j\neq i} E(R_j,\Delta t)=0\right\} = \frac{\alpha_i\left(\langle X(t)\rangle\right)}{\sum_{j=1}^{N}\alpha_j\left(\langle X(t)\rangle\right)} \tag{1-54}$$

Gillespie showed that this algorithm is mathematically equivalent to the FRM and the CME. The DM reduces the number of random numbers per step of the algorithm from $O(N)$ to $O(1)$. Pseudo-code for the DM is provided in Figure 9.

For a potential simulation of the autoregulated biochemical model given in Figure 4 using the DM, we would again begin by reading the model file and initializing data structures.

```
CurrentTime = 0
<X> = {G=1,M=0,P=0,D=0,R=0}
<R> = {R1,R2,R3,R4,R5,R6,R7,R8}
```

In step 2, we calculate the reaction propensities and the total propensity.

```
<Prop> = {ktcXG, kmdXM, ktlXM, kpdXP, kdiXP(XP-1)/2,
          krdiXD, kbindXDXP, kunbindXR}
<Prop> = {10, 0, 0, 0, 0, 0, 0, 0}
TotalPropensity = 10
```

In step 3, we generate a reaction execution time for the system using a randomly generated uniform random number and scaling it using equation 1-53. In this instance, the function `rand()` returns the uniformly distributed random number 0.1862.

```
T = -ln(rand())/10 = -ln(0.1862)/10 = 0.1681
```

In step 4, we select a reaction according to the probability distribution function given in equation 1-54. This is accomplished by the following procedure.

```
Selector = TotalPropensity * rand()
```

```
For I = 1..N
  Selector = Selector - Prop[I]
  If (Selector <= 0)
    SelRxn = I
    Break
  End If
End For
```

Our example would first calculate a value for the selector by generating a uniformly distributed random number and multiplying it by the current total propensity.

```
Selector = TotalPropensity * rand()
Selector = 10 * 0.5775 = 5.775
```

We would then enter the for loop and subtract the first propensity.

```
Selector = Selector - Prop[1] = 5.775 - 10 = -4.225
```

Since the selector has become negative, we break out of the loop and set our selelected reaction index to 1.

```
SelRxn = 1
```

In step 5, we again update the species populations by subtracting the selected reaction's reactants and adding the selected reaction's products.

$$\langle X \rangle = \langle X \rangle - \langle r_{sel} \rangle + \langle p_{sel} \rangle$$
$$\langle X \rangle = \{1,0,0,0,0\} - \{1,0,0,0,0\} + \{1,1,0,0,0\}$$
$$\langle X \rangle = \{1,1,0,0,0\}$$

We also update the current time to reflect the execution of the reaction.

```
CurrentTime = CurrentTime + MinTime
CurrentTime = 0 + 0.1681 = 0.1681
```

We check to see if our end time is reached and if not, we repeat the process.

First, we calculate the reaction propensities and the total reaction propensity.

```
<Prop> = {10, 1, 10, 0, 0, 0, 0, 0}
TotalPropensity = 21
```

Next, we generate a uniformly distributed random number and compute the reaction occurrence time.

```
T = -ln(rand())/21 = -ln(0.6444)/21 = 0.0209
```

We select the reaction to execute by generating a selector and subtracting propensities until the selector becomes negative.

```
Selector = TotalPropensity * rand()
Selector = 21 * 0.5111 = 10.7331

Selector = Selector - Prop[1] = 10.7331 - 10 = 0.7331
Selector = Selector - Prop[2] = 0.7331 - 1 = -0.2669
SelRxn = 2
```

Lastly, we update the species populations and current time variables.

$$\langle X \rangle = \langle X \rangle - \langle r_{sel} \rangle + \langle p_{sel} \rangle$$
$$\langle X \rangle = \{1,1,0,0,0\} - \{0,1,0,0,0\} + \{0,0,0,0,0\}$$
$$\langle X \rangle = \{1,0,0,0,0\}$$

```
CurrentTime = CurrentTime + MinTime
CurrentTime = 0.1681 + 0.0209 = 0.1890
```

The process is repeated until the desired end time is reached.

1.7. The Next Reaction Method

The performance of exact stochastic simulation was later improved by Gibson (2000) with the Next Reaction Method (NRM). This is accomplished through enhancing the First Reaction Method by rescaling potential reaction occurrence times and adding several useful data structures.

Realizing that for most models the execution of a single reaction affects the value of only a few reaction propensities, the NRM builds a reaction "dependency graph" during initialization by examining the stoichiometry of the model. This graph describes which propensities must be recalculated when a particular reaction occurs and eliminates unnecessary recalculation of unchanged propensity values. The graph is determined by first collecting the set of species affected by the execution of a reaction. Any reactions whose propensity values depend on the affected species are added to the update list for the executed reaction. A sample dependency graph for the model given in Figure 4 is provided in Figure 10. Employing this data structure significantly reduces the number of propensity calculations required by step 2 of the simulation for most biochemical models.

Another performance improvement comes from reusing reaction occurrence times. Recall that in the First Reaction Method, we generate potential execution times for each

reaction then select reaction with the earliest potential time. Reaction times of the unexecuted reactions are discarded and a new set of times is generated during each step of the algorithm.

In the Next Reaction Method, instead of discarding the generated potential reaction times, the times are rescaled and a new time is generated only for the executed reaction. Given that the species populations at time t are equal to $\langle X(t) \rangle$ and a single reaction occurs at time $t+dt$ that changes the species populations to $X(t+dt)$, if our original estimated execution time for reaction R_i is equal to τ_i, we can rescale this time using the following equation.

$$\tau_i = \frac{\alpha_i\left(\langle X(t) \rangle\right)}{\alpha_i\left(\langle X(t+dt) \rangle\right)}\left(\tau_i - (t+dt)\right) + (t+dt) \tag{1-55}$$

This equation states that we can rescale the original reaction times by subtracting the new current time ($t+dt$) and multiplying by the ratio of the old propensity to the new reaction propensity. This operation gives us a scaled relative potential occurrence time (a change in time relative to the current simulation time) and by adding back the new current time ($t+dt$), we get a scaled absolute potential occurrence time (a change in time relative to the original time at the beginning of the simulation). Notice that if the propensity of the reaction is unchanged, τ_i remains unchanged.

Reusing random numbers in simulation algorithms typically leads to inaccuracies, but (Gibson, 2000) shows that in this specific case, by transforming the random potential occurrence time variables from relative time (as they were in the FRM) to absolute time, the probability distribution functions for the reaction times are equivalent. By proving that the NRM and FRM are mathematically equivalent, Gibson and Bruck prove that the NRM is mathematically equivalent to the CME. This adaptation of the algorithm leads to a significant performance gain over the FRM and DM, because after generating N exponentially distributed random numbers for the first iteration of the algorithm, only one random number is required for subsequent steps.

The final enhancement added by the NRM was through the addition of an "indexed priority queue", that maintains a heap of potential reaction occurrence times in sorted order to reduce the amount of time required to select which reaction will occur next. The structure is a complete binary tree and each update to a node requires O(log N) time.

An outline of the NRM is given in Figure 11.

For a potential simulation of the model given in Figure 4 using the NRM, we again begin by reading the model file and initializing data structures. For this example, the initial species populations have been modified to give the reader a better impression of the overall execution of the algorithm.

```
CurrentTime = 0
<X> = {G=1,M=4,P=55,D=107,R=0}
<R> = {R1,R2,R3,R4,R5,R6,R7,R8}
```

We would additionally initialize the dependency graph so that it would reflect the data structure shown in Figure 8.

We then calculate the propensities of each reaction in the model.

$$\langle Prop \rangle = \{k_{tc}X_G,\ k_{md}X_M,\ k_{tl}X_M,\ k_{pd}X_P,\ k_{di}X_P(X_P-1)/2,\ k_{rdi}X_D,\ k_{bind}X_DX_P,\ k_{unbind}X_R\}$$

$$\langle Prop \rangle = \{10,\ 4,\ 40,\ 55,\ 1485,\ 1070,\ 10.7,\ 0\}$$

We generate reaction execution times for each reaction by generating uniform random numbers and scaling them using equation 1-51. We store the times in an indexed priority queue (MinHeap).

$$\langle T \rangle = \{0.12,\ 0.31,\ 0.043,\ 0.044,\ 2.3e{-}3,\ 3.4e{-}4,\ 0.15,\ \infty\}$$

```
MinHeap.add(<T>)
```

We select the reaction to execute by reading the root node of the indexed priority queue, which will contain the reaction with the smallest potential reaction occurrence time.

```
SelRxn = MinHeap.getMinIndex() = 6
```

In step 5, we update the species populations by subtracting the selected reaction's reactants and adding the selected reaction's products.

$$\langle X \rangle = \langle X \rangle - \langle r_{sel} \rangle + \langle p_{sel} \rangle$$
$$\langle X \rangle = \{1,4,55,107,0\} - \{0,0,0,1,0\} + \{0,0,2,0,0\}$$
$$\langle X \rangle = \{1,4,57,106,0\}$$

We also update the current time to reflect the execution of the reaction.

```
CurrentTime = T[SelRxn] = 3.4e-4
```

If our end time has not yet been reached, we return to step 2 and update only the changed propensity values based on the entries for the selected reaction in the dependency graph (those which are underlined have been changed).

$$DependentRxns = \{4,\ 5,\ 6,\ 7\}$$
$$\langle Prop \rangle = \{10,\ 4,\ 40,\ \underline{57},\ \underline{1596},\ \underline{106},\ \underline{10.6},\ 0\}$$

We now update each of the affected reaction's potential occurrence times using equation 1-55 and generate a new potential occurrence time for the previously executed reaction R_6 using equation 1-51. As we update the times, we also update the indexed priority queue to assure that the minimum time value remains at the top of the heap.

```
<T> = {0.12, 0.31, 0.043, 0.0425,
       2.16e-3, 6.81e-3, 0.151, ∞}
MinHeap.update(<T>)
```

We again select the reaction to execute by reading the reaction at the root node of the indexed priority queue, and update the species populations and times accordingly.

```
SelRxn = MinHeap.getMinIndex() = 5

<X> = <X> - <r_sel> + <p_sel>
<X> = {1,4,57,106,0} - {0,0,2,0,0} + {0,0,0,1,0}
<X> = {1,4,55,107,0}

CurrentTime = T[SelRxn] = 2.16e-3
```

The process is repeated until the desired end time is reached.

1.8. The Optimized Direct Method

The Optimized Direct Method (ODM) (Cao, 2004) is the most recent algorithm designed to improve the performance of exact stochastic simulation and is based on the Direct Method. The ODM makes several key improvements to the DM.

The first change is to utilize a dependency graph to recalculate only the propensity values that have changed, in a manner equivalent to the NRM. Also the dependency graph is used to aid in the recalculation of total propensity. Instead of summing all of the reaction propensities, the total propensity is updated by adding the difference between old and new propensity values for reactions whose propensity values have been changed.

The second change builds on the observation that when executing the NRM, a majority of the computational time used by the algorithm is spent maintaining the indexed priority queue data structure. The ODM uses an alternative approach for reducing the complexity of the reaction selection stage (step 4).

Performing the reaction selection stage (step 4) for the DM involves searching the propensity values to find the value of *sel,* the index of the selected reaction. This must be done in a way that maintains the reaction probabilities defined by the distribution given in equation 1-54. This is most easily accomplished using the pseudo-code found in Figure 12. Notice that as soon as the selected reaction is found, the search loop terminates. The

number of steps required for this process to complete will be referred to as "search depth."

Cao realized that by moving reactions that execute more frequently to the beginning of the reaction search order, the average search depth for the simulation would be reduced. Since most biological models have a few reactions that are executed frequently and a majority of the other reactions are executed infrequently, this modification can significantly reduce computation time. To determine which reactions will execute most frequently, the ODM executes pre-simulations that track reaction execution frequencies. This information is then used to presort the reactions in order of decreasing frequency of occurrence.

Pseudo-code for the ODM is provided in Figure 13.

For a potential simulation of the model given in Figure 4 using the ODM, we would begin by reading in the model and initializing a dependency graph.

```
CurrentTime = 0
<X> = {G=1,M=0,P=0,D=0,R=0}
<R> = {R1,R2,R3,R4,R5,R6,R7,R8}
```

We would then perform a pre-simulation to determine a histogram of the reaction execution counts. A sample histogram generated by executing such a pre-simulation is given in Figure 14. Notice that reactions R_5 and R_6 are most frequently executed. Based on this sample histogram, using this ordering of the reactions, the average search depth for the system is estimated to be 5.4. Sorting the reactions so that the most frequently executed reactions are at the top of the reaction search order, as shown in Figure 15, reduces the estimated average search depth to 1.6, significantly enhancing performance.

We would then reinitialize the simulator with an updated order of reactions.

```
CurrentTime = 0
<X> = {G=1,M=0,P=0,D=0,R=0}
<R> = {R5,R6,R3,R4,R1,R2,R7,R8}
```

In step 2, we calculate the propensities and the total propensity.

$$\langle Prop\rangle = \{k_{di}X_P(X_P-1)/2,\ k_{rdi}X_D,\ k_{tl}X_M,\ k_{pd}X_P,\ k_{tc}X_G,\ k_{md}X_M,\ k_{bind}X_DX_P,\ k_{unbind}X_R\}$$

```
<Prop> = {0, 0, 0, 0, 10, 0, 0, 0}
TotalPropensity = 10
```

In step 3, we generate a reaction execution time for the system by scaling a uniformly distributed random number using equation 1-53.

```
T = -ln(rand())/10 = -ln(0.1862)/10 = 0.1681
```

In step 4, we select a reaction according to the probability distribution function given in equation 1-54. This is accomplished using the same procedure used in the DM.

```
Selector = TotalPropensity * rand()
Selector = 10 * 0.2813 = 2.813

Selector = Selector - Prop[1] = 2.813 - 0 = 2.813
Selector = Selector - Prop[2] = 2.813 - 0 = 2.813
Selector = Selector - Prop[3] = 2.813 - 0 = 2.813
Selector = Selector - Prop[4] = 2.813 - 0 = 2.813
Selector = Selector - Prop[5] = 2.813 - 10 = -7.187
SelRxn = 5 (previously R1)
```

In step 5, we again update the species populations by subtracting the selected reaction's products and adding the selected reaction's products.

```
<X> = <X> - <r_sel> + <p_sel>
<X> = {1,0,0,0,0} - {1,0,0,0,0} + {1,1,0,0,0}
<X> = {1,1,0,0,0}
```

We also update the current time to reflect the execution of the reaction.

```
CurrentTime = CurrentTime + MinTime
CurrentTime = 0 + 0.1681 = 0.1681
```

If our end time has not yet been reached, we return to step 2 and update the total propensity and the changed propensity values based on the entries for the selected reaction in the dependency graph (those which are underlined have been changed).

```
DependentRxns = {3, 6}
TotalPropensity = TotalPropensity - Prop[3] - Prop[6]
TotalPropensity = 10 - 0 - 0 = 10
<Prop> = {0, 0, 10, 0, 10, 1, 0, 0}
TotalPropensity = TotalPropensity + Prop[3] + Prop[6]
TotalPropensity = 10 + 10 + 1 = 21
```

We then proceed in the same way we did for the DM.

```
T = -ln(rand())/21 = -ln(0.6056)/21 = 0.0502

Selector = TotalPropensity * rand()
Selector = 21 * 0.5100 = 10.71
Selector = Selector - Prop[1] = 10.71 - 10 = 0.71
```

```
Selector = Selector - Prop[2] = 0.71 - 1 = -0.29
SelRxn = 2

<X> = <X> - <r_sel> + <p_sel>
<X> = {1,1,0,0,0} - {0,1,0,0,0} + {0,0,0,0,0}
<X> = {1,0,0,0,0}

CurrentTime = CurrentTime + MinTime
CurrentTime = 0.1681 + 0.0502 = 0.2183
```

By reducing the overhead of the NRM heap structure, Cao demonstrated that their algorithm outperforms the NRM and is the fastest known algorithm for simulating typical biochemical models.

1.9. Approximation Techniques

Larger systems, such as whole-cell or multi-cell models that require the execution of billions of reactions, can take an unreasonable amount of time to simulate using the SSA (Endy, 2001). Compounding this performance problem is the fact that the algorithm only predicts a single potential time-evolution (random walk) of the chemically reacting system. Typically, multiple simulations with different initial random seeds are required to collect the data and statistics necessary to elucidate the system behavior. Improving the performance of the SSA is therefore critical to expanding the applicability of this technique.

To address the computational performance challenges presented by exact stochastic simulation ("exact" refers to algorithms which are statistically equivalent to the CME), researchers have developed a variety of approximation methods. These methods attempt to collapse or condense computational work to generate large performance gains while sacrificing an acceptable amount of accuracy.

One of the more promising areas of approximation research that has received a significant amount of recent attention is called "tau-leaping." This technique, first proposed by Gillespie (2001), condenses the execution of multiple reactions events into a single simulation step by making "leaps" in simulation time and using Poisson random variables to determine how many reaction events occurred during the leap. The key to the success of this technique is to select a leap size large enough to allow many reactions to occur during the leap, reducing computation, and small enough that none of the propensity functions will change significantly in value, causing error. This technique has continued to mature, particularly in the area of leap-size selection, through the work of a variety of several researchers (Gillespie, 2003; Rathinam, 2003; Tian, 2004) and further work in this area can be expected.

Another technique that has been effectively applied to the analysis of biochemical systems is the application of Chemical Langevin Equations (Gillespie, 2000; Simpson, 2003; Simpson, 2004). This approach begins with a set of ordinary differential equations (like the deterministic approach described in section 1.2) and adds time-dependent noise functions to each equation to represent the stochastic fluctuations (noise) in the network.

Other approximation techniques create hybrid simulations by combining deterministic and stochastic approaches. These techniques rely on the observation that most biochemical models are comprised of reactions with largely different timescales. The "fast" reactions, ones that execute on a short timescale relative to the other reactions in the system, typically dominate the computational processing time (Rao, 2003). Finding ways to reduce the complexity of processing the "fast" reactions through deterministic approximations while minimizing the amount of error introduced by these approximation is the primary goal of these techniques.

One such hybrid approach is implemented in the tool BioNetS (Biochemical Network Simulator), where users characterize certain reactions as "fast", so they can be approximated using ordinary differential equations, while the remaining "slow" reactions are simulated using the SSA (Adalsteinsson, 2004). Salis (2005) has produced a similar hybrid algorithm that uses Chemical Langevin Equations to simulate the "fast" reactions.

Another example of a hybrid simulation technique is the "slow-scale" stochastic simulation algorithm (Cao, 2005). This technique partitions the reactions into "slow" and "fast" categories based on their initial propensity values. The steady-state probability distribution functions for the "fast" species (species that are controlled by "fast" reactions) are computed and used to calculate the propensity functions of the "slow" reactions. The "slow" reactions are then simulated using a modified version of the SSA. One limitation of this algorithm is that calculation of the steady-state probability distribution functions for the "fast" species involves solving the CME, which can only be accomplished for simple reaction structures.

For detailed reviews of many of these approximation techniques and their application to biochemical systems, readers are encouraged to consult (Turner, 2004; Burrage, 2003).

While the author acknowledges the promise of these approaches, these techniques have not matured to the point where typical biochemical modelers can know when, and more importantly when not, to apply these methods. Although exact stochastic simulation techniques require significantly more computational effort, they remain effective, trusted, and accurate general-purpose tools for properly modeling stochastic effects in biochemical processes. This work therefore focuses solely on improving the performance of "exact" stochastic simulation techniques that are mathematically equivalent to Gillespie's SSA and the CME.

1.10. Approach

To accelerate stochastic simulation of biochemical systems, this work will assess the applicability of a variety of hardware and software acceleration techniques. Chapter 2 discusses accelerating a serial version of the algorithm by modifying the SSA to reduce the computational complexity of executing a single model simulation on a standard personal computer. Chapter 3 investigates parallelizing the SSA and executing simulations on parallel computer architectures like supercomputers, multi-processor desktop systems, and multi-core processors. In Chapter 4, custom hardware accelerators are developed to analyze the possibility of accelerating stochastic simulation using reconfigurable computing. Chapter 5 investigates the plausibility of using clusters of computers to perform multiple simulations in parallel for parameter sweeping and optimization. Through this work, the state-of-the-art in stochastic simulation is expanded and a discussion of what architectures and techniques should be applied to particular modeling problems is presented.

CHAPTER 2 - Serial Algorithm Improvements

This chapter is a revised version of a paper published in the journal *Computational Biology and Chemistry* in 2006 by James McCollum, Greg Peterson, Chris Cox, Mike Simpson, and Nagiza Samatova:

McCollum JM, Peterson GD, Cox CD, Simpson ML, Samatova NF, "The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior." *Journal of Computational Biology and Chemistry* 30 (1): 39-49, 2006.

My use of "we" refers to my co-authors and myself. My primary contributions to this paper include (1) a majority of the writing, (2) the formation of the literature review, (3) the implementation and performance analysis of the algorithms, and (4) the development and analysis of the sorting direct method.

2.1. Introduction

This chapter discusses improving the performance of serial exact stochastic simulation, that is, accelerating the execution of a single exact stochastic simulation of a typical biochemical model on a single processor.

To begin this study, implementations of the First Reaction Method (Gillespie, 1977), Direct Method (Gillespie, 1977), Next Reaction Method (Gibson, 2000), and Optimized Direct Method (Cao, 2004) are constructed. Their performance is then compared by using them to simulate a variety of biochemical system models.

Through this analysis, we discover that transient changes in reaction execution frequencies, which are typical of biochemical models with gene induction and repression, can dramatically affect simulator performance. To account for these shifts, a new algorithm called the Sorting Direct Method is proposed that maintains a loosely sorted order of the reactions as the simulation executes. Performance measurements show that the Sorting Direct Method performs favorably when compared to other well-known exact stochastic simulation algorithms.

We then show that the Sorting Direct Method can be customized for the specific problem of determining long-term probability distribution functions for particular species in a biochemical system. This customization allows us to eliminate the need to compute reaction times. Additional analysis shows a significant performance improvement when using this optimization.

2.2. Models

A significant portion of the text reviewing the biochemical models in this section was contributed by Dr. Chris D. Cox, Department of Civil and Environmental Engineering, University of Tennessee – Knoxville, and was used with his permission.

This section describes the models used for the study of algorithm performance, specifically, quorum sensing in *Vibrio fischeri* (Cox, 2003), the lysogeny decision in *λ-phage* (Arkin, 1998), and heat stress response in *Escherichia coli* (Kurata, 2001). Model files for each of these systems can be found in the Appendix.

Vibrio fischeri is a single celled prokaryote that colonizes the light organs of the Hawaiian bobtail squid *Euprymna scolopes*. Together, they share a symbiotic relationship; bioluminescence generated by *V. fischeri* helps camouflage the squid from predators and the squid provides *V. fischeri* a nutrient rich environment within its light organ.

Quorum sensing is a cell-cell communication mechanism used by bacteria to coordinate activities such as bioluminescence, virulence, or biofilm formation (Greenberg, 2000). *V. fischeri* uses quorum sensing to regulate bioluminescence and other colonization factors such that they occur only when the concentration of cells is relatively high (Lupp, 2003). This mechanism is primarily controlled by the action of two genes. The gene *luxI* codes for a protein that catalyzes the production of an acyl-homoserine lactone (AHL) auto-inducer molecule. AHL can freely diffuse across the cell membrane and acts as a signaling molecule to surrounding cells. The gene *luxR* codes for a receptor protein, which upon complexation with AHL, positively regulates the activity of both genes. The bioluminescent genes are encoded downstream of *luxI* on the same operon. The positive feedback loop acts as a switch to turn on bioluminescence when the cell senses sufficiently large AHL concentrations to indicate that the cell density is high.

The quorum-sensing model used here is similar to the model of Cox (2003), but differs in several respects. This model includes abstract representations of cell growth and regulation of *luxR* by a secondary quorum sensing circuit. This model also ignores the DNA-looping repression mechanism included in the earlier model. Most significantly, this model captures the quorum-sensing dynamics of eight interacting cells. The model consists of 122 species and 201 reactions and is executed for 30,000 simulated seconds.

The second model represents the lysogeny decision circuit in phage-λ infected *Escherichia coli* cells. When the phage-λ virus infects *E. coli*, the phage DNA can direct the fate of the cell down one of two pathways. In the lytic case, the phage DNA hijacks the cell machinery to make several additional copies of the virus, the cell ruptures and the viruses are released to infect additional cells. Alternatively, the infected cell may be directed into a lysogenic state in which the phage DNA is incorporated into the bacterial DNA. In this case, as the cell divides the viral DNA is replicated along with the bacterial DNA and is thereby passed down to the cell's progeny. The core regulatory mechanisms

controlling the decision are encoded into a four-promoter, five-gene network and are well studied (Ptashne, 1992). Arkin (1998) formulated a stochastic model of this network and used it to demonstrate that the fraction of cells selecting each pathway was determined by the stochastic fluctuations in the gene expression levels. For this work, a simplified version of this model was borrowed from Dr. Chris Myers of the University of Utah. It consists of 61 species, 117 reactions, and is executed for 10,000 simulated seconds.

The third model used in this study was provided by Yang Cao of the University of California – Santa Barbara. This is the same model used in their work to characterize the performance of the Optimized Direct Method (Cao, 2004). The model is a representation of the heat stress response in *Escherichia coli*. Heat stress is the term used to describe the process of protein denaturing at elevated cell temperature. At normal temperatures, RNA polymerase is bound to the sigma factor $\sigma 70$. At elevated temperatures the activity of $\sigma 32$ increases, which redirects RNA polymerase to a set of approximately 20 heat stress genes that refold or degrade denatured proteins. A regulatory mechanism controls the activity of $\sigma 32$ in response to elevated temperature. This model consists of 28 species and 61 reactions and is simulated for 500 seconds. For a detailed description of this model, consult (Cao, 2004; Kurata, 2001).

2.3. Performance Analysis

Each of the four simulation algorithms described in Chapter 1, which include the First Reaction Method (FRM), the Direct Method (DM), the Next Reaction Method (NRM), and the Optimized Direct Method (ODM) are implemented in C++ and compiled with the maximum optimization flags using the GNU C++ compiler. Source code for these implementations can be found in the Appendix. To characterize the reaction execution frequency as required by the ODM, each model is pre-simulated for 5% of the total simulation time. The platform used to execute the simulations is a lightly loaded computer with two 2.4 GHz Intel Xeon Pentium 4 processors, each with a 512kB L2 cache. The computer has 1 GB of RAM and runs GNU Debian Linux. All computation time measurements are given in wall-clock time.

Results for executing the simulations using the four simulation algorithms described above are given in Table 1. The ODM results reflect the sum of the time spent executing the simulator and running the pre-simulation. Since each run is a different random-walk, the number of reactions executed by the simulator can vary. To give a fair performance comparison, we use reactions executed per second as the metric for performance comparison as shown in Table 2. The results support the claim by Cao (2004) that the ODM is the fastest known algorithm for exact stochastic simulation for typical biochemical models.

2.4. Transient Propensity Shifts

To determine an ordering of reactions and minimize reaction search depth, the ODM requires the execution of several pre-simulations to characterize the occurrence frequency of the reactions. For this to be effective, it is assumed that the reaction execution behavior exhibited at the beginning of the simulation will be characteristic of the long-term reaction execution behavior. The following example shows that this assumption may not be valid for many biochemical networks.

Suppose we are modeling the gene regulation network given in Figure 16. Here we see that the genes *Gene1* and *Gene2* are transcribed at a particular basal rate. When the transcription factor *Protein1* binds to *Gene2*, it acts as an inducer and transcribes at a greatly accelerated rate. A stochastic simulation of this system is given in Figure 17. Notice that just before time 200.0, *Protein1* binds to *Gene2* and the *Protein2* population increases dramatically. Figure 18 shows a normalized histogram of the reaction execution counts for the first 100.0 seconds of the simulation. Figure 19 shows a normalized histogram of the reaction execution counts for the entire simulation.

If the pre-simulations used to order the reactions were based solely on the histogram generated during the first 100 seconds (the first 25%) of the simulation, the pre-simulation would overemphasize the reactions controlling the transcription, translation, and decay of *Protein1*, when over the entire simulation the reactions controlling *Gene2* would be the reactions most frequently executed. This would increase the reaction search depth dramatically and decrease the performance of the simulator. Although the pre-simulation typically gives us a better reaction ordering than just randomly assigning the order of the reactions, there is no way to assure that our pre-simulation will be long enough to characterize the reaction execution behavior of the system and give us optimal simulator performance.

2.5. The Sorting Direct Method

To overcome the degradation in performance caused by models with transient propensity shifts and to eliminate costly pre-simulations, a new algorithm called the Sorting Direct Method (SDM) is proposed. This algorithm maintains a reaction selection order that is approximately sorted throughout the simulation. Each time a reaction is executed, its position is moved up in the reaction selection order. This moves reactions that have occurred recently toward the top of the reaction search list, effectively reducing the search depth for this reaction the next time it is executed. Since the sort requires only a swap of two memory addresses, the approximate sorting adds negligible overhead to the simulation. The sorting allows the algorithm to adapt to sharp changes in propensity by temporarily moving frequently executed reactions to the top of the reaction search order, thus handling the on/off behavior of natural switches in genetic regulatory networks. Pseudo-code for the algorithm is given in Figure 20.

To demonstrate the performance improvement, the model given in Figure 16 is expanded to produce a twenty gene model with ten genes induced by proteins that the other ten genes produce. This model can also be found in the Appendix. Figure 21 shows periodic measurements of the average search depth as the simulation executes for both simulators. Notice that the ODM shows sharp changes in the reaction search depth as the simulation executes. These sharp changes occur as genes are induced. As the run continues, the average search depth increases because the pre-simulation has not effectively predicted the reaction execution behavior for the run. The SDM successfully adapts to changes in reaction execution behavior and consistently remains at or below the average search depth measured by the ODM. Because the SDM is able to maintain a relatively small average search depth, it is able to perform the simulation at a rate of 2.05 million reactions executed per second while the ODM is only able to perform the simulation at a rate of 1.75 million reactions executed per second.

Even in situations where the reaction execution frequency remains constant throughout the simulation and the ODM accurately predicts the optimal order of reactions, the SDM performs as well as the ODM and adds negligible overhead. To show this, we use the same twenty gene model and set the initial species populations to their long-term steady state values. With this configuration, the pre-simulation properly sorts the reactions in the model to provide the ideal reaction search order for the ODM. Performing 100,000 reactions for the pre-simulation consumes 0.09 seconds and executing 10,000,000 reactions takes an additional 4.44 seconds for a total run time of 4.53 seconds. The SDM is able to execute 10,000,000 reactions in 4.47 seconds. This demonstrates that the overhead added by the SDM to maintain the reaction sort order is small and may be less than the pre-simulation time required for the ODM.

Finally, we execute the biological models from Section 2.2. Figure 22 shows a chart comparing the performance of the SDM to the four other simulation algorithms using the biological models from Section 2.2 and the twenty gene model. The results show that the SDM performs as well or better than the ODM for all models tested. The SDM shows a notable performance improvement when executing the λ-phage and quorum sensing models, because these models include significant state transitions (certain reactions are turned on or off throughout the simulation) that cause transient propensity shifts. The heat stress response model shows fairly consistent reaction execution behavior throughout the simulation, so the pre-simulation performed by the ODM reasonably characterizes the long-term reaction behavior for the system and the ODM performs comparably well to the SDM.

2.6. Eliminating Reaction Times

One common question that modelers will often use a stochastic simulator to answer is, "What is the long-term probability distribution of a particular species?" To determine this, the user must execute thousands of independent simulations using different random seed values and monitor the species population at the end of the simulation. These

simulations runs must be sufficiently long to allow the simulator to progress through transient behavior and move into a steady state. In such cases, it may not be necessary to know the individual time of each reaction and it may be sufficient to ask the user to estimate a count of the number of reactions to execute before the steady-state is reached.

If the user can provide a reaction count instead of an end time and is only concerned with the final value of one or more species populations, we can ignore individual reaction times and customize the Sorting Direct Method to produce results faster.

Pseudo-code for the Un-timed Sorting Direct Method (USDM) is given in Figure 23. Notice that the computation of exponentially distributed random numbers is eliminated and the maintenance of the current time variable is eliminated, which saves significant computation time. A comparison of the performance of the Un-timed Sorting Direct Method to the standard Sorting Direct Method is provided in Figure 24. This figure shows that a significant performance improvement can be achieved using the USDM.

2.7. Conclusion

The structure of regulatory networks in cells leads to large shifts in the transient behavior of protein populations. Failure to account for this behavior when stochastically simulating such models can negatively impact computational performance. By loosely sorting the reactions as the simulation executes, the Sorting Direct Method employs an effective strategy for handling large shifts in reaction propensities and eliminates the need for pre-simulations. Furthermore, in certain situations it may be sufficient to ignore the calculation of individual reaction times. In such cases, it is possible to optimize the Sorting Direct Method to achieve additional performance gains.

CHAPTER 3 - Parallel Exact Stochastic Simulation

This chapter is a revised version of a paper titled "Parallel exact stochastic simulation of coupled chemical reactions" by James McCollum, Greg Peterson, Chris Cox, Mike Simpson, and Nagiza Samatova currently under review by the journal *Computational Biology and Chemistry*.

My use of "we" refers to my co-authors and myself. My primary contributions to this paper include (1) a majority of the writing and (2) the development, implementation, and analysis of the parallel reaction method.

3.1. Introduction

After developing an optimized exact stochastic simulation algorithm for a standard personal computer workstation, the focus of this work shifts to accelerating exact stochastic simulation for parallel computational architectures, like supercomputers, multi-processor desktop computers, and multi-core processor workstations. As the cost of these technologies continues to decrease, these systems will become more affordable to the typical biological modeler. Algorithms that can efficiently utilize these technologies may help overcome performance bottlenecks associated with exact stochastic simulation. This chapter therefore chronicles the development of a parallel algorithm called the *Parallel Reaction Method* (PRM) that distributes the computational work of an exact stochastic simulation to multiple threads of execution.

The applicability of this algorithm is assessed by comparing the performance of parallel and serial methods on a Silicon Graphics Altix shared-memory supercomputer. The results demonstrate that the computational efficiency of this algorithm largely depends on the model to be simulated and the architecture of the system performing the simulation. Furthermore, applying the PRM to a large genetic regulatory network model of *Escherichia coli* shows a significant performance improvement over the fastest known serial methods and demonstrates the potential utility of this algorithm.

Research into the parallelization of exact stochastic simulation has largely been neglected by many scientists, because they believe that the SSA can not be parallelized efficiently. The only documented attempts to parallelize the SSA have been developed by Schwehm (2001) and implemented in E-Cell 3 by Arjunan (2003). These works have shown favorable performance on parallel spatial models, but can only be applied when diffusion is approximated deterministically. The PRM differs from this work in that it uses no approximations and is not limited to spatial models.

To achieve this performance gain, The *Parallel Reaction Method* (PRM) initially divides the model into subsections, assigning each thread to handle the execution of a subset of the reactions in the model. The simulation threads execute a modified version of the SSA on their assigned subsection. When the execution of a reaction by one thread affects the

outcome of reactions executed by another thread, a message is sent to the affected thread. The affected thread then updates its local state to maintain the mathematical accuracy of the simulation based on the message received. This message handling scheme is based on techniques employed in Time Warp, a parallel discrete event simulation technique (Jefferson, 1985).

3.2. Mapping the SSA to a Discrete Event Simulation

The PRM is based on discrete event simulation, a technique that has been applied in a number of fields and to a variety of complex systems (Li, 2003; Lee, 1996; Martin, 2002; Roy, 2004). A discrete event simulation consists of a set of state variables, an event list, and a global clock. As events are read in time order from the event list, the state variables and clock are updated, causing more events to be generated and updated in the event list. The simulator periodically reports the status of the state variables and continues to execute until the desired end-time is reached.

The SSA can be mapped to a discrete event simulation using species populations as state variables and generated reaction events as the event list. Parallel discrete event simulation (PDES) techniques can then be applied to address the parallelization of the SSA.

Two main types of PDES protocols exist, conservative and optimistic. In conservative PDES, the causality of reaction events is always maintained, meaning that events that depend on the same set of state variables will never be executed out of order. In optimistic PDES, causality can be violated and conflicts are resolved using sophisticated messaging schemes and simulation rollbacks.

For the PRM, an optimistic PDES method called Time Warp was chosen for parallelization. In this algorithm, each thread executes events (reactions) independently and maintains a local virtual time. When one thread changes a state variable (species) that another thread depends on, a message is placed in the affected thread's input queue with information about the event, the virtual time, and the ID of the thread that generated the event. If the affected thread's local virtual time is before the event, it continues to execute its simulation until it reaches the event and processes it accordingly. If the affected thread's local virtual time is after the event, the thread executes a "rollback," where the events processed after the message time are undone until the thread reaches the message time and processes the event. As the affected thread executes its rollback, the thread must also send anti-messages which are used to cancel all invalid messages previously sent by the affected thread. For a detailed introduction to PDES refer to Fujimoto (1990). For a detailed introduction to Time Warp, refer to Jefferson (1985).

3.3. The Parallel Reaction Method

The following is an outline of the *Parallel Reaction Method*.

1. Initialization – Divide the input model into subsections and assign threads to each subsection.
2. Local Event Generation – Generate a local reaction to execute R_{local} and an execution time T_{local} using a serial stochastic simulator.
3. Remote Event Processing – Read the earliest incoming event with reaction R_{remote} at time T_{remote} from the incoming event list.
4. Event Resolution
    a. If ($T_{local} < T_{remote}$)
        i. Update the serial stochastic simulator to reflect the execution of R_{local} at time T_{local}
        ii. Add an event message containing R_{local} and T_{local} to each thread affected by the execution of R_{local}.
    b. Else
        i. Rollback – Unexecute each reaction executed after T_{remote} sending anti-messages when necessary.
        ii. If the incoming event is a message, update the serial stochastic simulator to reflect the execution of R_{remote} at time T_{remote}.
        iii. If the incoming event is an anti-message, update the serial stochastic simulator to reflect the cancellation of R_{remote} at time T_{remote}.
5. Synchronization – (Executed every K seconds) Temporarily halt all threads and compute the global minimum time. If the global minimum time is after the desired simulation end time, terminate the simulation.

In step 1 of the *Parallel Reaction Method*, the simulation begins by reading an input model file that defines the species and reactions within the system. A reaction assignments file is also read that defines the number of threads to use and the reaction responsibilities for each thread. An array of threads is then initialized, each containing data structures to hold lists of incoming event messages, a history of events executed, local virtual times, species populations, and all other variables required to execute a serial stochastic simulation on the assigned subsection.

In step 2, threads generate a "local" event using a serial stochastic simulator. A "local" event includes a reaction to execute R_{local} from the thread's assigned subsection and a reaction execution time T_{local}. For our implementation of the *Parallel Reaction Method,* an enhanced version of the *Direct Method* that includes the sum tree and dependency graph data structures proposed by Gibson (2000) was selected as the serial stochastic simulator for event generation.

In step 3, the thread reads the earliest incoming remote event (if one exists) from its incoming event list. Here R_{remote} represents the reaction for the remote event and T_{remote} represents the time at which that reaction occurs.

In step 4, if the local event time is less than the remote event time, the simulator executes the local event and contacts all threads affected by the execution of the local event by placing a message defining the event in the affected thread's incoming event message list. The affected threads can easily be determined using a dependency graph (Gibson, 2000).

If the last event time is greater than the remote event time, the simulator must execute a rollback. This is accomplished by "un-executing" all of the reactions that occurred prior to the remote event time. The "un-execute" action decrements the species populations of the event's products, increments the species populations of the event's reactants, and updates the local serial simulator's local virtual time. If the event to un-execute is local to the thread, anti-messages must be sent to remote threads to cancel the effects of a message triggered by the execution of this event. If a message and anti-message are both added to the incoming event list, the two messages cancel each other. If the un-executed event was not a local reaction, the event must be added back to the incoming event list so that when the thread again reaches the time to execute this reaction, the message will be properly reprocessed. This loop continues until the event history is empty or the local time is less than the earliest remote event time. Upon completion of the rollback, Step 4 completes the remote event processing by updating the serial simulator to reflect the execution of the remote reaction.

In Step 5, the execution of the threads is periodically suspended and a global minimum time across all the threads is calculated based on the minimum of the local virtual times and the earliest message in the system. This procedure is called "global virtual time" (GVT) calculation in PDES. Because it is not possible to rollback to events that occur prior to the global minimum time, events that occurred prior to the global minimum time can be deleted from the event history stack, saving memory. Reclaiming this memory is called "fossil collection" in PDES. The threads are then resumed upon completion of the global minimum time calculation. The period of the global minimum time calculation is determined by the parameter K. Performing the calculation too frequently can negatively impact the performance due to the overhead of stopping all of the threads. Performing the calculation infrequently will cause the event histories to grow excessively large, consuming large amounts of memory. This implementation uses a value of five seconds for K, which was found to be effective in practice for the tested models.

To generate streams of independent random numbers, the implementation utilizes the Scalable Pseudo-Random Number Generator (SPRNG) library (Mascagni, 2000). Generating statistically independent random streams is critical to the accuracy of this algorithm. Violating this requirement may cause certain reactions to be incorrectly executed more frequently than others and will invalidate results. To assure the proper execution of the algorithm, a history of the random numbers used is maintained. On rollbacks, the unused random numbers are recycled and reused. While critical for

assuring the accuracy of the simulator, the SPRNG library introduces a small overhead (less than 10%) to the implementation of the *Parallel Reaction Method*. Future improvements may assign one or more threads to fill buffers of random numbers for further parallelization and performance improvement.

For accurate execution, care must also be taken to assure that random numbers are properly scaled when remote reactions are executed. This is done using a technique consistent with the time scaling used in the *Next Reaction Method* (Gibson, 2000). To determine the time of the next local reaction τ, we first estimate τ using our serial stochastic simulator under the assumption that no remote reactions will occur. We then process all remote events that occur after the current local virtual time and before τ, by scaling τ using equation 1-55, which is restated here.

$$\tau_i = \frac{\alpha_i\left(\langle X(t)\rangle\right)}{\alpha_i\left(\langle X(t+dt)\rangle\right)}\left(\tau_i - (t+dt)\right) + (t+dt) \tag{3-1}$$

An additional responsibility of the threads is to ensure that the local species count does not fall below zero as a result of a message. This can occur if a model is simulated where three threads T1, T2, and T3 all have at least one reaction that has a reactant X. If the population of X is 1 at a particular time, it is possible that thread T1 and thread T2 could simultaneously execute reactions that consume X. Eventually the reaction with the greater time will be cancelled by an anti-message, but if both messages are processed by T3 before the anti-message arrives, thread T3 will set the local population of X to –1. This will eventually be corrected by rollbacks from anti-messages, but until the anti-message arrives, thread T3 could begin to execute reactions so quickly and will send so many messages to threads T1 and T2, that the system can reach a state where it is overwhelmed by the generation and cancellation of messages. The threads must therefore check for negative propensity values and set a flag within the thread to suspend the processing of local events until an anti-message is received that eliminates the error condition.

3.4. Parallel Algorithm Proof

Given a system exists in state *X(t)* at time *t*, the probability that we do not execute a system reaction R_i during a short time Δt is given by the following equation, which was introduced in Chapter 1.

$$\Pr\left\{E(R_i, \Delta t) = 0 \mid \langle X(t)\rangle\right\} = 1 - \alpha_i\left(\langle X(t)\rangle\right)\Delta t + o(\Delta t) \tag{3-2}$$

Because our probability function depends only on the current state of the system and the probability of first reaction events are independent, this equation holds for all of the reactions in the system. Based on this equation, Gillespie (1976) showed that we could

generate potential execution times for each system reaction given a current state using equation 3-3.

$$\tau_i = \frac{-\ln(U)}{\alpha_i\left(\langle X(t)\rangle\right)} \tag{3-3}$$

By selecting the smallest of the generated potential occurrence times, we could determine what time the next reaction would execute. These equations are the foundation for the First Reaction Method.

Because the probability of each first reaction event is independent, we can expand equation 3-2 to write an equation for the probability that no reactions in the system are executed during a short time *Δt*.

$$\Pr\left\{\sum_{i=1}^{N} E(R_i,\Delta t) = 0 \mid \langle X(t)\rangle\right\} = 1 - \sum_{i=1}^{N} \alpha_i\left(\langle X(t)\rangle\right)\Delta t + o(\Delta t) \tag{3-4}$$

Based on this equation, Gillespie (1977) showed that the next reaction time could be generated using equation 3-5.

$$\tau_i = \frac{-\ln(U)}{\sum_{j=1}^{N} \alpha_j\left(\langle X(t)\rangle\right)} \tag{3-5}$$

This equation is the basis for the Direct Method. This equation holds because each of the potential reaction execution times are independent and exponentially distributed, therefore the distribution of the first reaction events for the aggregate is also an exponential distribution with a parameter that is the sum of the propensities.

Gillespie proved that generating individual times for each reaction and selecting the minimum reaction time was equivalent to generating one reaction time based on the summation of all of the reaction propensities. Furthermore, he showed that both of these algorithms are mathematically equivalent to the Chemical Master Equation.

Using similar logic, suppose we divided the reactions in the system into a set of disjoint reaction sets. The probability of not executing any of the reactions in a particular sets *S* during a short time *Δt* would be given by equation 3-6.

$$\Pr\left\{\sum_{i\in S} E(R_i,\Delta t) = 0 \mid \langle X(t)\rangle\right\} = 1 - \sum_{i\in S} \alpha_i\left(\langle X(t)\rangle\right)\Delta t + o(\Delta t) \tag{3-6}$$

The propensity of the set is the sum of the propensities of the reactions within the set. A potential reaction time for each set could be determined using equation 3-7.

$$\tau_i = \frac{-\ln(U)}{\sum_{j \in S_i} \alpha_j \left(\langle X(t) \rangle \right)} \tag{3-7}$$

This is the basis for the Parallel Reaction Method. The model is initially divided into sets of reactions which are assigned to processors. An optimized version of the Direct Method is used to determine a reaction time for the reactions in the processor's assigned set and the earliest of these reactions is selected as the reaction set to execute.

Furthermore, Gibson (2000) showed that we could generate potential reaction occurrence times for a set of reactions, select the reaction with the smallest occurrence time, and determine new reaction times by scaling the original reaction times using equation 3-1. Gibson (2000) also proved that this algorithm was equivalent to the First Reaction Method. By similar logic, we can modify the Next Reaction Method so that instead of generating individual times for each reaction and scaling them as reactions occurred, we can divide the model's reactions into sets, generate potential reaction times using the Direct Method, select the earliest, and scale the remaining reaction times for each set to generate new reaction times.

The Parallel Reaction Method functions in precisely this way and is thus equivalent to the Next Reaction Method. The Parallel Reaction Method first breaks the model's reactions into sets of reactions and assigns them to processors. Each processor uses an optimized version of the Direct Method to generate potential reaction times for their set. The processor that generates the earliest potential reaction time executes its reaction and the remaining processors scale their reaction times based on equation 1-55.

The main difference between the Parallel and Next Reaction Methods is that the Parallel Method allows the processor to assume that the reaction time generated by the method was the earliest time. The Parallel Method then continues to execute reactions, thus speculating what the next reactions executed would be based on this assumption.

The messaging scheme, which is equivalent the Jefferson Time Warp, assures that messages will be sent to the processor that correct for this assumption if it was made in error. When the Parallel Method must correct for this using a rollback, all of the assumed reactions are discarded and the processor is reset to the point just before it executed the first invalid reaction. The next reaction time is then scaled using equation 1-55, in a method equivalent to the Next Reaction Method.

Because we maintain the integrity of the random number list by storing it as we execute and using it when we rollback, the Parallel Reaction Method, like the Next Reaction Method, functions as a deterministic simulator. This means that subsequent runs of the simulator using the same initial random seed will generate identical results, regardless of the ordering of messages between processors.

3.5. Performance Analysis

To analyze the performance of the Parallel Reaction Method, an implementation of the algorithm was executed on Oak Ridge National Laboratory's SGI Altix parallel supercomputer. This machine features 256 Intel Itanium2 processors running at 1.5 GHz with 2 TB of shared memory.

The PRM was implemented on a shared memory system, because this simplified the calculation of global minimum time, the coordination of messages, and minimized communication latency. The algorithm could easily be adapted to execute on a distributed memory architecture using message passing libraries such as PVM (Geist, 1994) or MPI (Snir, 1998).

Research into the performance of the Parallel Reaction Method indicates that two major factors impact the simulation speed: communication and load imbalance. The artificial model represented in Figure 25 is formulated to illustrate these ideas. The model, consisting of 18 species and 50 reactions, is symmetrical with two 3x3 sections (used to represent a set of reactions local to a thread) and a reversible reaction connecting the two sections (used to represent reactions that cause inter-thread communication). To perform a parallel simulation of this model, two threads are used with one thread assigned to each of the 3x3 sections.

For most computational architectures, the communication link between processing elements is much slower than the processing element, thus message passing between threads can cause significant time consumption. Also, the communication link is typically a shared resource (i.e. a bus connecting symmetric multi-processors, a router connecting a cluster of machines) and a large number of messages between multiple processing elements can overwhelm the system. The speed and efficiency of the Parallel Reaction Method is therefore highly dependent on the speed of this link and the amount of inter-thread communication required by the model.

To illustrate the performance degradation as the amount of communication increases, the model given in Figure 25 is configured so that local reactions have a rate constant of 1. The rate constant of the two inter-thread communication reactions (the reversible reaction between species A3 and B1) is varied to analyze the impact on performance. The performance results of these simulations, measured in reactions executed per second and communications per second (based on the average of three simulations), are given in Figure 26. As expected, as the rate constants of the inter-thread communication reactions are increased, the amount of inter-thread communication increases and a significant degradation in performance is observed. The results of this study depend on the message latency of the interconnection link between threads and would vary between hardware architectures.

Another model attribute that can affect the performance of the Parallel Reaction Method is "load balance" or the balance of reaction propensities between threads. If one thread

takes comparatively small steps in simulation time because its set of reactions has a larger average total propensity than reactions assigned to other threads, messages from the "slow" thread will cause large rollbacks on the other threads with far advanced simulation times degrading performance. It is therefore important to assign reactions to threads in such a way that balances their propensities.

This phenomenon can also be illustrated using the artificial model given in Figure 25. First, the inter-thread communication reaction rate constants are set to 0.0001, a value which in Figure 26 was shown to not cause a performance degradation due to communication overload. Next, the rate constants for thread 1's local reactions (the reversible reactions connecting all of the A species) are set to 1 and the rate constants of thread 2's local reactions (the reversible reactions connecting all of the B species) are varied. As thread 2's rate constants are increased, the average total propensity of thread 2 becomes proportionally large compared to the average total propensity of thread 1. This causes thread 2 to take much smaller time steps than thread 1. The performance results of these simulations measured in reactions executed per second and reactions unexecuted per second (based on the average of three simulations) are given in Figure 27. Notice that as thread 2's rate constants are increased, the load imbalance of the system increases, which increases the length of rollbacks (measured in the number of reactions unexecuted per second) and degrades performance.

These results indicate that partitioning reactions to threads in order to minimize communication and balance load is a critical step in maximizing the performance of the algorithm. Presently, this partitioning task must be handled manually by the user. Automating this process, potentially through the application of graph partitioning algorithms to the dependency graph, is an area of ongoing research.

Because the models in Chapter 2 contain tightly coupled reactions that can not be partitioned in a manner that allows the Parallel Reaction Method to achieve a performance gain, we test the performance of the Parallel Reaction Method on a large-scale model of *Escherichia coli* that is derived from motif data provided by Shen-Orr (2002). The motif data contains the inductive and repressive relationships among 423 genes. A model is generated from this data using the interpretive structure given in Figure 28 (defined in detail in Figure 29). The interpretive structure consists of transcription, translation, and dimerization reactions for each of the 423 genes. Overall the model consists of 2,299 species interconnected through 4,782 reactions.

Measurements of the parallel simulator performance (based on the average of five independent simulations) are given in Figure 30. Also shown is the performance of the two fastest known serial simulation algorithms, the *Next Reaction Method* (Gibson, 2000) and the *Sorting Direct Method* (McCollum, 2006). The data reveals that an over 5X performance improvement can be achieved over the fastest serial algorithm when 32 processors are used.

3.6. Conclusion

The performance of the *Parallel Reaction Method* depends greatly on the structure of the model (its ability to be mapped to multiple threads and achieve load balance while requiring little communication) and the computational architecture chosen to simulate the model (the speed at which messages can be passed between threads). The results show that for a large-scale model of *E. coli* simulated on a parallel shared-memory supercomputer, a significant performance improvement over the fastest known serial algorithms can be achieved. As low-cost parallel computational architectures mature and more multi-thread and multi-core thread desktop machines become available to computational biologists and chemists, the *Parallel Reaction Method* could become a significant contributor to the acceleration of biochemical system modeling.

CHAPTER 4 - Hardware Acceleration

4.1. Introduction

After discussing ways in which we can accelerate exact stochastic simulation on single-processor and multi-processor machines, we now shift our focus to the development of custom hardware acceleration units for exact stochastic simulation using Field Programmable Gate Arrays (FPGAs). FPGAs are "reconfigurable" digital logic devices, meaning the system design is not fixed and can be re-customized to fit the user's specific needs. The user can develop multiple task-specific circuits and reprogram the FPGA to execute any of the circuit designs. Because the FPGA logic is customizable and can execute multiple tasks in parallel, iterative tasks like those typically found in loops of serial programs can often be accelerated using custom logic. As FPGAs continue to increase in speed and decrease in price, hardware accelerated designs may soon become a viable solution for accelerating many computationally intensive tasks. In this chapter, we will discuss implementing a custom logic circuit for accelerating exact stochastic simulation.

Several attempts to accelerate the SSA using FPGAs have already been attempted with some success. One of the most fundamental works (Yoshimi, 2004) describes a hardware implementation of the First Reaction Method on an FPGA for a four reaction Lotka system. The authors were able to create a design that would achieve a throughput of approximately 2 million reactions per second. By pipelining the design and replicating it multiple times on the chip, they were able to perform 148 independent simulations simultaneously, resulting in 304 million reactions executed per second. This translated into an over 100X performance improvement over the software simulator.

Although the performance improvement is significant, using this system requires the modeler to develop VHDL or Verilog descriptions of their models, then synthesize them using FPGA synthesis and layout tools. The modeler must have FPGA synthesis software on their computer, which is expensive, and must synthesize each model they design. Many hardware design modules can take hours or even days to synthesize, thus it may take longer to synthesize the design than it would to simply simulate the design using a software implementation.

Other work on using FPGAs for stochastic simulation rely on introducing approximations to the SSA (Lok, 2004; Salwinski, 2004; Keane, 2004). In certain cases such approximations may effectively model the system, but the authors do not address the circumstances under which these assumptions may introduce significant error.

Each of these works contribute to the goal of applying reconfigurable computing to stochastic simulation, but do not address the practical issues of giving the biochemical modeler a simple and efficient system that can effectively replace the use of software simulators. The goal of the hardware design presented here is to develop a system that will not require the user to synthesize hardware description language code and will not

create hard-coded designs specific to one particular model. This method will attempt to create a system that can be sent commands to change the model structure without being reconfigured and could be used to replace pure software stochastic simulators.

4.2. Initial Work

First attempts to develop an effective hardware accelerated system that does not require re-synthesis for each model change were developed on the Pilchard Reconfigurable Computing Platform with Brandon Thurmon.

This work is discussed in a conference paper that we co-authored (Thurmon, 2005) and is discussed in detail in his master's thesis (2005). Sections of these texts are quoted and summarized in this section. My contributions to this work were (1) helping to plan the hardware design, (2) writing portions of the conference paper text, and (3) revising the conference paper text. Brandon Thurmon was responsible for the overall planning and implementation of the initial design discussed in this section.

The Pilchard reconfigurable computing platform system consists of a Pentium III 933 MHz processor with a Xilinx Virtex 1000E FPGA in one of the system's memory slots. Placing the FPGA in the memory slot allows for fast communication between the FPGA and the host processor.

A block diagram of the original system is given in Figure 31. The host processor sends commands to the FPGA to configure the species populations, and the reaction equations. The reaction equations include the reaction rate constant, the reactant indices and coefficients, and the product indices and coefficients. The species populations and reaction equations are used as input to propensity calculator blocks, which are depicted in Figure 32. These blocks use multiplexers to select the species populations of the reaction's reactants and multiply them by the reaction's rate constant. Each block is responsible for calculating the propensity of one of the reactions.

The resulting propensity values are used as input to a summation block to generate the total propensity of the system. A linear feedback shift register is used to generate a uniformly distributed random number, that is used as input into a reaction selection block that selects a reaction based on the procedure developed in the Direct Method. The selected reaction is then sent to the species update unit. This unit executes the reaction by subtracting the selected reaction's reactants and adding the selected reaction's products to the species populations.

The data returned to the CPU during each step of the algorithm includes the total propensity and the index of the selected reaction. The CPU maintains a copy of the species populations and generates reaction occurrence times in parallel to the generation of reaction events by the FPGA.

Using this system to execute several models with 16 reactions or less consumes 98% of the FPGAs slice resources and produces a speedup of approximately 1.5X when compared to executing the Sorting Direct Method on the host processor.

Although this design failed to significantly enhance overall performance and was limited to relatively small biochemical network models, it provided insights that led to the development of the more successful design described in the following sections.

4.3. Simplifying Propensity Calculation

The initial design contains several features that limit performance. The first is that each species population has to be tied to each propensity calculator, which can cause routing problems. For example, if species populations were represented as 16-bit numbers and there are 32 species in the system, adding a single reaction to the system (which would require the addition of a single propensity calculator block) would require an additional 32x16 = 512 signals to be added from the species populations to the propensity calculator. The solution therefore had limited scalability as we increase the number of species and reactions the system can handle. This problem does not affect the hard-coded simulators presented in section 4.1, because the multiplexer unit can be eliminated in fixed-model designs and only species populations that pertain to the specific reaction in the model will be routed to each a propensity calculation block.

Furthermore, the addition of each propensity calculation block requires the addition of one or more multiplication units, which consumes significant space on the chip.

In the new design, we eliminate these problems by setting the initial propensity value for each block and only requiring the propensity calculator block to update (not recalculate) the propensities. To accomplish this, each propensity calculator must be provided with the following information.

- K – The reaction rate constant
- A – The index of the first reactant species
- B – The index of the second reactant species
- KA – The product of the K and A
- KB – The product of K and B
- KAB – The product of K, A, and B, which represents the current propensity.

If we execute a reaction that increments the value of species A, we can update the values stored in the propensity calculator using the following two equations.

$$KA = KA + K \tag{4-1}$$

$$KAB = KAB + KB \tag{4-2}$$

Similarly if we decrement species B, we can update the values using the following equations.

$$KB = KB - K \tag{4-3}$$

$$KAB = KAB - KA \tag{4-4}$$

These new optimized propensity calculators can be connected to a species update bus that indicates which species are updated by the execution of a reaction. For this system to work properly, when we execute a particular reaction, we must sequentially execute each of the species population updates and send them on the species update bus. For instance, if we execute the reaction `X=>2Y+Z`, we execute the following instructions:

- First Clock Cycle – Decrement species X.
- Second Clock Cycle – Increment species Y.
- Third Clock Cycle – Increment species Y again.
- Fourth Clock Cycle – Increment species Z.

For most reactions, the number of species population updates is only 1 or 2, therefore these additional clock cycles do not significantly impact the performance of the design.

A block diagram of the optimized propensity calculator is given in Figure 33. The inputs to the optimized propensity calculator are a bus for receiving species updates, not the species populations for each species. Adding additional reactions to the system therefore only requires the additional logic for the block and additional signals for setting the values of K, KA, KB, KAB, A, and B. It does not require that each of the species populations values be routed to the block. The block can be implemented with only a few addition/subtraction units and two compare units instead of the larger multiplication units.

4.4. The Hardware Design

A block diagram of the hardware design is given in Figure 34. This design uses optimized propensity calculators, fed into a summation unit to generate total propensity values, and then into a reaction selection unit to select reactions. When a reaction to execute is determined, the finite state machine reads data from a memory unit that specifies what species need to be updated for the given reaction. These values are sent along the species update bus to update the propensities. The process repeats until a certain number of user specified reactions have taken place.

The species monitor block is used to monitor the population of a user-specified species. One or more of these blocks can be added to the design to monitor multiple species values. Once a species index and an initial species population are specified for the

species monitor block, the block can monitor the current population of the species by reading the species update bus.

The finite state machine allows the user to configure a model programmatically, run a predetermined number of reactions, and determine the species population of a species of interest at the end of the simulation run. This is all that is required for determining a long-term steady state probability distribution function for a particular species in a specified model. Customizing the finite state machine to allow the total propensity and selected reaction indices to be written to a ram block so they can be read and processed by a CPU, as was done in the original design, requires simple changes to the finite state machine that will not affect performance.

The block has the following inputs:

- Clock – Controls the speed at which the design operates.
- Write Enable – Indicates that a command is ready to be processed.
- Command – The command to be processed.
- Address – The address parameter for the command.
- Data – The data parameter for the command.

The following list describes each of the available commands in detail.

- Command “0000” – Sets the value of KAB for the reaction indexed by the address parameter to the value given in the data parameter.
- Command “0001” – Sets the value of K for the reaction indexed by the address parameter to the value given in the data parameter.
- Command “0010” – Sets the value of KA for the reaction indexed by the address parameter to the value given in the data parameter.
- Command “0011” – Sets the value of KB for the reaction indexed by the address parameter to the value given in the data parameter.
- Command “0100” – Sets the value of A, the species index of the first reactant, for the reaction indexed by the address parameter to the value given in the data parameter.

- Command “0101” – Sets the value of B, the species index of the second reactant, for the reaction indexed by the address parameter to the value given in the data parameter.

- Command “0110” – Sets the initial population value of the species that we wish to monitor throughout the simulation.

- Command “0111” – Sets the index of the species that we wish to monitor throughout the simulation.

- Command “1000” – Sets the species to update when executing the reaction indexed by the address parameter. This command can be used to set up to 8 different species to update when a particular reaction is updated.

- Command “1001” – Sets the reaction counter, which specifies how many reactions we wish to execute before completing the simulation.

- Command “1111” – This is the “go” command. It instructs the block to simulate the system for as many reactions as is indicated by the reaction counter and to indicate when it has completed the simulation via the done signal.

The outputs of the block include the following items.

- Done – Indicates when the go command has completed executing all of its reactions.

- Species – The current population of the species of interest.

4.5. Performance Analysis

The design is implemented on a Xilinx Virtex 2 Pro XCV2P30 FPGA. Measurements of device resource utilization in terms of slices is given in Table 3. Maximum operating frequency measurements are given in Table 4. The table shows the results for designs that support 8, 16, 32, and 64 reactions for propensity sizes of 16, 24, and 32 bits.

To generate different hardware designs for various bit-widths and reaction counts, a java program is implemented to generate VHDL code. Source code for the java code and the generated VHDL code are given in the Appendix.

We calculate the total propensity from the individual reaction propensities using a summing tree structure that takes $\log_2(N)$ clock cycles to complete, where N is the number of reactions. We then consume one clock cycle to multiply the total propensity by the generated random number. The reaction selection stage consumes $\log_2(N)$ clock

cycles. Reading species updates from the species update memory requires two clock cycles. Each update to a species population requires a clock cycle (we will refer to this number as UpdateCount). Therefore, the number of clock cycles required to process a single reaction is given by the following equation.

$$\text{Clocks} = 2 \log_2 N + 3 + \text{UpdateCount} \qquad (4\text{-}5)$$

The system allows for UpdateCount values from 1 to 8. For typical models, the average UpdateCount is between 1 and 2. Using an assumed value of 2 for the UpdateCount, Table 5 shows the estimated number of reactions executed per second for each design based on the data reported in Table 4.

4.6. Prototyping the Design

The design is implemented on a Xilinx XUP Virtex II Pro Development System board. This board includes a Xilinx Virtex 2 Pro XCV2P30 FPGA and a variety of input/output connections including PS/2 ports, an RS232 serial port, an Ethernet Port, several switches, and several light emitting diodes. The board is programmed through a USB cable that interfaces to a host computer.

A wrapper module is built around the stochastic simulation block to allow data to be passed to and from a host machine through the RS232 serial port. Although this is not how the system would be implemented in practice, it provides a simple and flexible interface for testing the system. A complete design would include a faster mechanism for reading and writing data to the system. The complete hardware design includes an RS232 transceiver and a finite state machine for controlling signals and clocks between the transceiver and the stochastic simulation module. The stochastic simulation module uses a 50 MHz input clock based on clock dividing the 100 MHz system clock signal. Source VHDL code for the complete design is given in the Appendix.

Several models are built to test the performance of the prototyped system. The first is a model of the transcription and translation of a single gene. The model includes external noise sources from RNAP and ribosomes represented as birth-death processes. This model was recently used to perform noise analysis in a paper accepted for publication in the journal Chaos (Cox, 2006). The rate constants are rescaled to integer values for use within the FPGA. A diagram of the model consisting of 8 reactions and 4 species is given in Figure 35.

A second model is built by expanding the Chaos model to represent the transcription and translation of 15 different genes and consists of 64 reactions and 32 species. This model is depicted in Figure 36.

A third model represents diffusion of a chemical species between an external environment and 32 cells. This model consists of 33 species and 64 reactions and is depicted in Figure 37. Model files for each of these models is given in the Appendix.

Each model is executing using the hardware prototype and using the Sorting Direct Method on the same platform that was used in Chapter 2. The hardware reaction rates given reflect the time required to execute the simulation, but do not include the time to configure the FPGA, because this would only need to be done a single time when the hardware system was first powered up. Furthermore the measurements do not include the time to load the model from the computer to the FPGA, because this would include the slow speed of the RS232 serial port communication, which would not be a factor in a production quality system.

The Sorting Direct Method can simulate the single cell transcription model at a rate of 2.7 million reactions per second. Using the 8 reaction 24-bit hardware accelerated simulator, we can execute 4.9 million reactions per second. The hardware accelerated version is 1.8 times faster than the software for this model.

Using the 64 reaction 32-bit hardware implementation, we can simulate the 15 gene transcription and translation model at a rate of 3.1 million reactions per second and the diffusion model at a rate of 2.9 million reactions per second. The Sorting Direct Method simulates the 15 gene model at a rate of 2.2 million reactions per seconds and the diffusion model at a rate of 0.5 million reactions per seconds. The hardware is 1.4 times faster for the 15 gene model and is 5.8 times faster for the diffusion model.

For the 64 reaction models, we notice that the FPGA performs both of the simulations at nearly the same rate. For a given reaction count, the hardware's performance depends only of the average update count of the model, which is 1.0 for the 15 gene model and 2.0 for the diffusion model. These performance results are summarized in Table 6.

The Sorting Direct Method shows much larger fluctuations in reaction execution rate because its performance relies on how well it can sort the reactions to reduce its average search depth and how many propensity recalculations it can eliminate using the dependency graph. The 15 gene model performs extremely well using the Sorting Direct Method because the average search depth remains fairly small and relatively few propensity updates are required for each reaction. The diffusion model requires over half of the propensities in the model to be updated for each reaction executed. The diffusion model also maintains a relatively high average search depth throughout the run because each reaction has relatively the same propensity value. These attributes cause the diffusion model to run almost five times slower in software than the 15 gene model and cause the 5.8X performance gain by the hardware.

4.7. Multiple Stochastic Simulation Cores on an FPGA

Looking again at Table 3, we notice that the 24-bit, 8 reaction design used to simulate the original single gene transcription-translation model from section 4.6 consumes only 10% of the chip's slices. It is conceivable that we could implement multiple copies of our stochastic simulator core on the chip and could simulate multiple independent runs simultaneously.

To test this theory, a design is implemented that uses the eight of the 24-bit eight reaction cores, each using different random seeds as to produce different time evolutions. This new multi-core design uses 9286 slices, which consumes 67% of the available resources on the XCV2P30 FPGA and can operate at a maximum frequency of 62.1 MHz. Prototyping this design produces identical simulation rate results to the single-core system for each independent simulation run, 4.9 million reactions per second, but since we are able to perform 8 independent simulations in parallel, the multi-core design is able to generate a total of 39.2 million reactions per second. This is a 14.4X speedup over the single processor. In the next chapter, we will show how we can achieve similar speedups by using multiple processors in a cluster.

4.8 Conclusion

Prior work in hardware acceleration of stochastic simulation has provided insights into how we may replace software simulation with hardware accelerated simulation, but have not sufficiently addressed the issue of re-synthesis for each model. The approach shown in this chapter provides a framework for allowing the chip to be reconfigured to simulate multiple models without re-synthesis. A direct comparison of a hardware prototype of the system shows a range of 1.4X to 5.8X performance improvement over a software simulator for a single simulation. By implementing multiple cores on an FPGA or using larger and faster FPGA technologies, it may be possible to achieve larger performance improvements.

CHAPTER 5 - Accelerating Sets of Simulations

This chapter is a revised version of a paper published in the journal *Simulation* in 2004 by James McCollum, Chris Cox, Mike Simpson, and Greg Peterson:

McCollum JM, Cox CD, Simpson ML, Peterson GD, "Accelerating gene regulatory network modeling using grid based simulation." *Simulation* 80 (4-5): 231-241, 2004.

My use of "we" refers to my co-authors and myself. My primary contributions to this paper include (1) a majority of the writing and (2) the implementation and performance analysis of the distributed computing environment.

5.1. Introduction

Many times, modelers need to execute sets of simulations to sufficiently analyze their biochemical models. Suppose one was attempting to determine the evolving probability distribution of the population of a particular species. The modeler could execute thousands of simulations with different random seed values and combine the outputs of these runs to build the probability distributions for the desired species at each time point. Another modeler may wish to determine the sensitivity of the parameter values by running different variations of parameters and observing the changes in the output, requiring multiple simulations for each desired parameter value. Another modeler may wish to match experimental data to the simulation output of their model through parameter optimization techniques like simulated annealing or genetic algorithms. These techniques require numerous independent simulation runs to be effective.

Previous chapters have addressed improving the performance of a single stochastic simulation run for a particular biochemical model. This chapter addresses the problem of accelerating batches of simulations.

A grid-based distributed computing environment is developed that distributes independent simulations and post-processing jobs to a cluster of computational elements. Applying this environment to biochemical network simulation shows a significant reduction in execution time versus running simulation jobs locally. Additionally, a technique is developed for reusing data from previous simulations to eliminate recalculation of unchanged data in related simulations. Combining these techniques with the improvements discussed in previous chapters may provide a viable solution for improving the efficiency of simulating sets of stochastic models.

5.2. The Distributed Computing Environment

In an attempt to create a simple, general-purpose, biologist-friendly platform for executing simulation and post-processing jobs, the Distributed Computing Environment (DCE) is developed. The primarily goals of this system are:

1. To provide a flexible and simple interface for adding functions and tools
2. To provide a simple installation and setup procedure.

The DCE is not limited to biochemical modeling problems; applications specific to biochemical modeling are developed and installed into the DCE to perform parallel simulation and data analysis. The DCE could be used for any modeling, simulation, and post-processing problem if the proper modeling software is written and installed into the DCE. Applications associated with the DCE can be classified into one of the four groups described in Figure 38 and the data flow between these groups is illustrated in Figure 39.

In an attempt to make the DCE as flexible as possible, the system is designed to be platform independent. To achieve this, each application interface is implemented in Java and communicates using platform-independent Java sockets. Application developers who wish to use a simulator or data analysis tool that is not written in Java can access their tool through the Java native interface (JNI) or by making direct system calls to run their tool as a child process.

To simplify data exchange between elements of the DCE, as well as the process of integrating a non-Java tool, all data within the system are stored and passed as files. This simplifies the software interface by not requiring data to be classified into types or requiring conversions to be performed from Java classes to other languages. A client can add data in any format to the *server* as long as the *worker* knows how to handle these data. This feature can improve performance when handling large amounts of data because instead of storing large messages in memory, the data are stored on the disk and can be accessed randomly.

The DCE programming interface allows application developers to quickly integrate their simulator or data processing tool into the DCE system. The programmer first creates an object derived from the `WorkerCommand` class. This class requires the programmer to derive the following four functions:

```
public abstract String getCommandName()
public abstract int getInputCount()
public abstract int getOutputCount()
public abstract void run(Worker worker, File inputs[], File
outputs[])
```

The `getCommandName`, `getInputCount`, and `getOutputCount` functions return the name of the command, the number of input files required to execute the

command, and the number of output files returned by the command. The `run` command executes the simulator or post-processing tool on the files contained in the `inputs` array. Output data are written to the empty files stored in the `outputs` array. By calling `worker.reportProgress()`, the `run` command can periodically report how much of the task has been completed. If an exception is thrown from within this function, an error message is sent to the server and is eventually passed to the corresponding client who requested the job. Exception handling allows the job request to fail gracefully and prevents the worker from prematurely disconnecting from the server. To complete the worker, the application developer creates an executable class that makes three simple function calls that connect to the server, register `WorkerCommand` objects, and request work from the server.

To call the functions implemented in the workers, the application developer must develop a corresponding client application to act as a run manager. The following commands are used to implement a client:

```
public ServerConnection(String host, int port, String type)
public FileId addFile(File file)
public JobId addJob(String command, FileId[] inputs,
                    int outputCount)
public Job[] getJob(JobId jobId[])
public void getFile(FileId fileId, File file)
```

The client constructs a `ServerConnection` object that connects the client to the server. The client then adds data with the `addFile` function and calls the `addJob` function to add work to the server. After work has been added to the server, the client enters a loop that makes periodic calls to the `getJob` function. This function returns a data structure containing information reflecting the current status of the job (Pending, Running, Complete, Error), the percentage of the job that has been completed, the output `FileId`s, and a string representing the error message if the job has completed with an error. If the job completes successfully, the client calls the `getFile` function to retrieve the job's outputs.

5.3. Comparison to Other Grid Management Software

A significant portion of the text reviewing grid management software in this section was contributed by Dr. Gregory D. Peterson, Department of Electrical and Computer Engineering, University of Tennessee – Knoxville, and was used with his permission.

The DCE provides capabilities similar to existing grid computing middleware tools but with some important distinctions, which are described here.

In the NetSolve system (Casanova, 1997), client applications communicate computational requests to one of a set of agents. Each agent maintains status information

on a collection of servers, including the specific functions they support, benchmarking results to indicate their relative performance, and their current loading status. Based on this information, when a client requests resources to compute a function, the agent determines which server is the best match for the client and returns the server information to the client. The client then sends the function identity and parameters to the server for execution. Once complete, the server communicates the results back to the client. Note that after assigning a server to the client, the agent does not participate further in the computation. In contrast to the DCE centralized management of job assignment and parameter/result file exchange, NetSolve employs a centralized job assignment via the agents, but parameters and results are exchanged directly between clients and servers in a distributed manner. The NetSolve system requires servers to register their computational capabilities with agents by submitting a program description file describing the prototype for each function supported. Each server also provides the agent with performance results from executing a benchmark (such as LAPACK) to differentiate server capabilities. Agents may support multiple servers, and each server can be registered with multiple agents. The agents act as "matchmakers" by identifying the most appropriate server to execute a given function for a client.

An important, implicit assumption made by the NetSolve agents in determining the best server to select is that each server contains one or more processors with a von Neumann architecture. In particular, because agents assess server capabilities with a standard benchmark, the agents do not account for servers with custom computation engines, such as hardware acceleration units implemented using reconfigurable computing hardware. In prior work with NetSolve, some investigation has been performed into extending NetSolve to support reconfigurable computing platforms (Lehrter, 2002), but the NetSolve system does not currently provide such capabilities. In contrast, the DCE was developed with the flexibility to support versions of ESS implemented in software only or using reconfigurable computing-based custom accelerators.

A number of job-scheduling systems exist for managing grid resources. One particularly popular system is Condor (Berman, 2003). Condor supports serial or parallel applications executing on a collection of machines in an attempt to exploit the idle cycles of these machines. Condor's usage policy on desktop machines is to not interfere with users at the console. Because it periodically checkpoints applications, Condor will stop jobs and migrate them to another machine when mouse or keyboard activity indicates that the first machine is no longer idle. As with NetSolve, Condor does not support reconfigurable computing-based hardware acceleration engines.

The Globus toolkit (Foster, 1999) employs a flexible, layered architecture to provide higher level services built on core services. For resource management, a set of local resource managers, known as Globus resource allocation managers (GRAMs), each provides access to their local processors. For example, a GRAM could interface with Condor to use a collection of workstations. In addition, Globus provides a suite of other services, such as security and authentication, access to system status, and communications. In principal, a GRAM could interface with the DCE to access a set of

applications as well, although this integration into Globus has not yet been implemented. As mentioned with Condor above, none of the resource managers currently supported with Condor provides support for reconfigurable computing-based hardware acceleration.

Because the DCE grid infrastructure, particularly as implemented in the BioGrid tool, attempts to provide a simple interface to help biological researchers perform computational tasks, the programming interface is much less flexible than what is provided with parallel programming libraries such as PVM (Geist, 1994) or MPI (Snir, 1998). Application developers may choose to employ these communications facilities to create parallel processing workers, but the intent of DCE is to hide such details from users.

5.4. Applying the DCE to Biochemical Modeling

To accelerate the simulation of sets of biochemical models, workers and clients are implemented that allow biologists to run multiple-simulation jobs over a network of machines. This enables the efficient sweeping of lists of parameter and random seed values using the DCE.

To evaluate the performance of the DCE, speedups and efficiency are evaluated for a typical problem. As a baseline, 100 simulation jobs of a simple *Vibrio fischeri* quorum sensing model are executed sequentially without using the DCE on a lightly loaded Sun Ultra-60 workstation running Sun OS v5.8. The total execution time for this run was 11,222.40 seconds, or over 3 hours. A series of experiments are performed using the same set of 100 simulation jobs executing on a cluster of up to 14 processors that are of the same specifications as the sequential case. In all of the experiments, the processors were lightly loaded.

The simulation execution times and associated data are given in Table 7. In this table, DCE ideal is the ideal predicted time for the simulation, which would be the baseline time divided by the number of processors used. The DCE actual represents the actual wall clock time measured from the simulation. The speedup is computed by taking the baseline time and dividing it by the DCE actual time. The utilization, which is a measure of how efficiently the multiple processors are being utilized, is the ratio of the measured speedup to the number of processors.

As one can see with these results, the DCE provides quite good results, with nearly ideal speedup and efficiency for the simulations of interest. Note that a biologist using the serial version of the simulator must wait more than 3 hours for the set of simulations to complete, as opposed to less than 14 minutes when using 14 workstations via the DCE. This not only represents an impressive speedup in the simulations but also enables a much more aggressive use of virtual experimentation in understanding biological problems.

5.5. Reusing Simulation Results

Recall the two gene model presented in Chapter 2, which is reprinted here as Figure 40. In this model, one gene is transcribed and translated to produce a protein that regulates the transcription rate of a second gene. If we simulate this model using an exact stochastic simulation algorithm, the results produce a vector of times when we execute each reaction R_i, which we will call $<T(R_i)>$, and a vector of times when we increase or decrease the value of species S_i, which we will call $<T(S_i)>$. Sweeping different values for the rate constants, which would be required to perform sensitivity analysis or optimization, will cause us to run additional stochastic simulations and get different results for $<T(R_i)>$.

Now suppose that we were sweeping a set of values for the rate constant for reaction R_8. Changing the constant will alter $<T(R_8)>$, which will subsequently alter the values of $<T(S_{Protein2})>$, but because $S_{Protein2}$ does not affect the propensity function of any of the other reactions in the system, the remaining reaction occurrence time vectors will not be affected. Because the propensity functions for reactions other than R_8 are independent of $S_{Protein2}$ and the rate constants for these reactions do not change in the subsequent simulations, the estimated reaction times are statistically independent of the rate constant for reaction R_8. We can therefore reuse $<T(R_i)>$ for all reactions other than R_8.

Changing the rate constant for reaction R_5 alters $<T(R_5)>$ which alters $<T(S_{mRNA2})>$. Altering $<T(S_{mRNA2})>$ affects $<T(R_6)>$ and $<T(R_7)>$, which eventually affects $<T(S_{Protein2})>$ and $<T(R_8)>$. The propensity functions for reactions R_1, R_2, R_3, R_4, R_9 and R_{10} all do not depend on S_{mRNA2} or $S_{Protein2}$, thus their reaction occurrence times are statistically independent of the rate constant for reaction R_5 and their reaction execution times can safely be reused for future simulations. Because R_6, R_7, and R_8 all depend on $<T(R_5)>$, subsequent simulations must re-simulate these dependent reactions.

Changing the value of the rate constant for reaction R_1 affects all of the reactions in the model, so none of the results can be reused. The effectiveness of this technique therefore depends on the model structure and the rate constant we choose to vary.

So how do we programmatically determine which results can be reused and which results are important to save for subsequent simulations? We begin by reading our original model and building a directed graph. We create a node for each reaction and a node for each species. For each species S_x that exists in the reactant list of reaction R_y, we add a directed edge from S_x to R_y. This represents the relationship between species and reaction propensity functions, thus if S_x changes, the propensity value for R_y will change. For each reaction R_x whose execution affects the population of species S_y, we add an edge from R_x to S_y. A depiction of this sample graph is shown in Figure 41. We will refer to this graph as the species-reaction relationship graph so that it is not confused with the reaction dependency graph developed by Gibson (2000).

When subsequent models are submitted to the simulator, we mark nodes that contain reactions whose rate constant has changed and nodes that contain species whose initial population has changed. We will refer to these nodes as "changed nodes." By performing reachability analysis on each of these nodes, we can determine the "dependent nodes", which are the nodes that depend on the modified species and reactions. This is accomplished by recursively marking nodes using the pseudo-code given in Figure 42. The nodes containing reactions that affect marked species and nodes that contain reactions that affect species that affect marked reactions are referred to as "helper nodes," because their $<T(R_i)>$ vectors can be used to help regenerate the reaction occurrence times for the changed and dependent nodes. Figure 43 shows an example of the different types of nodes given that we have changed the value of the rate constant for reaction R_5 for the model given in Figure 40.

Now that we can identify which results are reusable, how do we reuse these results within the SSA? We do so through the development of a new simulation algorithm which we will refer to as the Partitioning Sorting Direct Method (PSDM), outlined in Figure 44.

The PSDM algorithm takes an initial input model and simulates it using the Sorting Direct Method, recording the values of $<T(S_i)>$ and $<T(R_i)>$. For subsequently submitted models, we identify the changed, dependent, and helper nodes using the procedure defined in section 5.5. We then use the PSDM to generate $<T(S_i)>$ and $<T(R_i)>$ for the changed and dependent nodes.

The SDM simulator is configured to simulate the reactions stored in the changed and dependent nodes. We use the simulator to generate potential reaction occurrence times for the changed and dependent reactions and compare the generated times to the reaction occurrence times $<T(R_i)>$ of the helper nodes. If a helper reaction occurs earlier than the newly generated reaction time, we execute the helper reaction and scale the newly generated reaction time using the same procedure that was used in the NRM for scaling potential reaction occurrence times. Otherwise we execute the changed or dependent reaction in a manner similar to the SDM. The helper reaction times are stored in an indexed priority queue data structure to accelerate the search time for the earliest reaction.

5.6. Performance Analysis

A version of the Partitioning Sorting Direct Method is implemented and is tested using the same lightly-loaded dual-processor system used in Chapter 2. Source code for the implementation is given in the Appendix. Executing a single simulation of the two gene model given in Figure 40 for 40,000 simulated seconds takes the SDM 22.1 seconds and the PSDM 26.5 seconds. The additional overhead is introduced by the PSDM because it must save the times of each individual reaction event to disk.

If we execute five simulations and increase the reaction rate constant of R_8 by 0.1 each time, the SDM requires a total of 110.4 seconds to perform all of the simulations in the set and the PSDM requires only 92.1 seconds (the first PSDM simulation run must generate entirely original results). This performance gain will grow as we increase the number of subsequent model files in the set and as we increase the length of the simulation.

Performing the same test and modifying the reaction rate constant of R_5 allows the SDM to complete the five simulations in 104.3 seconds, while the PSDM requires 175.9 seconds. The PSDM performs poorly when the reaction rate constant of R_5 is changed because it still must process a majority of the reactions in the system resulting in little savings from reusing results and large overhead from reading reactions from files.

If we replicate the two gene model ten times to produce the twenty gene model used in Chapter 2, the PSDM results in a significant performance gain because a majority of the reactions can be reused on subsequent simulations. Modifying one of the ten copies of the R_5 reaction by 0.1 for five simulations and running for 5,000 simulated seconds allows the SDM to complete the first simulation in 28.6 seconds and the entire simulation set in 141.9 seconds. The PSDM requires 34.1 seconds to simulate the first simulation and an average of 3.3 seconds to complete each subsequent simulation, resulting in a total of 47.3 seconds to simulate the entire set.

Clearly, the performance of the PSDM is closely tied to the number of reactions it can eliminate in future simulations, which depends entirely on the structure of the model and the rate constants that are being varied. The PSDM will generally achieve its best performance on parameter sweeps of node values that affect a relatively small chain of cascading reaction propensities and models with sparse dependency graphs that have few cyclic dependencies.

The performance of the PSDM will also depend on the speed at which data can be written to and read from disk. Long simulations may result in the buildup of extremely large reaction occurrence time data files.

5.7. Conclusion

When performing a set of simulations, it is often possible to improve performance by distributing independent simulation jobs to a cluster of computers. Furthermore, when simulating particular model structures that allow a majority of the simulation results to be reused, significant performance gains can also be realized. Combining these techniques with those defined in the previous chapters may provide a viable solution for accelerating sets of stochastic simulations.

Furthermore, a traditional problem in biochemical modeling is that most of the parameter values for biological models are unknown. One approach to solving this problem is to

use optimization heuristics like simulated annealing or genetic algorithms to fit model simulation results to experimental data. To execute such a system, extremely large sets of simulations must be executed. Applying the techniques discussed in this chapter for accelerating sets of simulations may help to enable automated parameter fitting for biochemical models.

CHAPTER 6 – Summary and Conclusions

The following list summarizes the major points, ideas, and accomplishments of this work.

- Creating a virtual laboratory where biologists can perform experiments *in silico* instead of *in vitro* or *in vivo* would serve to enhance the microbiological and pharmaceutical discovery process.

- To accurately model biochemical systems affected by noise, we can use a stochastic formulation that represents chemical species as discrete-valued populations and reactions as random processes.

- The poor computational performance of exact stochastic simulation algorithms limits the applicability of the stochastic formulation.

- By adding an approximate sorting data structure to the Optimized Direct Method, we create the Sorting Direct Method which properly handles transient propensity shifts and eliminates the need for pre-simulations.

- Performance analysis shows that the Sorting Direct Method is the fastest known serial exact stochastic simulation algorithm.

- When attempting to find long-term steady-state probability distributions, using the Un-timed Sorting Direct Method we can further increase simulation efficiency by eliminating the generation of reaction times.

- In an attempt to accelerate exact stochastic simulation on supercomputers and multi-processor computers, the Parallel Reaction Method is developed and is the only known parallel exact stochastic simulation algorithm.

- The performance of the Parallel Reaction Method is highly dependent on model structure. Models that can be partitioned in such a way that allows for minimum communication and sufficient load balancing can show a significant performance gain.

- Using 32 processors on a parallel shared memory supercomputer, the Parallel Reaction Method demonstrates a 5X performance improvement over serial exact stochastic simulation algorithms when executing a large-scale model of *E. coli*.

- A hardware accelerated stochastic simulation is implemented that does not require the modeler to re-synthesize their design to simulate different models.

- Handling species updates serially allows us to eliminate multiplication from propensity calculation and makes the hardware design more scalable.

- Using the hardware accelerated design, a performance gain ranging from 1.4X to 5.8X versus running the Sorting Direct Method can be achieved. Further performance can be gained by adding multiple cores to the hardware design.

- A flexible distributed computing environment is developed to accelerate parameter sweeping, parameter optimization, sensitivity analysis, and data post-processing on a grid or cluster of computers.

- The distributed computing environment demonstrates a significant performance improvement over simulating jobs locally.

- In certain cases, we can use the Partitioning Sorting Direct Method to analyze the model structure and allow the user to reuse simulation results, saving significant computation time.

So how do we use all of these tools to accelerate stochastic simulation? This largely depends on the modeling problem and the structure of the model.

If the structure of the model is loosely coupled meaning that the execution of a most reaction events trigger relatively few propensity updates to other reactions, the Parallel Reaction Method may be employed because the communication required between processors should be relatively small. If the structure of the model is tightly coupled, a hardware accelerated approach may be the best option because the hardware can update all of the propensities simultaneously, saving time when compared to the parallel and serial methods.

If the modeling problem requires one extremely long simulation run where many reaction events are executed, employing the Parallel Reaction Method is a good option. For modeling problems where sets of simulations are required, using the DCE and/or a multi-core hardware accelerated approach would be efficient. If the modeling problem can be partitioned and a majority of the simulation results can be reused, the Partitioning Sorting Direct Method should be employed. Users only concerned with long-term steady-state probability distributions should utilize the Un-timed Sorting Direct Method, which could be combined with the DCE to execute jobs in parallel.

Now that we have an understanding of how the performance of the simulator depends on both the model structure and the computational architecture selected to simulate the model, we could now develop a system that would automate the selection of a computational system for simulating a particular problem. This system would ask the user for a model and a set of analyses to be executed and would automatically route the job to the particular computational architecture that would be best suited for the project. This system could also possibly apply multiple techniques that have been discussed in this work, such as using running a parallel simulation that uses multiple FPGAs instead of processors. Such a system would allow the modeler to spend more time modeling and

less time attempting to choose the correct computational architecture for their modeling problem.

The applicability of each of these techniques is highly dependent on model structure, the computational resources available to the modeler, and the specific information that is to be extracted from the model. Using advanced computational architectures and novel algorithm enhancements, we can significantly improve the performance of exact stochastic simulation.

LIST OF REFERENCES

LIST OF REFERENCES

Adalsteinsson D, McMillen D, Elston TC, "Biochemical Network Stochastic Simulator (BioNetS): software for stochastic modeling of biochemical networks." *BMC Bioinformatics* 5: Art. No. 24, 2004

Arjunan SNV, Takahashi K, Tomita M, "Shared-memory multiprocessing of Gillespie's stochastic simulation algorithm in E-Cell 3." *Proceedings of the Fourth International Conference on Systems Biology*, 2003.

Arkin A, Ross J, McAdams HH, "Stochastic kinetic analysis of developmental pathway bifurcation in phage λ-infected Escherichia coli cells." *Genetics* 149: 1633-1648, 1998.

Berman F, Fox G, Hey AJG, *Grid computing: Making the global infrastructure a reality*. New York: John Wiley, 2003.

Burrage K, Tian T, Burrage P, "A multi-scaled approach for simulating chemical reaction systems." *Progress in Biophysics and Molecular Biology* 85, 217-234, 2003.

Cao Y, Li H, Petzold LR, "Efficient formulation of the stochastic simulation algorithm for chemically reacting systems." *Journal of Chemical Physics* 121: 4059-4067, 2004.

Cao Y, Gillespie DT, Petzold LR, "The slow-scale stochastic simulation algorithm." *Journal of Chemical Physics* 122: 014116, 2005.

Casanova H, Dongarra J, "NetSolve: A network-enabled server for solving computational science problems." *International Journal of Supercomputer Applications and High Performance Computing* 11 (3): 212-23, 1997.

Chickarmane V, Paladugu SR, Bergmann F, Sauro HM, "Bifurcation discovery tool." *Bioinformatics* 21 (18): 3688-3690, 2005.

Cox CD, Peterson GD, Allen MS, Lancaster JM, McCollum JM, Austin D, Yan L, Sayler GS, Simpson ML, "Analysis of noise in quorum sensing." *OMICS* 9: 317-334, 2003.

Cox CD, McCollum JM, Austin DW, Allen MS, Dar RD, Simpson ML, "Frequency domain analysis of noise in simple gene circuits," *Chaos* 16: 026102, 2006.

Endy D, Brent R, "Modeling cellular behavior." *Nature* 409: 391-395, 2001.

Foster I, Kesselman C, *The Grid: Blueprint for a new computing infrastructure.* NewYork: Morgan Kaufmann, 1999.

Fujimoto RM, "Parallel discrete event simulation." *Communication to the ACM* 33: 30-53, 1990.

Geist A, Beguelin A, Dongarra J, Jiang W, Manchek R, Sunderam VS, *PVM: Parallel Virtual Machine*. MIT Press, Cambridge, 1994.

Gibson M, Bruck J, "Efficient exact stochastic simulation of chemical systems with many species and many channels." *Journal of Physical Chemistry A* 104: 1876-1889, 2000.

Gillespie DT, "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions." *Journal of Computational Physics* 22: 403-434, 1976.

Gillespie DT, "Exact stochastic simulation of coupled chemical reactions." *Journal of Physical Chemistry* 81: 2340-2361, 1977.

Gillespie DT, "A rigorous derivation of the chemical master equation." *Physica A* 188: 404-425, 1992.

Gillespie DT, "The Chemical Langevin Equation." *Journal of Chemical Physics* 113: 297-306, 2000.

Gillespie DT, " Approximate accelerated stochastic simulation of chemically reacting systems." *Journal of Chemical Physics* 115: 1716-1733, 2001.

Gillespie DT, Petzold LR, "Improved leap-size selection for accelerated stochastic simulation." *Journal of Chemical Physics* 119: 8229-8234, 2003.

Greenberg EP, "Acyl-homoserine lactone quorum sensing in bacteria." *Journal of Microbiology* 38 (3): 117-121, 2000.

Hindmarsh A., "A Brief Description of ODEPACK – A systematized collection of ODE solvers." http://www.netlib.org/odepack/opkd-sum, 2003.

Hucka M, Finney A, Sauro HM, Bolouri H, Doyle JC, Kitano H, Arkin AP, Bornstein BJ, Bray D, Cornish-Bowden A, Cuellar AA, Dronov S, Gilles ED, Ginkel M, Gor V, Goryanin II, Hedley WJ, Hodgman TC, Hofmeyr JH, Kummer U, Le Novere N, Loew LM, Lucio D, Mendes P, Minch E, Mjolsness ED, Nakayama Y, Nelson MR, Nielson PF, Sakurada T, Schaff JC, Shapiro BE, Shimizu TS, Spence HD, Stelling J, Takahashi K, Tomita M, Wagner J, Wang J, "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models." *Bioinformatics* 19 (4): 524-531, 2003.

Jefferson DR, "Virtual time." *ACM Transactions on Programming Languages and Systems* 7: 404-425, 1985.

Kaern M, Elston TC, Blake WJ, Collins JJ, "Stochasticity in gene expression: from theories to phenotypes." *Nature Reviews Genetics* 6: 451-464, 2005.

Kashtan N, Itzkovitz S, Milo R, Alon U, "Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs." *Bioinformatics* 20 (11): 1746-1758, 2004.

Keane J, Bradley C, Ebeling C, "A compiled accelerator for biological cell signaling simulations." *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*: 233-241, 2004.

Kot M, *Elements of Mathematical Ecology*. Cambridge University Press, Cambridge, 2001.

Kurata H, El-Samad H, Yi TM, Khammash MH, Doyle J, "Feedback regulation of the heat shock response in E. coli." *Proceedings of the IEEE Conference on Decision and Control* 837-842, 2001.

Lee T, Ghosh S, "Simulating asynchronous, decentralized military command and control." *IEEE Computational Science and Engineering* 3: 69-79, 1996.

Lehrter JM, Abu-Khzam FN, Bouldin DW, Langston MA, Peterson GD, "On special-purpose hardware clusters for high-performance computational grids." *Proceedings of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems*, 2002.

Li Y, Qian DP, Zhang WJ, "A practical approach for constructing a parallel network simulator." *Lecture Notes in Computer Science* 2869: 739-746, 2003.

Lloyd CM, Halstead MD, Nielsen PF, "CellML: its future, present and past." *Progress in Biophysics and Molecular Biology* 85 (2-3): 433-450, 2004.

Lok L, "The need for speed in stochastic simulation." *Nature Biotechnology* 22 (8): 964-965, 2004.

Lupp C, Urbanowski M, Greenberg EP, Ruby EG, "The Vibrio fischeri quorum-sensing systems ain and lux sequentially induce luminescence gene expression and are important for persistence in the squid host." *Molecular Microbiology* 50 (1): 319-331, 2003.

Martin DE, Radhakrishnan R, Rao DM, Chetlur M, Subramani K, Wilsey P, "Analysis and simulation of mixed-technology VLSI systems." *Journal of Parallel and Distributed Computing* 62: 468-493, 2002.

Mascagni, M, Srinivasan, A, "Algorithm 806: SPRNG: a scalable library for pseudorandom number generation." *ACM Transactions on Mathematical Software* 26 (3): 436-461, 2000.

McCollum JM, Cox CD, Simpson ML, Peterson GD, "Accelerating gene regulatory network modeling using grid based simulation." *Simulation* 80 (4-5): 231-241, 2004.

McCollum JM, Peterson GD, Cox CD, Simpson ML, Samatova NF, "The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior." *Journal of Computational Biology and Chemistry* 30 (1): 39-49, 2006.

Ptashne M, *A genetic switch: lambda phage and higher organisms*, Second Edition, Blackwell Scientific Publications: Cambridge, MA, 1992.

Rao CV, Wolf DM, Arkin AP, "Control, exploitation, and tolerance of intracellular noise." *Nature* 420: 231-237, 2002..

Rao CV, Arkin AP, "Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the Gillespie algorithm." *Journal of Chemical Physics* 118 (11): 4999-5010, 2003.

Rathinam M, Petzold LR, Cao Y, Gillespie DT, "Stiffness in stochastic chemically reacting systems: the implicit tau-leaping method." *Journal of Chemical Physics* 119: 12784-12794, 2003.

Roy R, Arunachalam R, " Parallel discrete event simulation algorithm for manufacturing supply chains." *The Journal of the Operational Research Society* 55: 622-629, 2004.

Salis H, Kaznessis Y, "Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions." *Journal of Chemical Physics* 122: 054103, 2005.

Salwinski L, Eisenberg D, "In silico simulation of biochemical network dynamics." *Nature Biotechnology* 22 (8): 1017-1019, 2004.

Schmidt H, Jirstrand M, "Systems Biology Toolbox for MATLAB: a computational platform for research in systems biology." *Bioinformatics* 22 (4): 514-515, 2006.

Schwehm M, "Parallel stochastic simulation of whole-cell models." *Proceedings of the Second International Conference on Systems Biology*, 2001.

Shampine LF, Reichelt MW, "The MATLAB ODE suite." *SIAM Journal of Scientific Computing* 18: 1-22, 1997.

Shen-Orr SS, Milo R, Mangan S, Alon U, "Network motifs in transcriptional regulation of Escherichia coli." *Nature Genetics* 31: 64-68, 2002.

Simpson ML, Cox CD, Sayler GS, "Frequency domain analysis of noise in autoregulated gene circuits." *Proceedings of the National Academy of Sciences* 100: 4551-4556, 2003.

Simpson ML, Cox CD, Sayler GS, "Frequency domain chemical Langevin analysis of stochasticity in gene transcriptional regulation." *Journal of Theoretical Biology* 229: 383-394, 2004.

Snir M, Otto S, Huss-Lederman S, Walker D, *MPI: The Complete Reference*. MIT Press, Cambridge, 1998.

Strohman R, "Maneuvering in the complex path from genotype to phenotype." *Science* 296 (5568): 701-703, 2002.

Tian T, Burrage K, "Binomial leap methods for simulating chemical kinetics." *Journal of Chemical Physics* 121: 10356-10364, 2004.

Thurmon BP, *Reconfigurable hardware acceleration of exact stochastic simulation*. A thesis presented for the Master of Science in Electrical Engineering from the University of Tennessee – Knoxville, 2005.

Thurmon BP, McCollum JM, Peterson GD, Cox CD, Samatova NF, Sayler GS, and Simpson ML, "Accelerating Exact Stochastic Simulation using Reconfigurable Computing." *Engineering of Reconfigurable Systems and Algorithms Conference (ERSA 2005)*, Las Vegas, NV, 2005

Turner TE, Schnell S, Burrage K, "Stochastic approaches for modeling in vivo reactions." *Computational Biology and Chemistry* 28: 165-178, 2004.

Vass M, Allen N, Shaffer CA, Ramakrishnan N, Watson LT, Tyson JJ, "The JigCell Model Builder and Run Manager." *Bioinformatics* 20 (18): 3680-3681, 2004

Yeger-Lotem E, Sattath S, Kashtan N, Itzkovitz S, Milo R, Pinter RY, Alon U, Margalit H, "Network motifs in integrated cellular networks of transcription-regulation and protein-protein interaction." *Proceedings of the National Academy of Sciences* 101 (16): 5934-5939, 2004.

Yoshimi M, Osana Y, Fukushima T, Amano H, "Stochastic simulation for biochemical reactions on FPGA." *Lecture Notes in Computer Science* 3203: 105-114, 2004.

APPENDIX

Table 1. Simulation Time Comparison.

	Heat Stress Response		Phage-λ		Quorum Sensing	
	Reactions	Time (s)	Reactions	Time (s)	Reactions	Time (s)
First (FRM)	48392805	1065.72	7887143	532.59	6590231	424.63
Direct (DM)	46604186	147.81	6936655	61.38	6649760	62.42
Next (NRM)	46786907	168.57	5156293	87.21	7464846	13.70
Optimized (ODM)	46312042	35.33	6090326	13.08	6545829	7.63

Table 2. Simulator Reaction Rate Comparison.

Simulator	Heat Stress Response	Phage-λ	Quorum Sensing
First (FRM)	45.51	14.81	15.52
Direct (DM)	315.29	113.01	106.53
Next (NRM)	277.55	59/13	544.75
Optimized (ODM)	1310.95	465.45	857.72

Reaction rate is measured in thousands of reactions per second.

Table 3. Hardware Device Utilization.

	16 bit	24 bit	32 bit
8 Reactions	1081 (7%)	1450 (10%)	1796 (13%)
16 Reactions	1898 (13%)	2589 (18%)	3276 (23%)
32 Reactions	3466 (25%)	4819 (35%)	6163 (44%)
64 Reactions	6613 (48%)	9292 (67%)	11961 (87%)

Device utilization is measured in slices used and the percentage of the chip used.

Table 4. Hardware Operating Frequencies.

	16 bit	24 bit	32 bit
8 Reactions	100.593 MHz	75.364 MHz	74.582 MHz
16 Reactions	84.696 MHz	81.031 MHz	70.279 MHz
32 Reactions	78.229 MHz	68.241 MHz	64.973 MHz
64 Reactions	73.828 MHz	66.300 MHz	70.937 MHz

Based on the report from the trace utility.

Table 5. Hardware Execution Rates.

	16 bit	24 bit	32 bit
8 Reactions	9.14 MR/s	6.85 MR/s	6.78 MR/s
16 Reactions	6.52 MR/s	6.23 MR/s	5.41 MR/s
32 Reactions	5.22 MR/s	4.55 MR/s	4.33 MR/s
64 Reactions	4.34 MR/s	3.90 MR/s	4.17 MR/s

Measurements are given in millions of reactions executed per second.
This table assumes an average update count of two.

Table 6. Hardware vs. Software Comparison.

Model	Sorting Direct Rate (MR/s)	Hardware Rate (MR/s)	Hardware Core	Hardware Speedup
Single Gene	2.7	4.8	8-Reaction 24-bit	1.8
15 Gene	2.2	3.1	64-Reaction 32-bit	1.4
Diffusion	0.5	2.9	64-Reaction 32-bit	5.8

Execution rates are given in millions of reactions executed per second (MR/s).

Table 7. DCE Performance Results.

Processors	DCE Ideal (s)	DCE Actual (s)	Speedup	Utilization
2	5611.2	5663.5	1.981531	99.1%
4	2805.6	2818.9	3.981127	99.5%
6	1870.4	1919.7	5.845913	97.4%
8	1402.8	1439.9	7.793875	97.4%
10	1122.24	1139.1	9.851988	98.5%
12	935.2	954.1	11.76229	98.0%
14	801.6	832.5	13.48036	96.3%

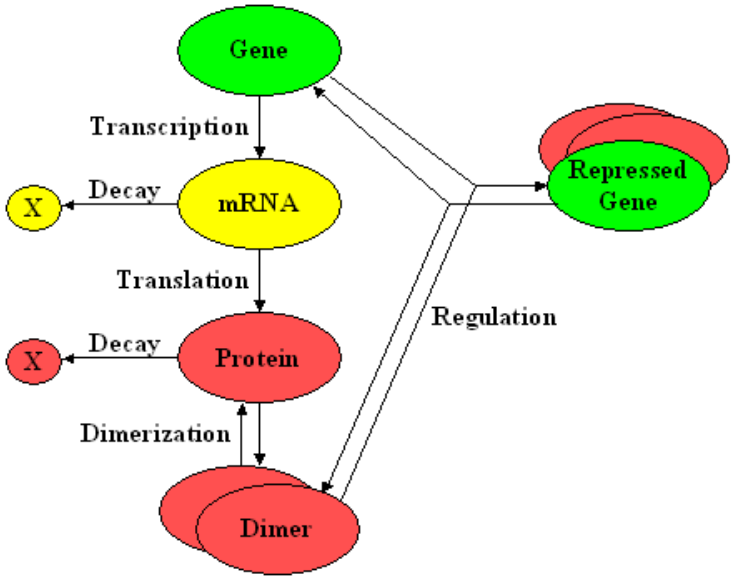


Figure 1. Sample Model of an Autoregulated Biochemical System.

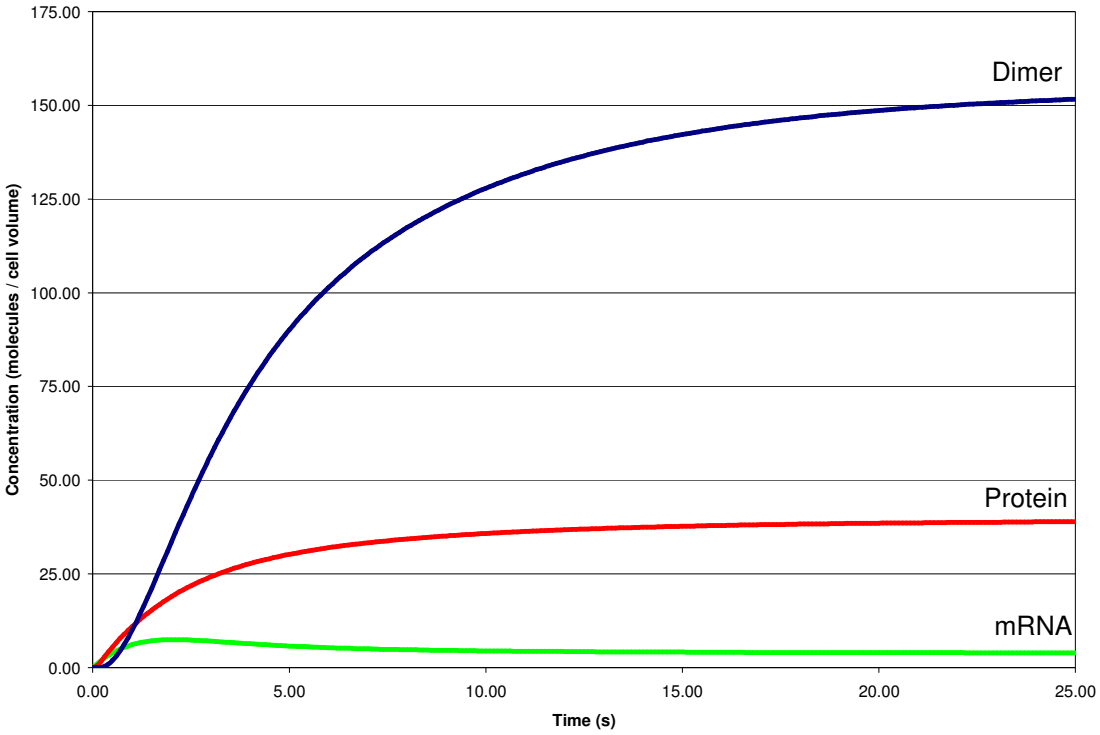


Figure 2. Deterministic Solution of the Autoregulated Gene System.

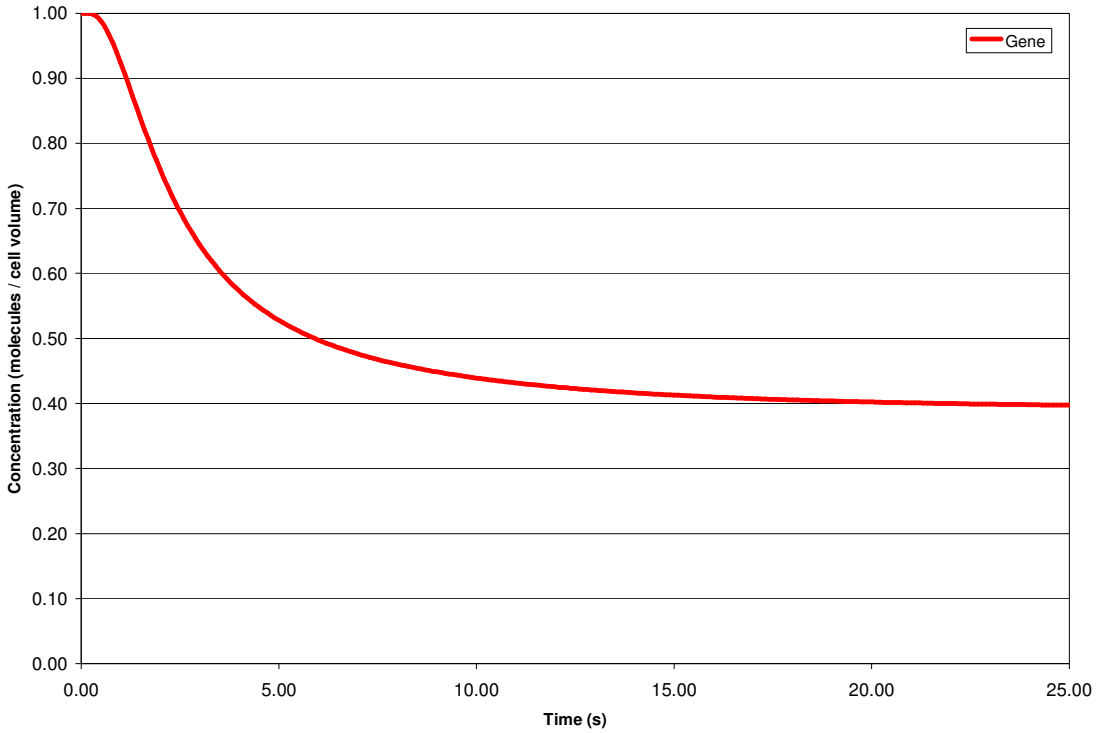


Figure 3. Deterministic Solution of the Gene Concentration.

$G(0) = 1$
$M(0) = 0$
$P(0) = 0$
$D(0) = 0$
$R(0) = 0$

R_1: G -> G + M rate = $k_{tc} = 10$
R_2: M -> * rate = $k_{md} = 1$
R_3: M -> M + P rate = $k_{tl} = 10$
R_4: P -> * rate = $k_{pd} = 1$
R_5: P + P -> D rate = $k_{di} = 1$
R_6: D -> P + P rate = $k_{rdi} = 10$
R_7: G + D -> R rate = $k_{bind} = 0.1$
R_8: R -> G + D rate = $k_{unbind} = 10$

$\langle S \rangle = \{G,M,P,D,R\}$
$\langle R \rangle = \{R_1,R_2,R_3,R_4,R_5,R_6,R_7,R_8\}$
$\langle X(t=0) \rangle = \{1,0,0,0,0\}$

$\langle k \rangle = \{k_{tc},k_{md},k_{tl},k_{pd},k_{di},k_{rdi},k_{bind},k_{unbind}\}$

$\langle r_1 \rangle = \{1,0,0,0,0\}$
$\langle r_2 \rangle = \{0,1,0,0,0\}$
$\langle r_3 \rangle = \{0,1,0,0,0\}$
$\langle r_4 \rangle = \{0,0,1,0,0\}$
$\langle r_5 \rangle = \{0,0,2,0,0\}$
$\langle r_6 \rangle = \{0,0,0,1,0\}$
$\langle r_7 \rangle = \{1,0,0,1,0\}$
$\langle r_8 \rangle = \{0,0,0,0,1\}$

$\langle p_1 \rangle = \{1,1,0,0,0\}$
$\langle p_2 \rangle = \{0,0,0,0,0\}$
$\langle p_3 \rangle = \{0,1,1,0,0\}$
$\langle p_4 \rangle = \{0,0,0,0,0\}$
$\langle p_5 \rangle = \{0,0,0,1,0\}$
$\langle p_6 \rangle = \{0,0,2,0,0\}$
$\langle p_7 \rangle = \{0,0,0,0,1\}$
$\langle p_8 \rangle = \{1,0,0,1,0\}$

Figure 4. Autoregulated Gene Circuit Stochastic Model.

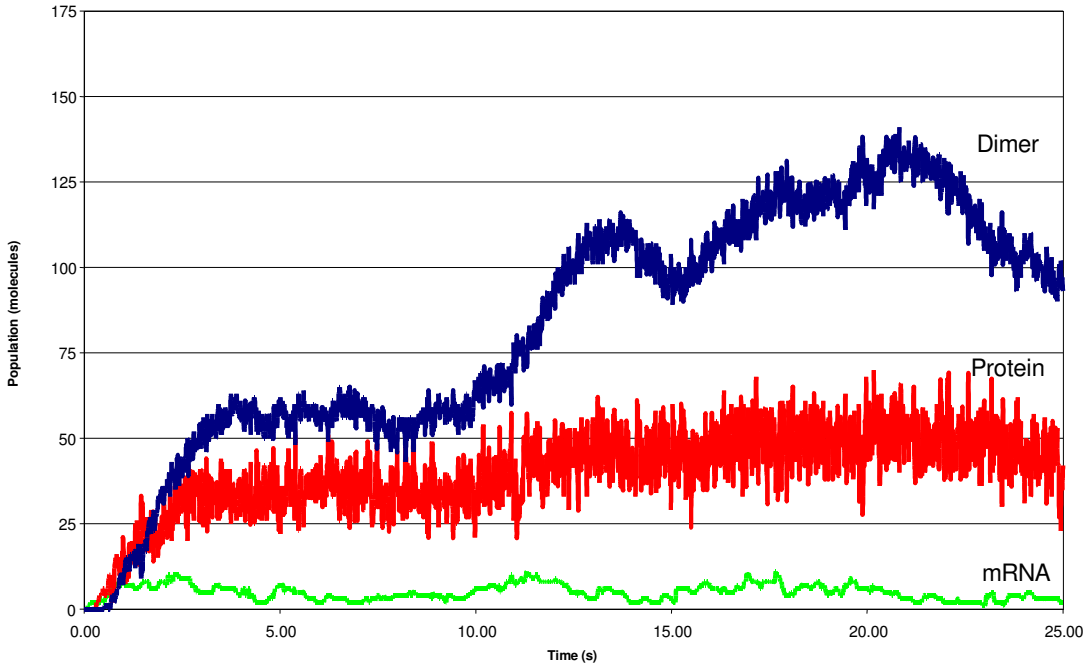


Figure 5. Stochastic Simulation of the Autoregulated Gene Circuit.

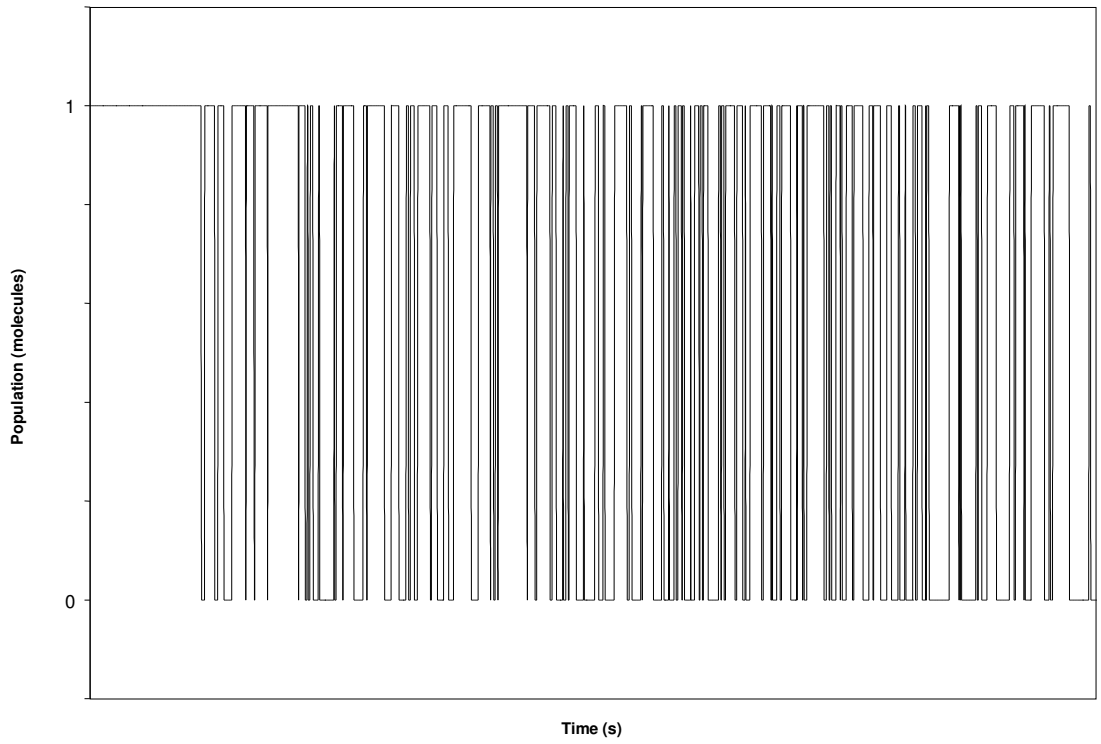


Figure 6. Stochastic Simulation of the Gene Population.

	First Reaction	Direct	Next Reaction	Optimized Direct
1. Initialization	Read the model.	Read the model.	Read the model and generate DG[1] and IPQ[2].	Read the model, generate DG[1], pre-simulate, and reorder the reactions.
2. Propensity Calculation	Calculate propensities for all reactions.	Calculate propensities for all reactions.	Calculate only affected propensities using DG[1].	Calculate only affected propensities using DG[1].
3. Putative Time Generation	Generate putative times for each reaction	Sum the propensities and generate a putative time for the system.	Generate a new putative time for the last reaction executed and update IPQ[2].	Update the total propensity using DG[1] and generate a putative time for the system.
4. Reaction Selection	Find the minimum putative time using a linear search.	Generate a scaled uniform random number and search the propensities using a linear search[3].	Get the minimum putative time, stored at the head of the IPQ[2].	Generate a scaled uniform random number and search the propensities using a linear search[3].

Figure 7. Comparison of Stochastic Simulation Algorithms. [1]DG = Dependency Graph. [2]IPQ = Indexed Priority Queue. [3]Because the Optimized Direct Method reorders the reactions following the pre-simulation at startup, the search depth required by the reaction selection stage will typically be less than the search depth of the Direct Method.

`CurrentTime = 0.0`
`X[1..M] = Initial Species Populations`
`R[1..N] = Reactions`
	1. Initialization

`For I = 1..N`
`  Prop[I] = CalcPropensity(X,R[I])`
`End For`
	2. Propensity Calculation

`For I = 1..N`
`  T[I] = -ln(rand())/Prop[I]`
`End For`
	3. Putative Time Calculation

`SelRxn = 1`
`MinTime = T[1]`
`For I = 2..N`
`  If (MinTime > T[I])`
`  SelRxn = I`
`  MinTime = T[I]`
`  End If`
`End For`
	4. Reaction Selection

`X = X - R[SelRxn].reactants + R[SelRxn].products` `CurrentTime = CurrentTime + MinTime`	5. Reaction Execution

`If (CurrentTime < EndTime)`
`  Goto Propensity Calculation`
`End If`
	6. Termination

Figure 8. Pseudo-code for the First Reaction Method.

CurrentTime = 0.0 X[1..M] = Initial Species Populations R[1..N] = Reactions	1. Initialization

TotalPropensity = 0.0
For I = 1..N
  Prop[I] = CalcPropensity(X,R[I])
  TotalPropensity = TotalPropensity + Prop[I]
End For
	2. Propensity Calculation

T = -ln(rand())/TotalPropensity	3. Putative Time Calculation

Selector = TotalPropensity * rand()
For I = 1..N
  Selector = Selector - Prop[I]
  If (Selector <= 0)
  SelRxn = I
  Break
  End If
End For
	4. Reaction Selection

X = X - R[SelRxn].reactants + R[SelRxn].products CurrentTime = CurrentTime + T	5. Reaction Execution

If (CurrentTime < EndTime)
  Goto Propensity Calculation
End If
	6. Termination

Figure 9. Pseudo-code for the Direct Method.

Name	Reaction	Depends on	Affects	Update
R1	G -> G + M	G	M	R2,R3
R2	M -> *	M	M	R2,R3
R3	M -> M + P	M	P	R4,R5
R4	P -> *	P	P	R4,R5
R5	P + P -> D	P	P,D	R4,R5,R6,R7
R6	D -> P + P	D	P,D	R4,R5,R6,R7
R7	D + G -> R	D,G	D,G,R	R1,R6,R7,R8
R8	R -> G + D	R	D,G,R	R1,R6,R7,R8

Figure 10. Sample Dependency Graph.

CurrentTime = 0.0
X[1..M] = Initial Species Populations
R[1..N] = Reactions
Dependencies = InitDependencyGraph(R)
For I = 1..N
  Prop[I] = CalcPropensity(X,R[I])
  T[I] = -ln(rand())/Prop[I]
  MinHeap.add(I,T[I])
End For
Goto Reaction Selection
	1. Initialization

DependentRxns = Dependencies[SelRxn]
Foreach DepRxn in DependentRxns
  OldProp[DepRxn] = Prop[DepRxn]
  Prop[DepRxn] = CalcPropensity(X,R[DepRxn])
End Foreach
	2. Propensity Calculation

Foreach DepRxn in DependentRxns
  NewT = OldProp[DepRxn] * (T[DepRxn] – CurrentTime)
  NewT = NewT / Prop[DepRxn] + CurrentTime
  T[DepRxn] = NewT
  MinHeap.update(DepRxn,T[DepRxn])
End Foreach
T[SelRxn] = -ln(rand())/Prop[SelRxn]
MinHeap.update(SelRxn,T[SelRxn])
	3. Putative Time Calculation

SelRxn = MinHeap.getMinIndex()	4. Reaction Selection
X = X – R[SelRxn].reactants + R[SelRxn].products CurrentTime = T[SelRxn]	5. Reaction Execution

If (CurrentTime < EndTime)
  Goto Propensity Calculation
End If
	6. Termination

Figure 11. Pseudo-code for the Next Reaction Method.

```
Selector = TotalPropensity * rand()
For I = 1..N
  Selector = Selector - Prop[I]
  If (Selector <= 0)
    SelRxn = I
    Break
  End If
End For
```

Figure 12. Direct Method Reaction Selection.

Simulate the system for a period of time < EndTime
  and monitor the number of times each reaction is
  executed
Sort the reactions, placing the most frequently
  executed reaction at R[1] and the least at R[N]
	Presimulation

CurrentTime = 0.0
X[1..M] = Initial Species Populations
R[1..N] = Reactions
Dependencies = InitDependencyGraph(R)
TotalPropensity = 0.0
For I = 1..N
  Prop[I] = CalcPropensity(X,R[I])
  TotalPropensity = TotalPropensity + Prop[I]
End For
Goto Reaction Selection
	1. Initialization

DependentRxns = Dependencies[SelRxn]
Foreach DepRxn in DependentRxns
  TotalPropensity = TotalPropensity – Prop[DepRxn]
  Prop[DepRxn] = CalcPropensity(X,R[DepRxn])
  TotalPropensity = TotalPropensity + Prop[DepRxn]
End Foreach
	2. Propensity Calculation

T = –ln(rand())/TotalPropensity	3. Putative Time Calculation

Selector = TotalPropensity * rand()
For I = 1..N
  Selector = Selector – Prop[I]
  If (Selector <= 0)
  SelRxn = I
  Break
  End If
End For
	4. Reaction Selection

X = X – R[SelRxn].reactants + R[SelRxn].products CurrentTime = CurrentTime + T	5. Reaction Execution

If (CurrentTime < EndTime)
  Goto Propensity Calculation
End If
	6. Termination

Figure 13. Pseudo-code for the Optimized Direct Method.

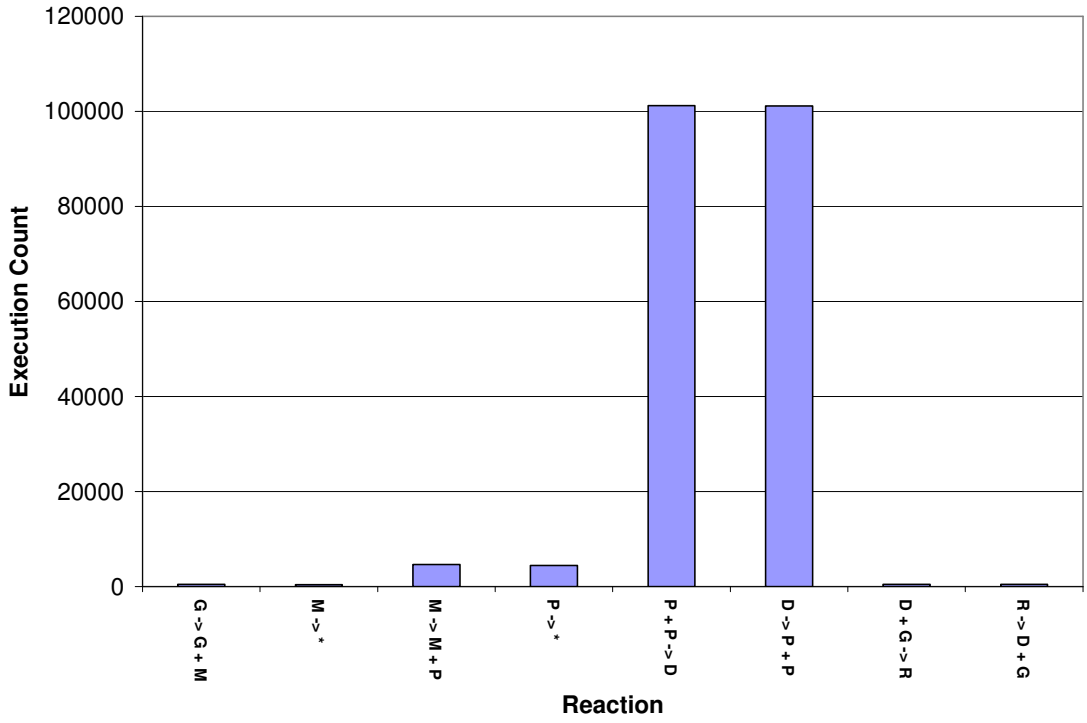


Figure 14. Histogram of Reaction Execution Counts.

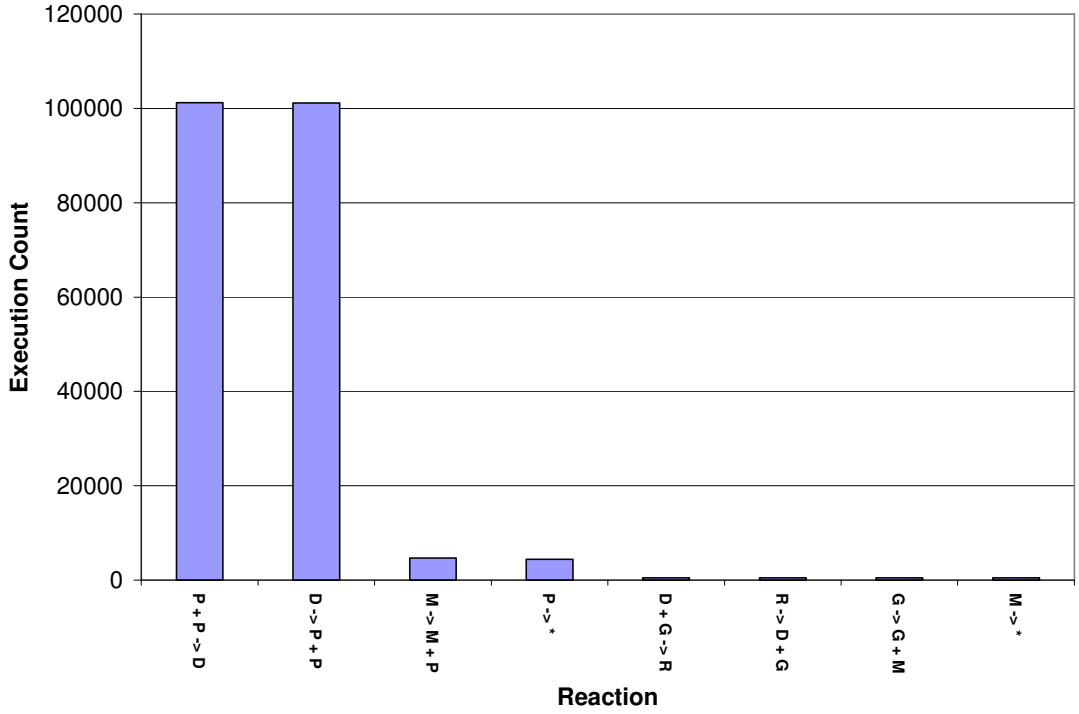
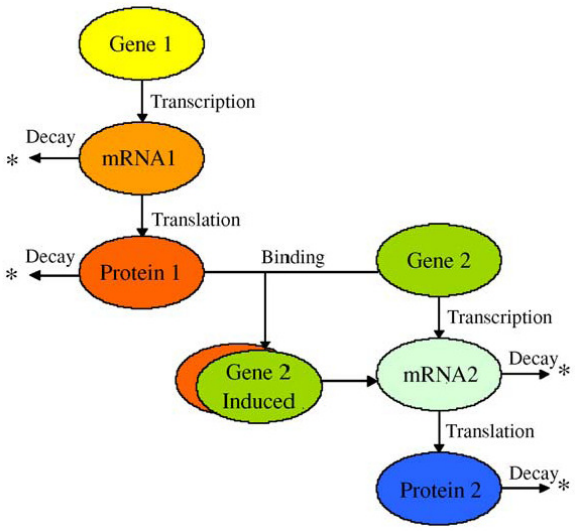


Figure 15. Histogram of Sorted Reaction Execution Counts.




```
GENE1 -> GENE1 + mRNA1             10.0
mRNA1 -> mRNA1 + Protein1          10.0
mRNA1 -> *                          1.0
Protein1 -> *                       1.0

GENE2 -> GENE2 + mRNA2              5.0
mRNA2 -> mRNA2 + Protein2           5.0
mRNA2 -> *                          1.0
Protein2 -> *                       1.0

Gene2 + Protein1 -> Gene2_IND    0.0001
Gene2_IND -> Gene2_IND + mRNA2    100.0
```

Figure 16. Gene Regulation Network Model. This model includes the transcription and translation of Gene1 into Protein1 and Gene2 into Protein2. When Protein1 binds to Gene2, it induces the production of Protein2.

Figure 17. Stochastic Simulation of the Gene Regulation Network Model. Notice that the dynamics of the simulation change just before time 200 when the inducer protein binds to the gene and induces the production of Protein2.

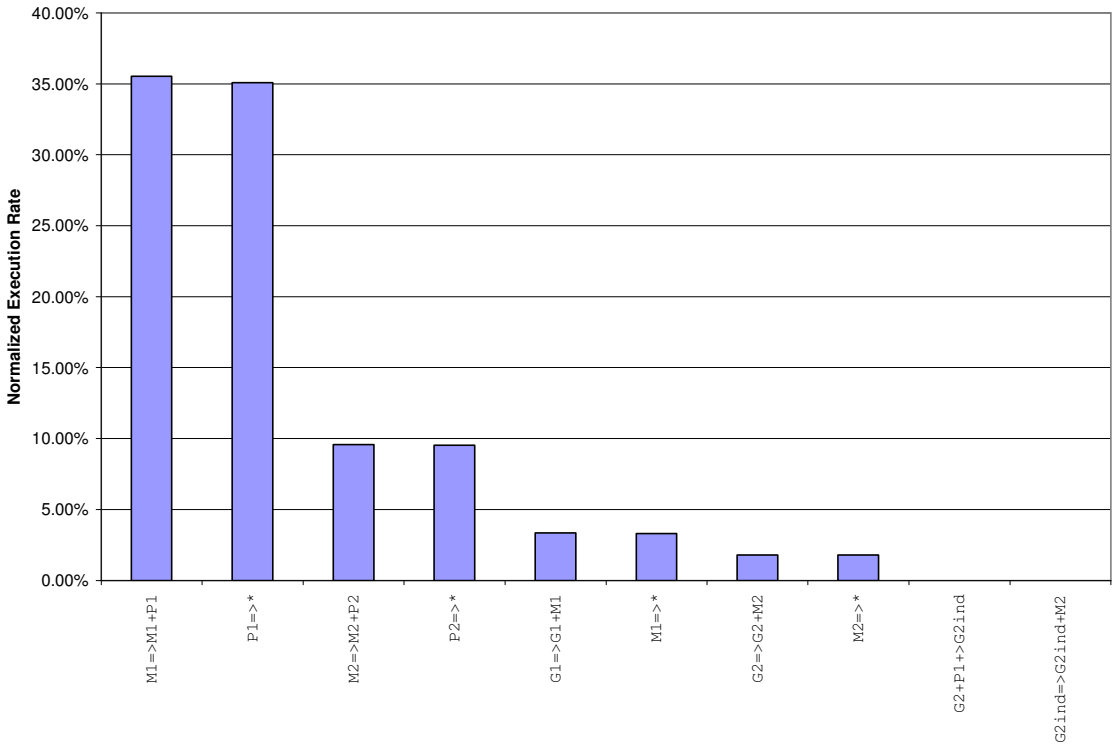


Figure 18. Normalized Reaction Execution Rate Histogram A. This histogram is based on the first 100 seconds of simulated time.

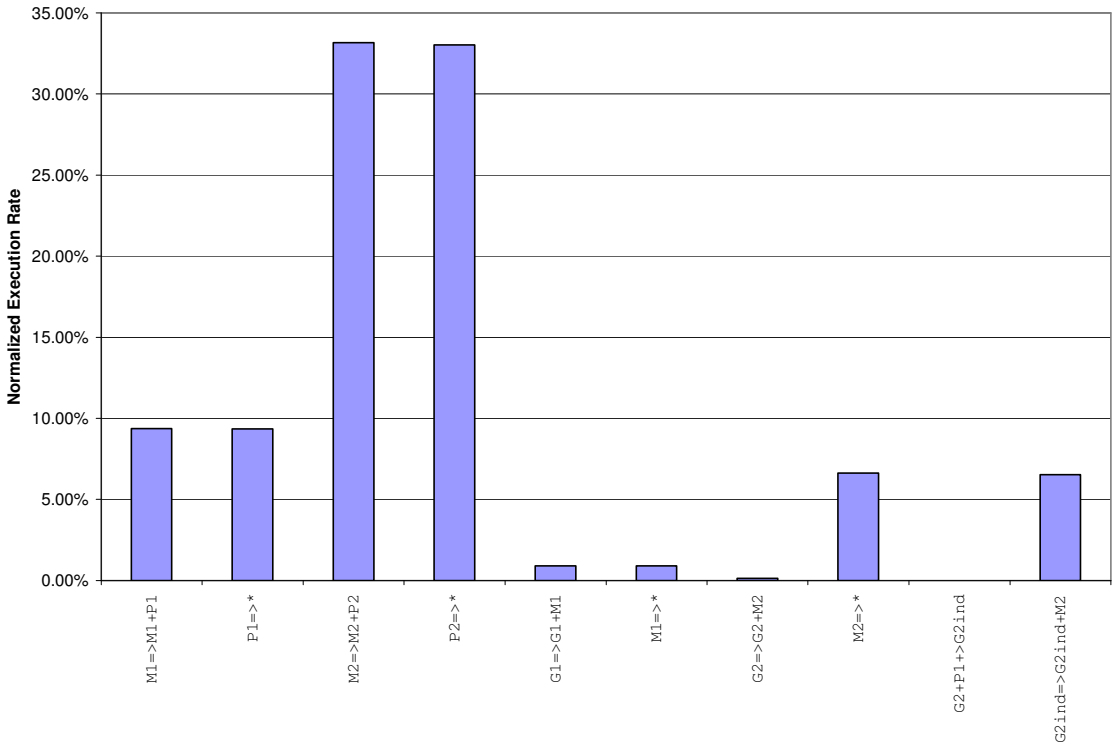


Figure 19. Normalized Reaction Execution Rate Histogram B. This histogram is based on the first 400 seconds of simulated time.

CurrentTime = 0.0
S[1..M] = Initial Species Populations
R[1..N] = Reactions
RSO[1..N] = 1..N (Reaction Search Order)
Dependencies = InitDependencyGraph(R)
TotalPropensity = 0.0
For I = 1..N
  Prop[I] = CalcPropensity(S,R[I])
  TotalPropensity = TotalPropensity + Prop[I]
End For
Goto Putative Time Calculation
	1. Initialization

DependentRxns = Dependencies[SelRxn]
Foreach DepRxn in DependentRxns
  TotalPropensity = TotalPropensity – Prop[DepRxn]
  Prop[DepRxn] = CalcPropensity(S,R[DepRxn])
  TotalPropensity = TotalPropensity + Prop[DepRxn]
End Foreach
	2. Propensity Calculation

T = -ln(rand())/TotalPropensity	3. Putative Time Calculation

Selector = TotalPropensity * rand()
For I = 1..N
  Selector = Selector – Prop[RSO[I]]
  If (Selector <= 0)
  RSOIndex = I
  Break
  End If
End For
SelRxn = RSO[RSOIndex]
	4. Reaction Selection

S = S – R[SelRxn].reactants + R[SelRxn].products
CurrentTime = CurrentTime + T
If (RSOIndex != 1)
  Temp = RSO[RSOIndex]
  RSO[RSOIndex] = RSO[RSOIndex – 1]
  RSO[RSOIndex] = Temp
End If
	5. Reaction Execution

If (CurrentTime < EndTime)
  Goto Propensity Calculation
End If
	6. Termination

Figure 20. Pseudo-code for the Sorting Direct Method.

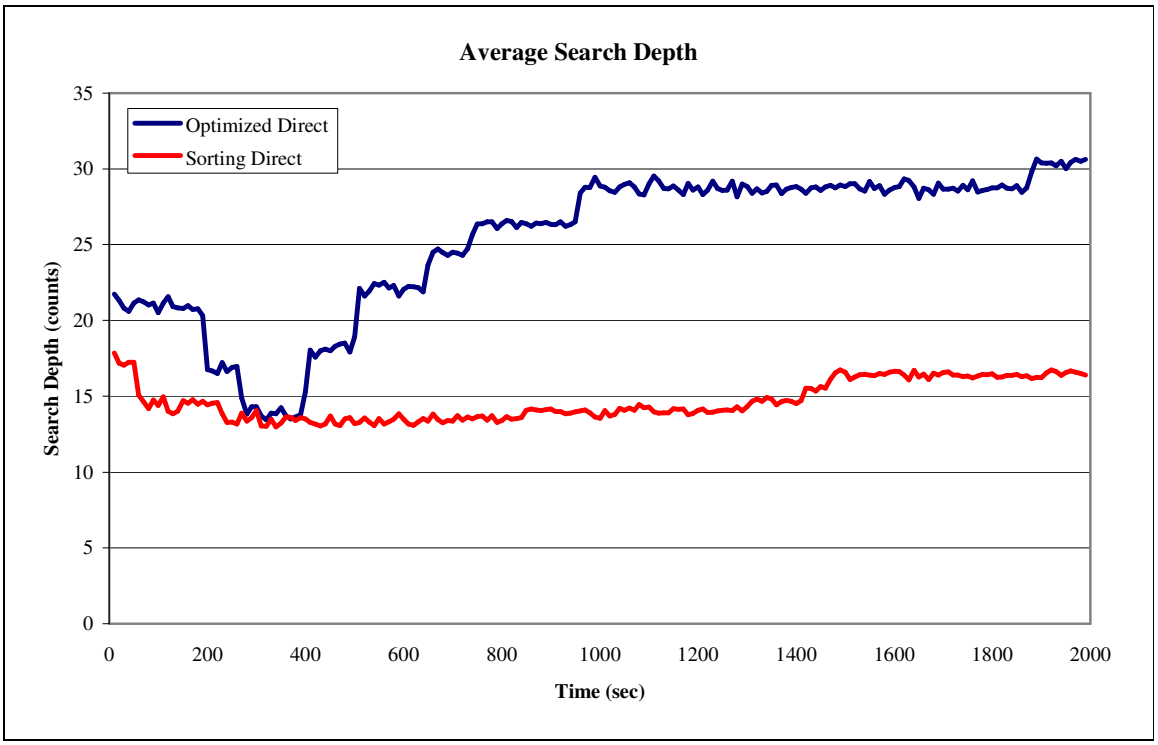


Figure 21. Average Search Depth Comparison. Average search depth for the ODM and the SDM while running the 20 gene model. Sharp decreases or increases occur in the search depth for the ODM when genes are induced, while the SDM remains small and constant throughout the simulation.

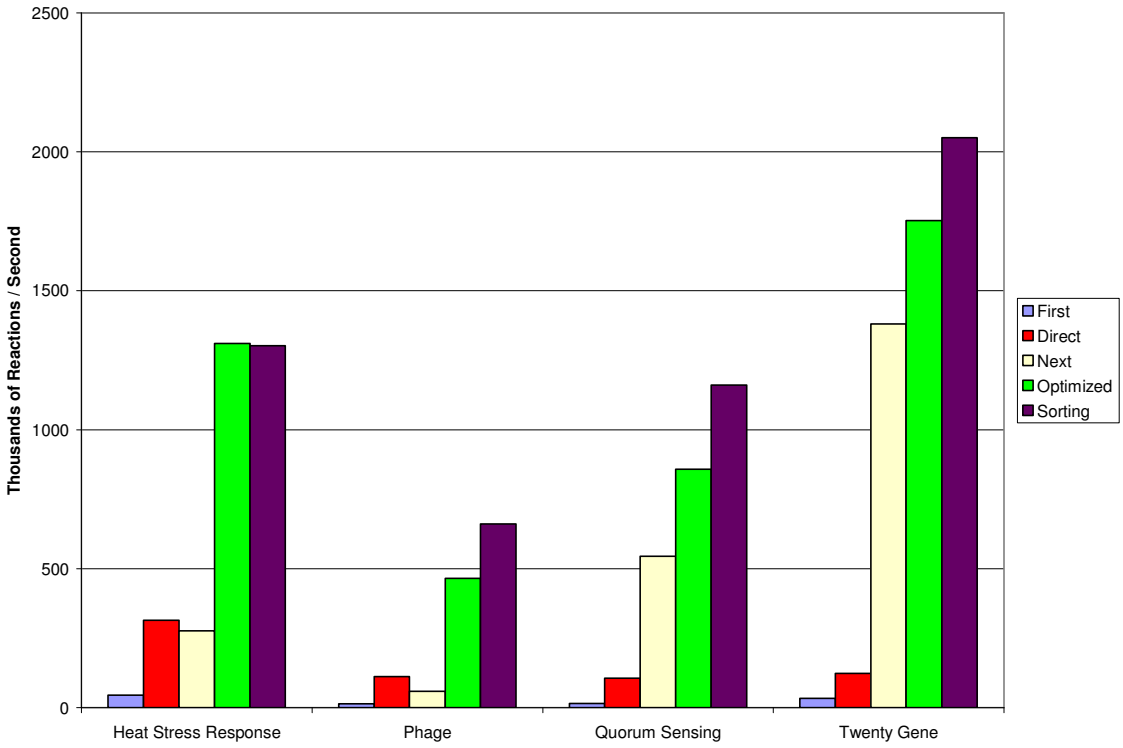


Figure 22. Simulator Performance Comparison. Comparison of simulator performance for a variety of models.

ReactionsToExecute = User Input
S[1..M] = Initial Species Populations
R[1..N] = Reactions
RSO[1..N] = 1..N (Reaction Search Order)
Dependencies = InitDependencyGraph(R)
TotalPropensity = 0.0
For I = 1..N
  Prop[I] = CalcPropensity(S,R[I])
  TotalPropensity = TotalPropensity + Prop[I]
End For
Goto Reaction Selection
	1. Initialization

DependentRxns = Dependencies[SelRxn]
Foreach DepRxn in DependentRxns
  TotalPropensity = TotalPropensity – Prop[DepRxn]
  Prop[DepRxn] = CalcPropensity(S,R[DepRxn])
  TotalPropensity = TotalPropensity + Prop[DepRxn]
End Foreach
	2. Propensity Calculation

	3. Putative Time Calculation

Selector = TotalPropensity * rand()
For I = 1..N
  Selector = Selector – Prop[RSO[I]]
  If (Selector <= 0)
  RSOIndex = I
  Break
  End If
End For
SelRxn = RSO[RSOIndex]
	4. Reaction Selection

S = S – R[SelRxn].reactants + R[SelRxn].products
ReactionsToExecute = ReactionsToExecute – 1
If (RSOIndex != 1)
  Temp = RSO[RSOIndex]
  RSO[RSOIndex] = RSO[RSOIndex – 1]
  RSO[RSOIndex] = Temp
End If
	5. Reaction Execution

If (ReactionsToExecute > 0)
  Goto Propensity Calculation
End If
	6. Termination

Figure 23. Pseudo-code for the Un-timed Sorting Direct Method.

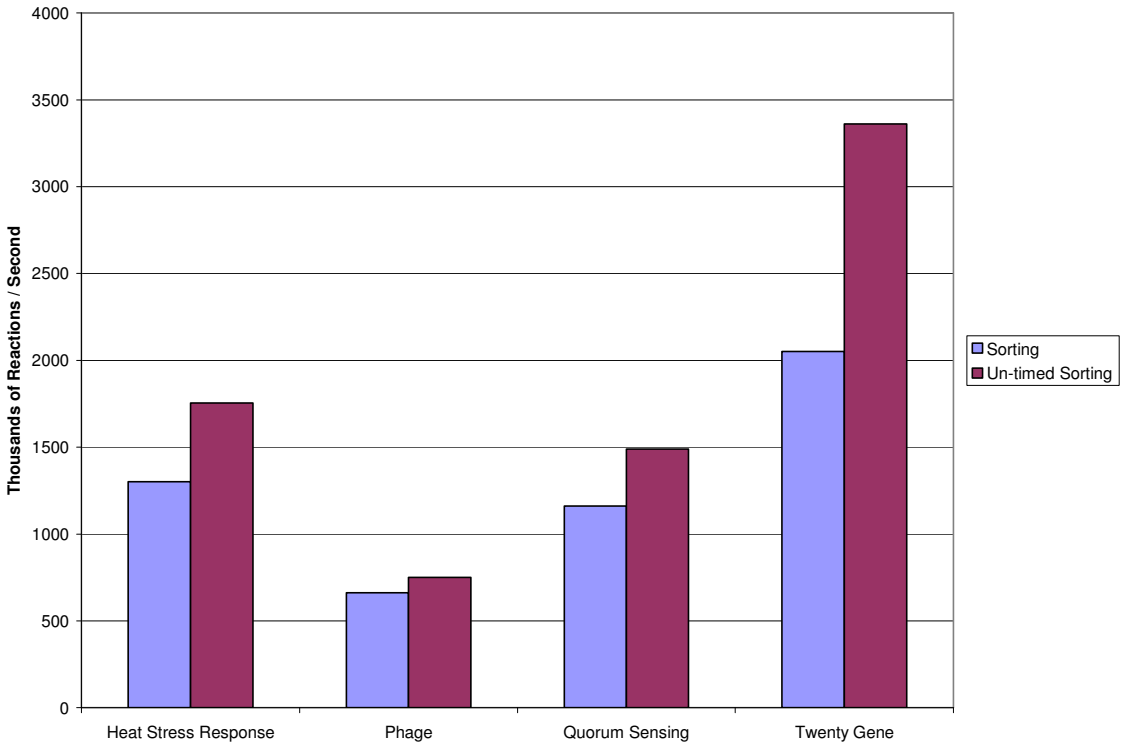


Figure 24. Un-timed vs. Timed Performance Comparison.

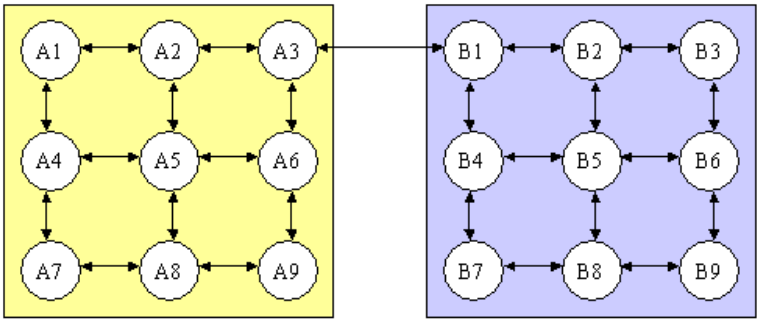


Figure 25. Artificial Model for Parallel Performance Analysis.

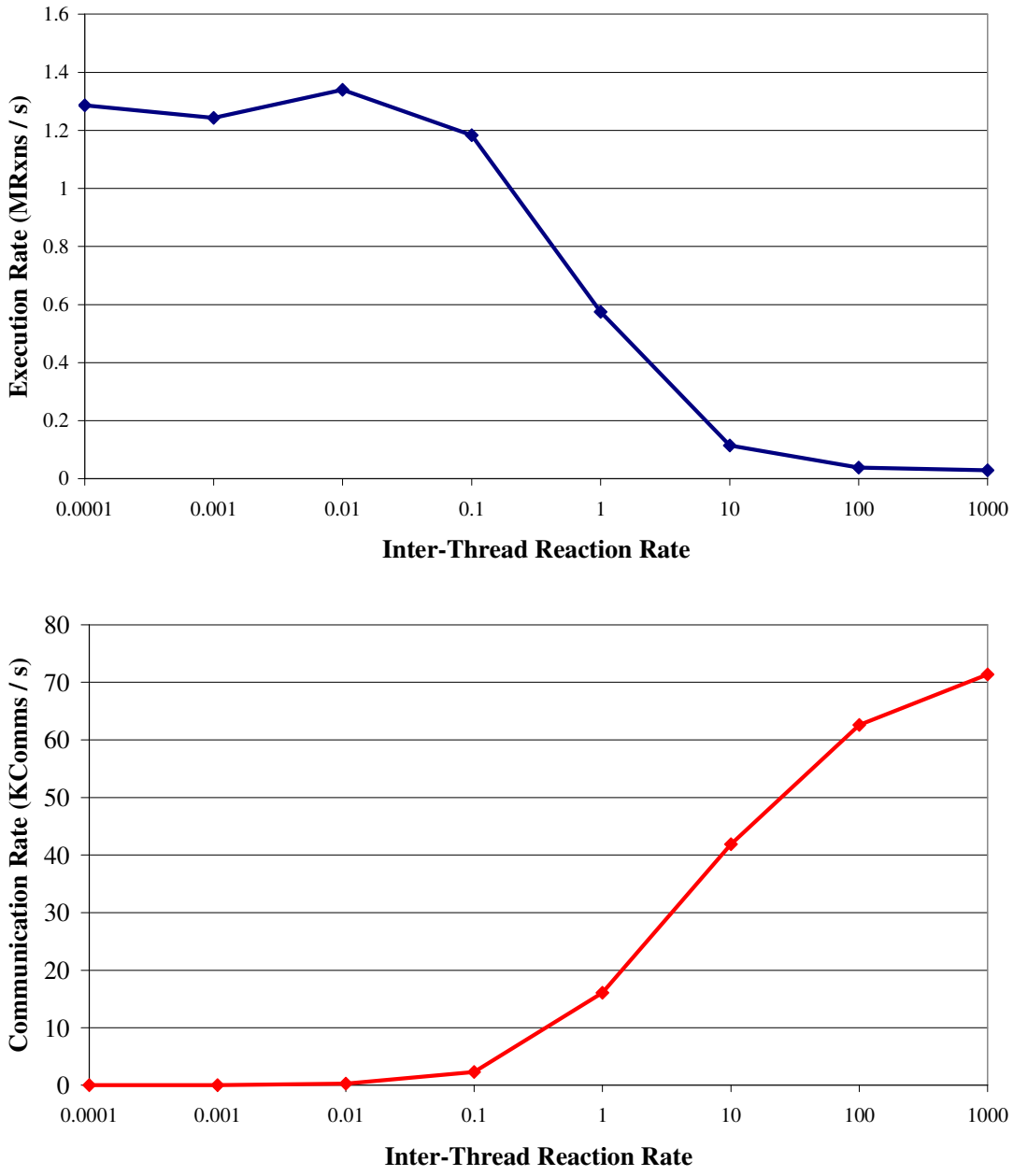


Figure 26. Parallel Execution and Communication Rates. Measurements of simulation speed and communication rate for the artificial model with varying inter-thread reaction rate constants.

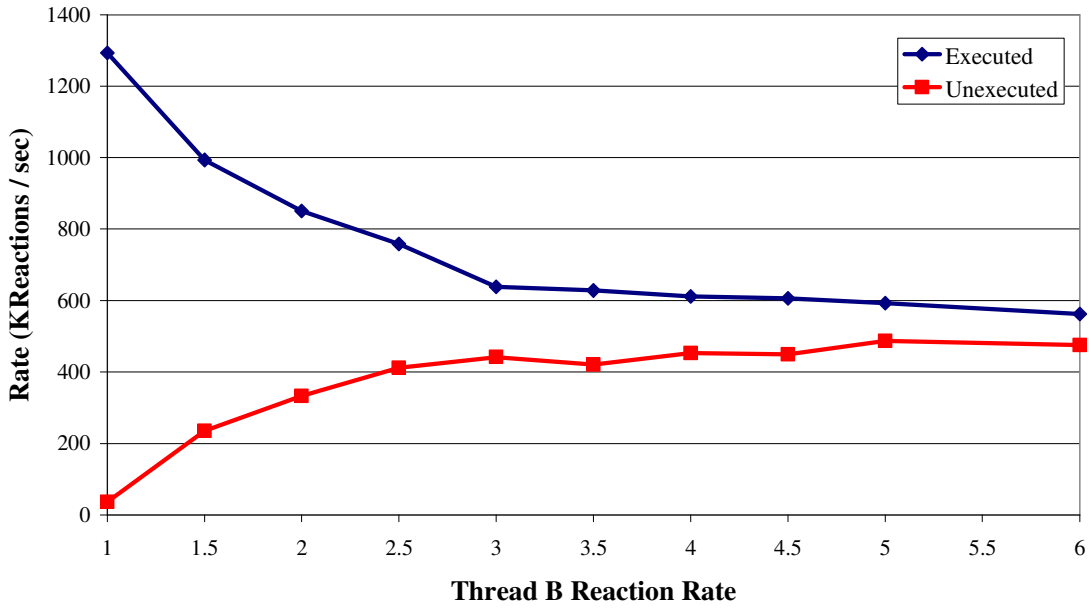


Figure 27. Parallel Execute and Unexecute Rates. Measurements of reaction execution rate and reaction un-execution rate for the artificial model with varying reaction load imbalance.

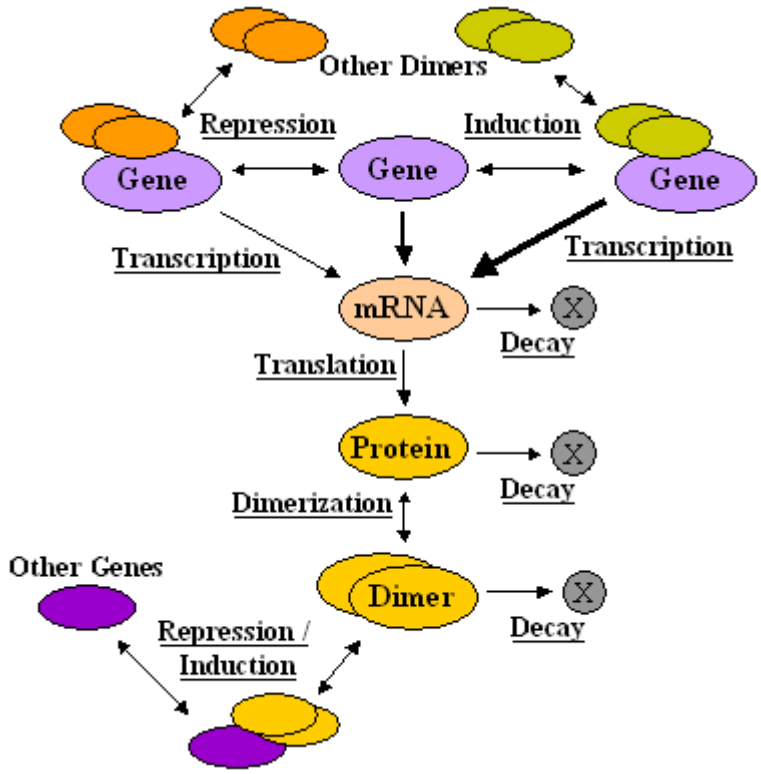


Figure 28. Graphical Representation of the *E. coli* Model Template.

Reaction	Rate	Function
$Gene_i$ => $Gene_i$ + $mRNA_i$	0.01445	Transcription
$mRNA_i$ => *	0.005776	Transcript Decay / Dilution
$mRNA_i$ => $mRNA_i$ + $Protein_i$	0.11552	Translation
$Protein_i$ => *	2.89E-04	Protein Decay / Dilution
$Protein_i$ + $Protein_i$ => $Dimer_i$	2.89E-04	Dimerization
$Dimer_i$ => $Protein_i$ + $Protein_i$	2.89E-03	Dimerization
$Dimer_i$ => *	2.89E-04	Dimer Decay / Dilution
$Dimer_i$ + $Gene_j$ => Induced $Gene_{ij}$	1.00E-05	Induction
Induced $Gene_{ij}$ => $Dimer_i$ + $Gene_j$	1.00E-03	Induction
$Dimer_i$ + $Gene_j$ => Repressed $Gene_{ij}$	1.00E-05	Repression
Repressed $Gene_{ij}$ => $Dimer_i$ + $Gene_j$	1.00E-03	Repression
Induced $Gene_{ij}$ => Induced $Gene_{ij}$ + $mRNA_j$	0.1445	Transcription
Repressed $Gene_{ij}$ => Repressed $Gene_{ij}$ + $mRNA_j$	1.45E-04	Transcription

Figure 29. The *E. coli* model template. Note that the induction the repression reactions only appear for genes and dimers that exist in the gene regulation data provided by Shenn-Orr (2002).

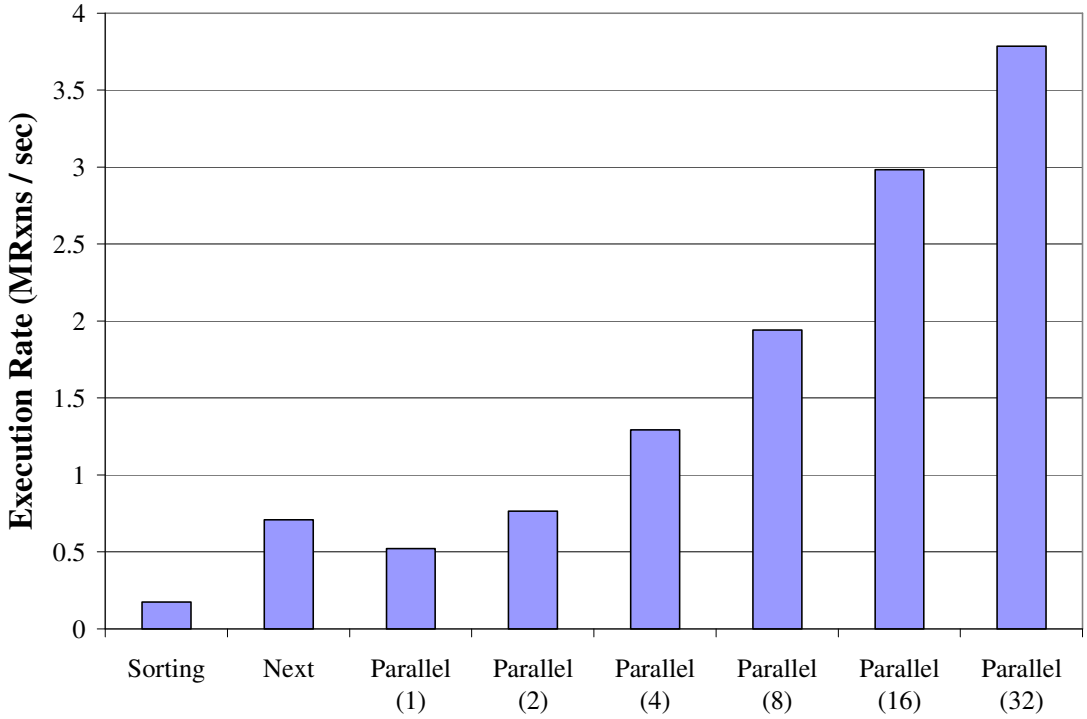


Figure 30. *E. coli* Parallel Performance Results.

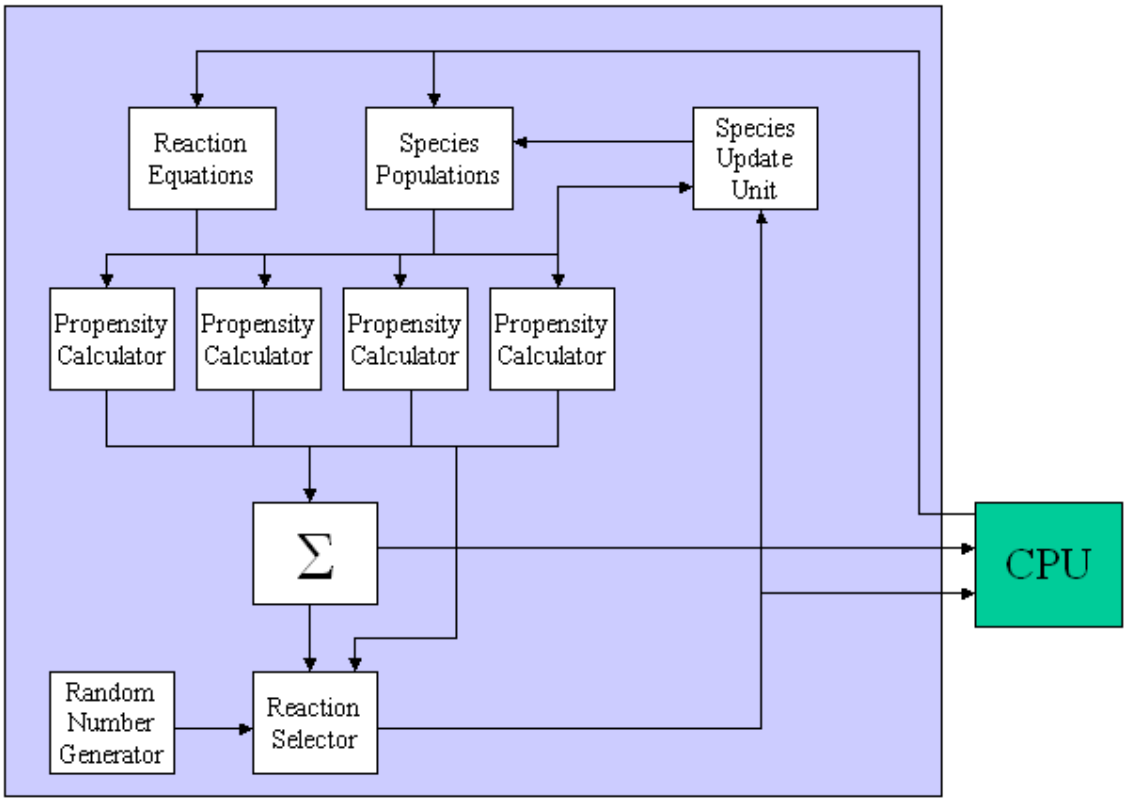


Figure 31. Block Diagram of the Original Hardware Design.

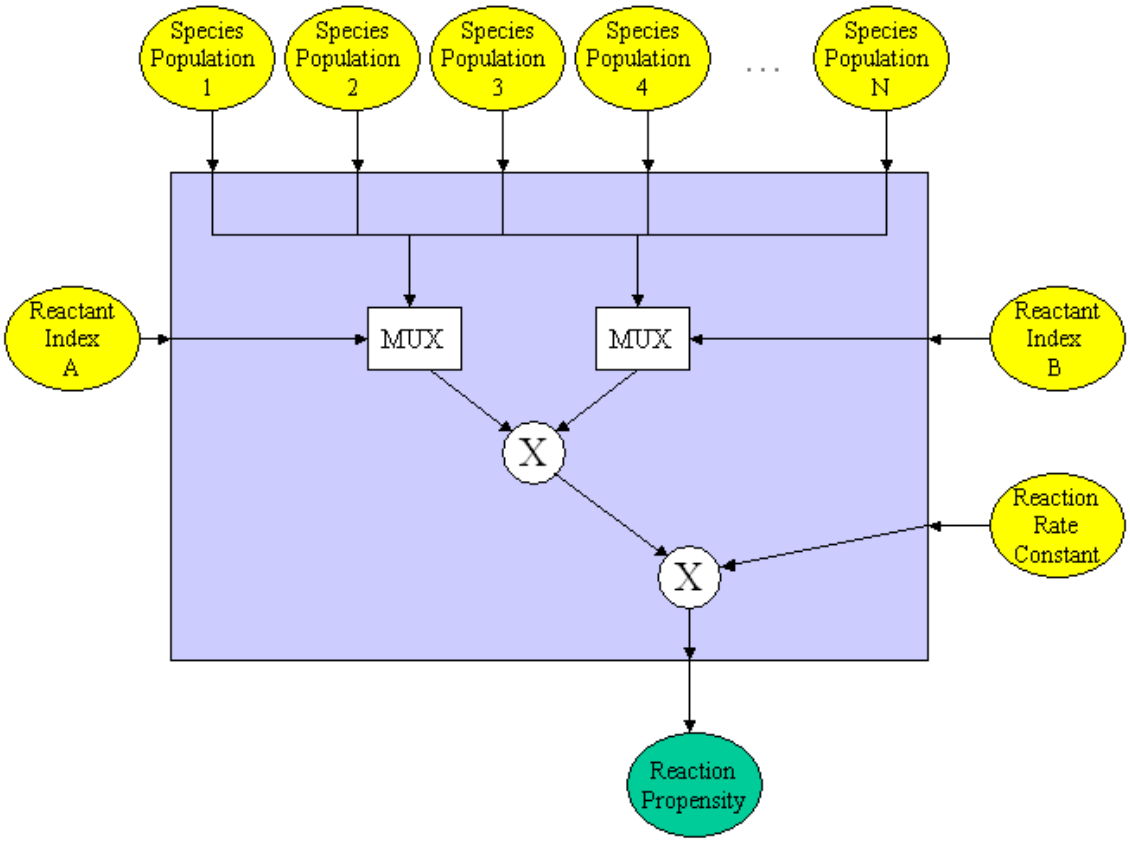


Figure 32. Propensity Calculator from the Original Hardware Design.

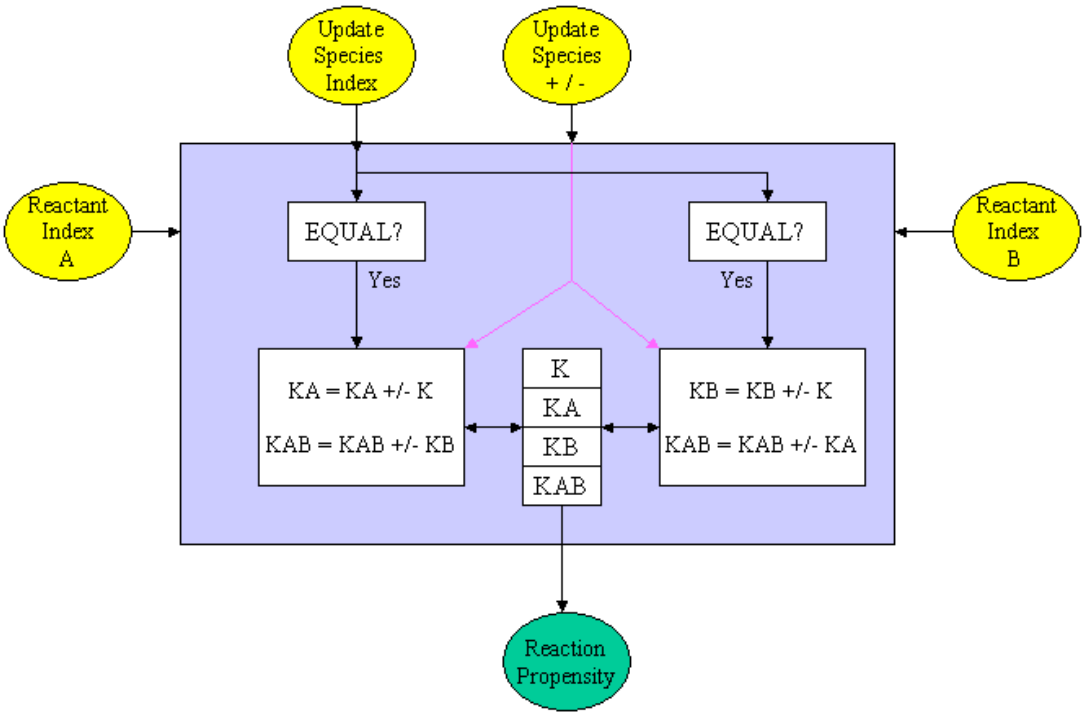


Figure 33. Block Diagram of the Optimized Propensity Calculator.

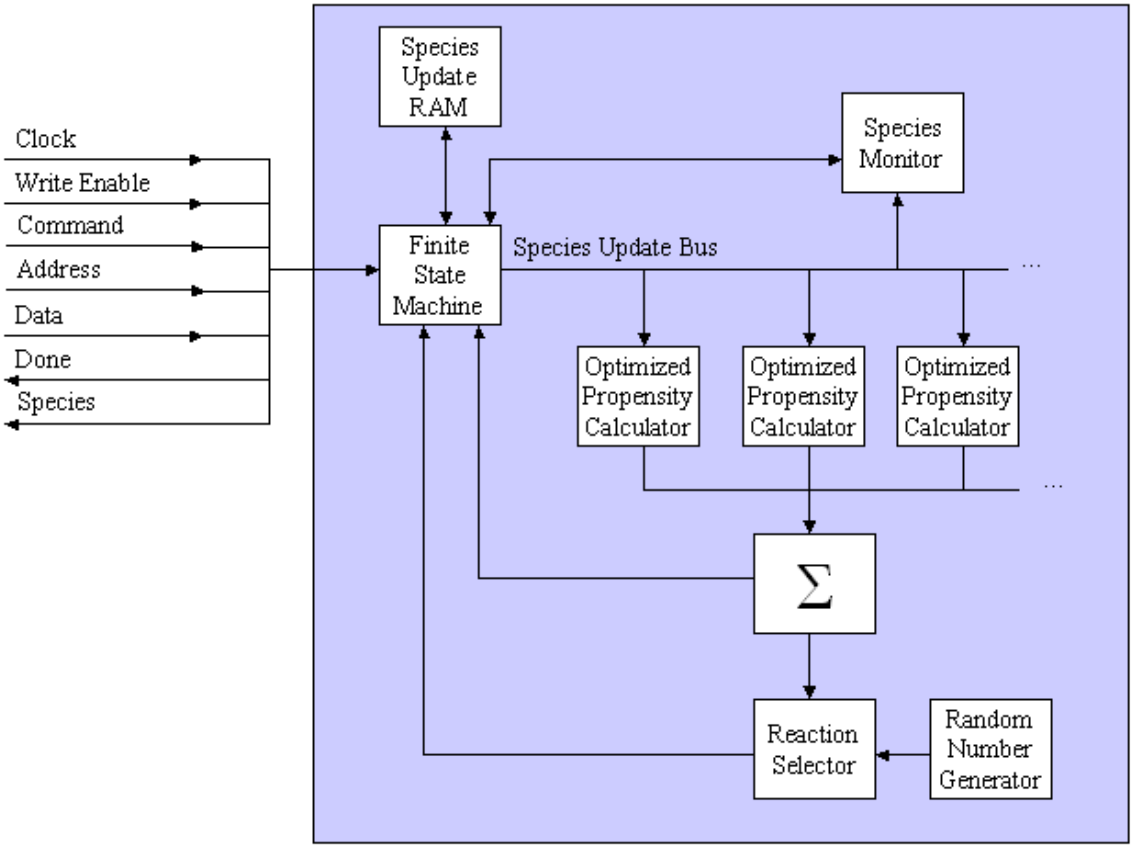
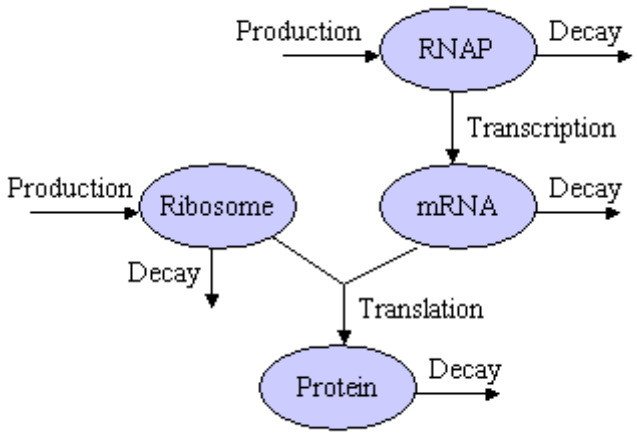


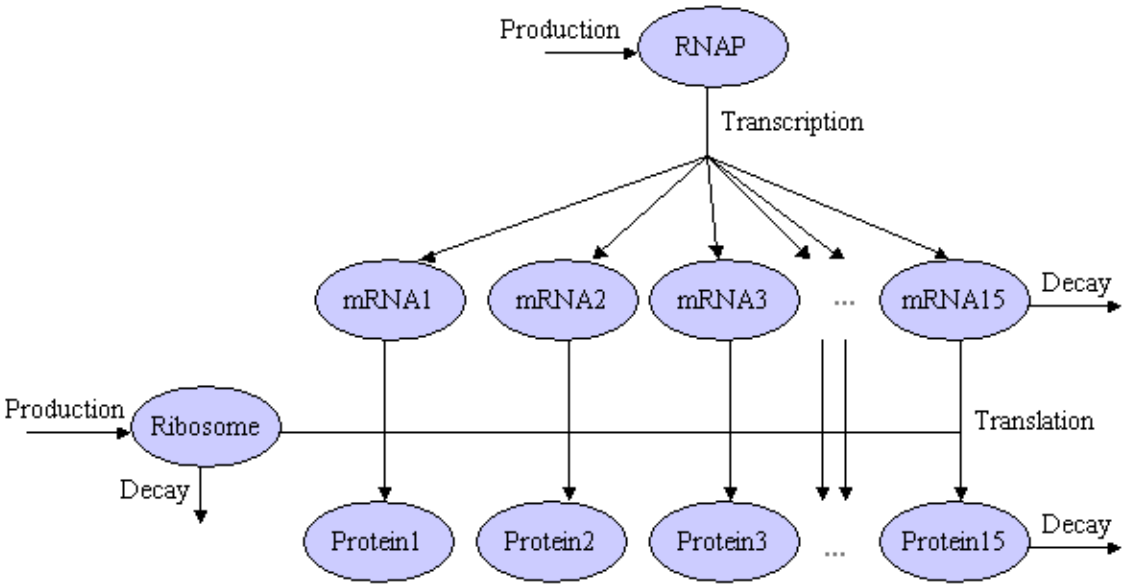
Figure 34. Block Diagram of the Final Hardware Design.




```
* -> RNAP                                          k = 784
RNAP -> *                                          k = 2
* -> RIBO                                          k = 78
RIBO -> *                                          k = 2
RNAP -> RNAP + mRNA                                k = 2
mRNA -> *                                          k = 58
mRNA + RIBO -> mRNA + RIBO + Protein               k = 22
Protein -> *                                       k = 3

RNAP = 0
RIBO = 0
mRNA = 0
Protein = 0
```

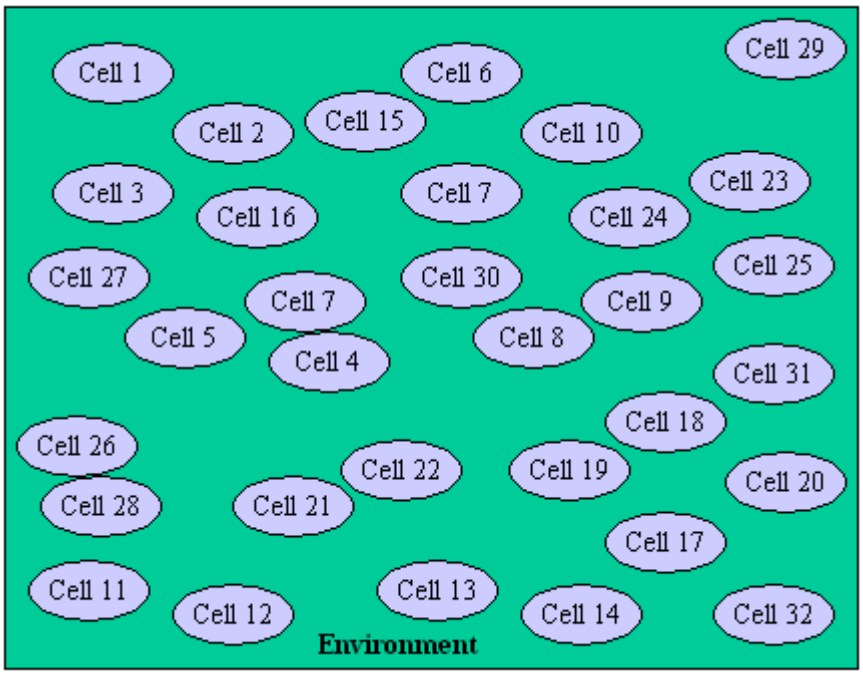
Figure 35. Single Gene Transcription and Translation Model.




```
* -> RNAP                                          k = 784
RNAP -> *                                          k = 2
* -> RIBO                                          k = 78
RIBO -> *                                          k = 2
RNAP -> RNAP + mRNA[1..15]                         k = 2
MRNA[1..15] -> *                                   k = 58
MRNA[1..15] + RIBO ->
     mRNA[1..15] + RIBO + Protein[1..15]           k = 22
Protein[1..15] -> *                                k = 3

RNAP = 0
RIBO = 0
MRNA[1..15] = 0
Protein[1..15] = 0
```

Figure 36. Fifteen Gene Transcription and Translation Model.




```
Environment => InsideCell[1..32]        k = 10
InsideCell[1..32] => Environment        k = 10

Environment = 10000
InsideCell[1..32] = 0
```

Figure 37. Diffusion Model.

Group	Tasks
Server	– Manages all data and connections associated with the grid – Accepts jobs from clients – Distributes jobs to workers – Monitors the progress of all jobs – Retrieves output data from workers – Sends output data to the clients
Client	– Connects to the server – Adds data in the form of files to the server – Adds jobs to be performed on the data to the server – Monitors the progress of the executing jobs – Retrieves the results of the jobs
Worker	– Connects to the server – Registers a set of commands that can be executed by this worker – Requests jobs to be executed from the server – Reports progress as it completes a job – Returns output results to the server
Monitor	– Connects to the server – Observers the status of all jobs in the server – Observes the status of all clients and workers associated with the system – Does not add or remove data from the system

Figure 38. Distributed Computing Environment Components

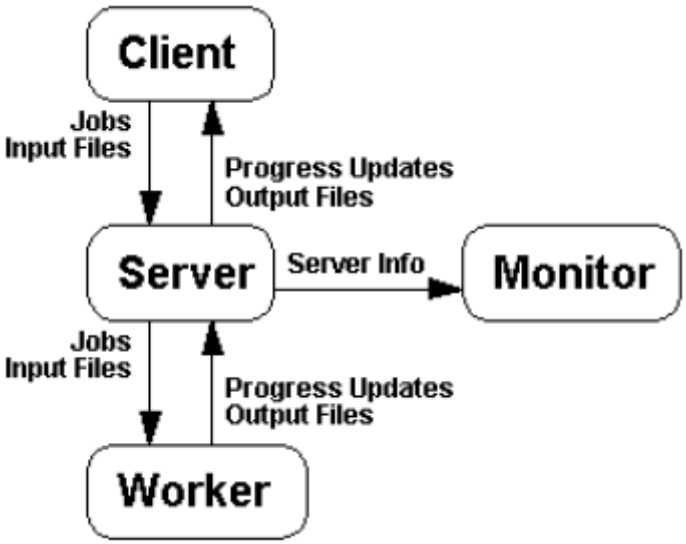
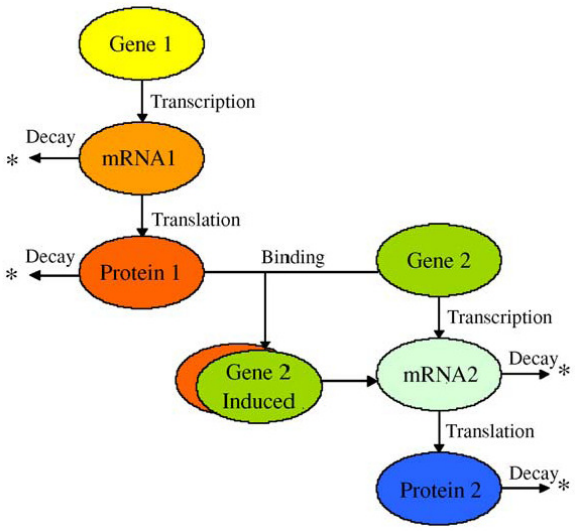


Figure 39. Distributed Computing Environment Dataflow Diagram.




```
R1:   GENE1 -> GENE1 + mRNA1            10.0
R2:   mRNA1 -> mRNA1 + Protein1         10.0
R3:   mRNA1 -> *                         1.0
R4:   Protein1 -> *                      1.0

R5:   GENE2 -> GENE2 + mRNA2             5.0
R6:   mRNA2 -> mRNA2 + Protein2          5.0
R7:   mRNA2 -> *                         1.0
R8:   Protein2 -> *                      1.0

R9:   Gene2 + Protein1 -> Gene2_IND   0.0001
R10:  Gene2_IND -> Gene2_IND + mRNA2   100.0
```

Figure 40. Gene Regulation Network Model. This model includes the transcription and translation of Gene1 into Protein1 and Gene2 into Protein2. When Protein1 binds to Gene2, it induces the production of Protein2.

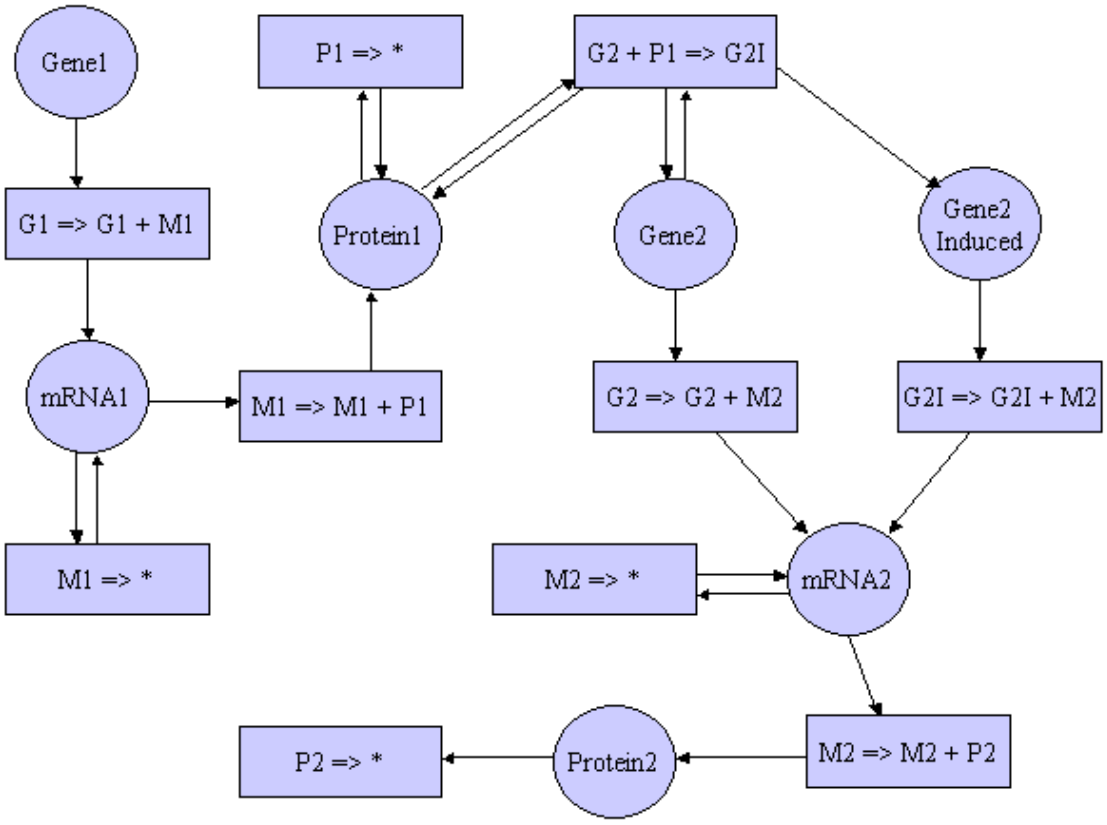


Figure 41. Directed Relationship Graph for the Two Gene Model.

```
Function RecursiveMarkNode(node N)
     If N is Not Marked
          Mark N
          Foreach Edge E in N's Edge List
               RecursiveMarkNode(Node Pointed To By Edge E)
          End Loop
     End If
End
```

Figure 42. Pseudo-code for Marking Dependent Nodes.

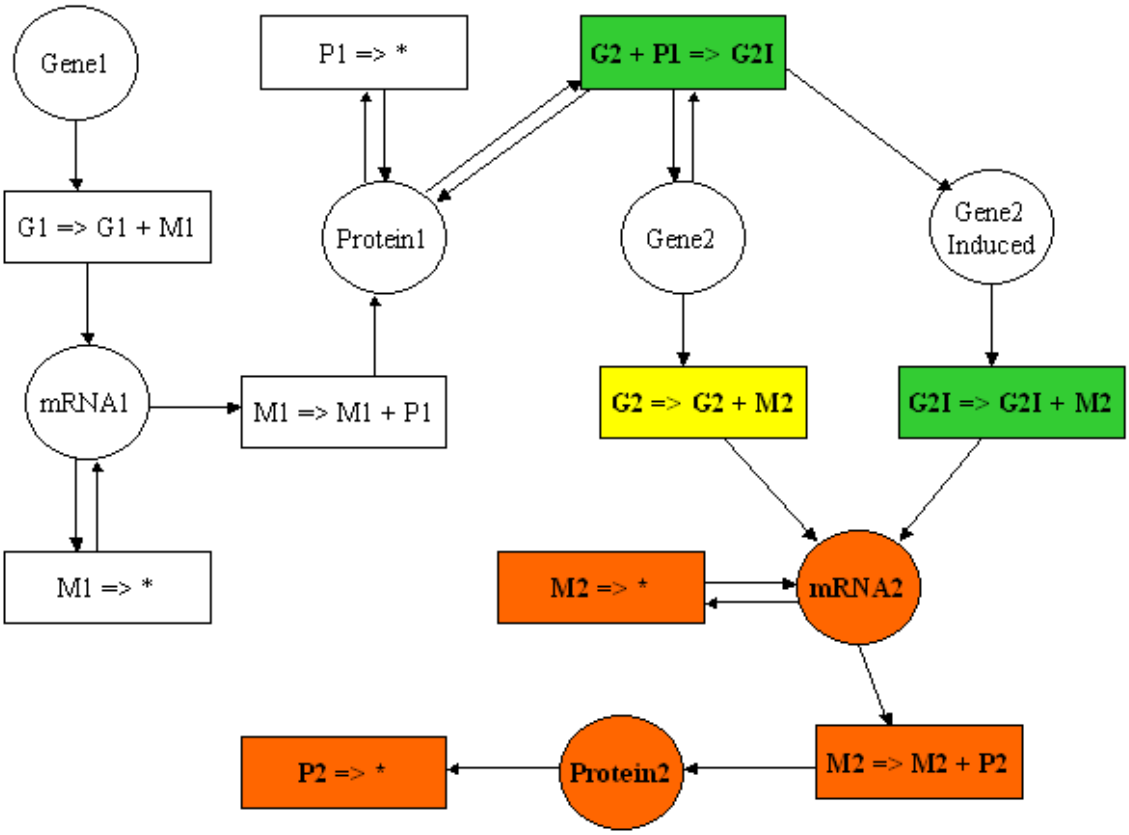


Figure 43. Highlighted Relationship Graph for the Two Gene Model. Species-Reaction relationship graph for the two gene model with and altered rate constant for reaction 5. The yellow node is the changed node. The orange nodes are dependent nodes. The green nodes are helper nodes.

S[1..M] = Initial Species Populations
R[1..N] = Reactions
<T_{Old}[1..N]> = Lists of Old Reaction Occurrence Times
$T_{Current}$ = 0.0
RSO[1..N] = 1..N (Reaction Search Order)
DG = InitDependencyGraph(R)
RG = InitSpeciesReactionRelationGraph(R)
TotalPropensity = 0.0
For I = 1..N
  If (RG.Changed(R[I]) or RG.Dependent(R[I]))
  Prop[I] = CalcPropensity(S,R[I])
  TotalPropensity = TotalPropensity + Prop[I]
  Else If (RG.Helper(R[I]))
  MinHeap.add(R[I],T_{Old}[I])
  End If
End For
T_{New} = −ln(rand())/TotalPropensity
Goto Reaction Selection
	1. Initialization

OldPropensity = TotalPropensity
DependentRxns = DG[R_{Next}]
Foreach DepRxn in DependentRxns
  If (RG.Changed(DepRxn) or RG.Dependent(DepRxn))
  TotalPropensity = TotalPropensity − Prop[DepRxn]
  Prop[DepRxn] = CalcPropensity(S,R[DepRxn])
  TotalPropensity = TotalPropensity + Prop[DepRxn]
  End If
End Foreach
	2. Propensity Calculation

If (T_{Next} == T_{New}) T_{New} = −ln(rand())/TotalPropensity
else
  MinHeap.update(R_{Next}, T_{Prev}[R_{Next}].next())
  T_{New} = OldPropensity/TotalPropensity(T_{New} − $T_{Current}$)
  T_{New} = T_{New} + $T_{Current}$
End If
	3. Putative Time Calculation

If (T_{New} < MinHeap.getMinTime())
  T_{Next} = T_{New}
  Selector = TotalPropensity * rand()
  For I = 1..N
  Selector = Selector − Prop[RSO[I]]
  If (Selector <= 0)
  RSOIndex = I
  Break
  End If
  End For
  R_{Next} = RSO[RSOIndex]
Else T_{Next} = T_{Old} and R_{Next} = MinHeap.getMinRxn()
	4. Reaction Selection

S = S − R_{Next}.reactants + R_{Next}.products
CurrentTime = T_{Next}
If (RSOIndex != 1)
  Temp = RSO[RSOIndex]
  RSO[RSOIndex] = RSO[RSOIndex − 1]
  RSO[RSOIndex] = Temp
End If
	5. Reaction Execution

If (CurrentTime < EndTime)
  Goto Propensity Calculation
End If
	6. Termination

Figure 44. Pseudo-code for the Partitioning Sorting Direct Method.

Matlab Code

The following Matlab code implements a deterministic solution to the auto-regulated biochemical model given in Figure 1-1.

```
[t,y] = ode15s(@diffeq,0:0.1:50,[1 0 0 0 0])

function Dy=diffeq(t,y)

     ktc = 10;
     kmd = 1;
     ktl = 10;
     kpd = 1;
     kdi = 1;
     krdi = 10;
     kbind = 0.1;
     kunbind = 10;

     g = y(1);
     m = y(2);
     p = y(3);
     d = y(4);
     r = y(5);

     Dg = kunbind*r - kbind*d*g;
     Dm = ktc*g - kmd*m;
     Dp = ktl*m - kpd*p - 2*kdi*p*p + 2*krdi*d;
     Dd = kdi*p*p - krdi*d - kbind*d*g + kunbind*r;
     Dr = kbind*d*g - kunbind*r;

     Dy = [Dg Dm Dp Dd Dr]';
```

Model Files

The following are a list of model files that can be read by the stochastic simulators used throughout this work . The format of these files is the following.

1. The number of species in the model (Integer)
2. The initial populations for each species (Integers)
3. The number of reactions in the model (Integer)
4. Each reaction in the model
    a. The number of reactants (Integer)
    b. Each reactant listed as a reactant coefficient (Integer) and reactant species index (Integer).
    c. The number of products (Integer)
    d. Each product listed as a reactant coefficient (Integer) and reactant species index (Integer).
    e. The reaction rate constant (Real)
5. The number of species to output (Integer)
6. The species index for each species to output (Integer)

Autoregulated Gene Circuit Model

```
5
1 0 0 0 0
8
1 1 0 2 1 0 1 1 10.0
1 1 1 0 1.0
1 1 1 2 1 1 1 2 10.0
1 1 2 0 1.0
1 2 2 1 1 3 1.0
1 1 3 1 2 2 10.0
2 1 3 1 0 1 1 4 0.1
1 1 4 2 1 3 1 0 10.0
5
0 1 2 3 4
```

Heat Stress Response Model

```
28
0 0 0 0 1 4645670 1324 80 16 3413 29 584 1 22 0 171440 9150 2280 6 596
0 13 3 3 7 0 260 0
61
2 1 0 1 1 1 1 2 2.54
1 1 2 2 1 0 1 1 1.0
2 1 0 1 3 1 1 4 0.254
1 1 4 2 1 0 1 3 1.0
2 1 0 1 5 1 1 6 0.0254
2 1 3 1 13 1 1 14 254.0
1 1 14 2 1 3 1 13 10000.0
2 1 13 1 15 1 1 16 2.54E-4
1 1 16 2 1 13 1 15 0.01
2 1 2 1 5 1 1 7 2.54E-4
1 1 7 2 1 2 1 5 1.0
2 1 4 1 5 1 1 8 2.54E-4
1 1 8 2 1 4 1 5 1.0
2 1 2 1 9 1 1 11 2.54
1 1 11 2 1 2 1 9 1.0
2 1 4 1 10 1 1 12 2540.0
1 1 12 2 1 4 1 10 1000.0
2 1 14 1 17 1 1 18 0.0254
1 1 18 2 1 14 1 17 1.0
0 1 1 21 6.62
1 1 21 0 0.5
0 1 1 13 20.0
1 1 13 0 0.03
1 1 16 1 1 15 0.03
1 1 14 1 1 3 0.03
1 1 18 2 1 3 1 17 0.03
1 1 27 2 1 3 1 26 0.03
0 1 1 22 1.67
1 1 22 0 0.5
0 1 1 17 20.0
1 1 17 0 0.03
1 1 18 1 1 14 0.03
0 1 1 24 0.00625
1 1 24 0 0.5
0 1 1 3 7.0
1 1 3 0 0.03
1 1 18 2 1 13 1 17 3.0
1 1 20 1 1 19 0.7
1 1 27 2 1 13 1 26 0.5
0 1 1 23 1.0
1 1 23 0 0.5
0 1 1 19 20.0
1 1 19 0 0.03
1 1 20 1 1 3 0.03
2 1 3 1 19 1 1 20 2.54
1 1 20 2 1 3 1 19 10000.0
0 1 1 25 0.43333
1 1 25 0 0.5
```

```
0 1 1 26 20.0
1 1 26 0 0.03
1 1 27 1 1 14 0.03
2 1 14 1 26 1 1 27 2.54
1 1 27 2 1 14 1 26 10000.0
1 1 4 1 1 0 0.03
1 1 12 2 1 10 1 0 0.03
1 1 8 1 1 6 0.03
1 1 14 1 1 13 0.03
1 1 18 2 1 13 1 17 0.03
1 1 27 2 1 13 1 26 0.03
1 1 20 1 1 19 0.03
1 1 6 2 1 0 1 5 10.0
0
```

Lambda Phage Model

```
61
0 0 12 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 100 0 0 0 0 1 1 0 0 0 0 0 0
1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 40 30 1 1 0 0 0 0 0 0 1 0
117
2 1 53 1 2 3 1 53 1 2 10 9 0.011
2 1 50 1 14 1 1 54 0.01
1 1 54 2 1 50 1 14 1.0
3 1 50 1 0 1 12 1 1 6 0.02569
1 1 6 3 1 50 1 0 1 12 1.0
3 1 50 1 0 1 12 1 1 35 0.00967
1 1 35 3 1 50 1 0 1 12 1.0
2 1 26 1 9 1 1 5 0.2
1 1 5 2 1 26 1 9 1.0
2 1 49 1 36 1 1 3 0.01
1 1 3 2 1 49 1 36 0.01
1 1 43 0 7.0E-4
2 1 0 1 12 1 1 10 0.2165
1 1 10 2 1 0 1 12 1.0
3 1 2 1 28 1 60 4 1 2 1 28 1 60 10 55 0.014
3 1 34 1 2 1 41 4 1 34 1 2 1 41 10 43 0.0010
3 1 35 1 2 1 33 4 1 35 1 2 1 33 10 55 0.014
3 1 30 1 5 1 2 4 1 30 1 5 1 2 10 55 0.014
2 1 51 1 9 1 1 60 0.2
1 1 60 2 1 51 1 9 1.0
3 1 34 1 2 1 6 4 1 34 1 2 1 6 10 43 0.0010
1 2 31 1 1 1 0.1
1 1 1 1 2 31 0.5
4 1 1 1 50 1 0 1 12 1 1 19 8.0E-5
1 1 19 4 1 1 1 50 1 0 1 12 1.0
2 1 9 1 52 1 1 33 0.2
1 1 33 2 1 9 1 52 1.0
3 1 53 1 2 1 27 4 10 36 1 53 1 2 1 27 0.0022
2 2 1 1 40 1 1 42 0.0316
1 1 42 2 2 1 1 40 1.0
1 1 56 1 1 21 0.6
3 1 51 1 2 1 28 4 1 51 1 2 1 28 10 55 0.0070
3 2 1 1 50 1 12 1 1 32 5.2E-4
1 1 32 3 2 1 1 50 1 12 1.0
2 1 50 1 12 1 1 28 0.69422
1 1 28 2 1 50 1 12 1.0
1 1 18 1 1 21 0.0010
2 1 0 1 40 1 1 29 0.2025
1 1 29 2 1 0 1 40 1.0
2 1 49 1 55 1 1 37 2.0E-4
1 1 37 2 1 49 1 55 0.05
1 1 9 0 0.00231
2 1 14 1 55 1 1 38 0.00726
1 1 38 2 1 14 1 55 1.0
3 1 1 1 0 1 12 1 1 4 0.1779
1 1 4 3 1 1 1 0 1 12 1.0
1 1 37 1 1 49 0.6
3 1 20 1 2 1 59 4 1 20 1 2 1 59 10 55 0.0070
```

```
2 1 2 1 24 3 1 2 10 43 1 24 0.015
2 1 30 1 2 3 10 31 1 30 1 2 0.014
3 1 50 1 0 1 12 1 1 57 0.0019
1 1 57 3 1 50 1 0 1 12 1.0
1 2 43 1 1 0 0.1
1 1 0 1 2 43 0.5
3 1 35 1 2 1 52 4 1 35 1 2 1 52 10 55 0.0070
3 1 19 1 2 1 7 4 1 19 1 2 10 43 1 7 0.011
2 3 0 1 12 1 1 23 0.00486
1 1 23 2 3 0 1 12 1.0
2 2 1 1 12 1 1 44 0.06684
1 1 44 2 2 1 1 12 1.0
2 2 0 1 40 1 1 46 0.116
1 1 46 2 2 0 1 40 1.0
3 1 53 1 47 1 2 4 10 36 1 53 1 47 1 2 0.011
3 1 2 1 45 1 7 4 1 2 1 45 10 43 1 7 0.011
3 1 1 1 50 1 12 1 1 25 0.01186
1 1 25 3 1 1 1 50 1 12 1.0
2 1 35 1 2 3 1 35 10 31 1 2 0.014
3 1 1 1 0 1 40 1 1 39 0.014
1 1 39 3 1 1 1 0 1 40 1.0
2 1 2 1 54 3 1 2 10 43 1 54 4.0E-5
2 1 9 1 27 1 1 47 0.2
1 1 47 2 1 9 1 27 1.0
3 1 26 1 30 1 2 4 1 26 1 30 1 2 10 55 0.0070
3 1 34 1 2 1 25 4 1 34 1 2 10 43 1 25 0.0010
2 1 50 1 12 1 1 41 0.1362
1 1 41 2 1 50 1 12 1.0
3 1 1 2 0 1 12 1 1 8 0.04266
1 1 8 3 1 1 2 0 1 12 1.0
3 1 1 1 50 1 12 1 1 30 0.25123
1 1 30 3 1 1 1 50 1 12 1.0
3 1 50 1 14 1 55 1 1 24 0.00161
1 1 24 3 1 50 1 14 1 55 1.0
2 1 1 1 14 1 1 11 1.0E-5
1 1 11 2 1 1 1 14 0.1
3 1 34 1 2 1 15 4 1 34 1 2 1 15 10 43 0.0010
1 1 31 0 0.0025
2 1 21 1 55 1 1 56 2.5E-4
1 1 56 2 1 21 1 55 0.065
2 2 0 1 12 1 1 22 0.13136
1 1 22 2 2 0 1 12 1.0
1 1 3 1 1 49 0.0010
2 2 50 1 12 1 1 20 0.1891
1 1 20 2 2 50 1 12 1.0
3 2 1 1 0 1 12 1 1 48 0.00644
1 1 48 3 2 1 1 0 1 12 1.0
3 1 20 1 34 1 2 4 1 20 1 34 1 2 10 43 0.0010
2 1 36 1 21 1 1 18 0.01
1 1 18 2 1 36 1 21 0.01
2 1 1 1 12 1 1 13 0.449
1 1 13 2 1 1 1 12 1.0
2 3 1 1 12 1 1 58 0.00414
1 1 58 2 3 1 1 12 1.0
2 1 1 1 40 1 1 17 0.4132
```

```
1 1 17 2 1 1 1 40 1.0
3 1 32 1 34 1 2 4 1 32 1 34 1 2 10 43 0.0010
4 1 1 1 50 1 0 1 12 1 1 15 0.00112
1 1 15 4 1 1 1 50 1 0 1 12 1.0
3 1 50 2 0 1 12 1 1 45 0.0158
1 1 45 3 1 50 2 0 1 12 1.0
2 1 9 1 59 1 1 16 0.2
1 1 16 2 1 9 1 59 1.0
2 1 2 1 28 3 10 31 1 2 1 28 0.014
3 1 20 1 2 1 16 4 1 20 1 2 1 16 10 55 0.014
2 1 20 1 2 3 10 31 1 20 1 2 0.014
3 1 2 1 57 1 7 4 1 57 1 2 10 43 1 7 0.011
2 1 50 1 40 1 1 53 0.6942
1 1 53 2 1 50 1 40 1.0
61
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
```

Quorum Sensing Model

```
122
1 16000000 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0
1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0
0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1
0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0
201
1 1 0 1 2 0 .0001194
1 1 2 2 1 2 1 3 5.55e-3
1 1 3 0 2.78e-5
1 2 3 1 1 4 .00003
1 1 4 1 2 3 .01
1 2 4 1 1 5 .0006
1 1 5 1 2 4 .01
2 1 6 1 5 1 1 7 .02
1 1 7 2 1 6 1 5 .01
1 1 7 2 1 7 1 8 0.06
1 1 8 0 .006
1 1 8 2 1 8 1 9 .03
1 1 9 0 0.0006
1 1 1 1 1 10 4.7e-7
1 1 10 1 1 1 0.8
2 1 9 1 10 1 1 11 0.001
1 1 11 2 1 9 1 10 0.3636
1 2 11 1 1 12 0.02
1 1 12 1 2 11 0.433
2 1 12 1 13 1 1 14 0.1
1 1 14 2 1 12 1 13 4
1 1 14 2 1 14 1 15 0.3
1 1 15 0 0.006
1 1 15 2 1 15 1 16 0.03
1 1 16 0 0.0006
2 1 16 1 0 3 1 0 1 16 1 1 0.006
1 1 17 2 1 17 1 18 5.55e-3
1 1 18 0 2.78e-5
1 2 18 1 1 19 .00003
1 1 19 1 2 18 .01
1 2 19 1 1 20 .0006
1 1 20 1 2 19 .01
2 1 21 1 20 1 1 22 .02
1 1 22 2 1 21 1 20 .01
1 1 22 2 1 22 1 23 0.06
1 1 23 0 .006
1 1 23 2 1 23 1 24 .03
1 1 24 0 0.0006
1 1 1 1 1 25 4.7e-7
1 1 25 1 1 1 0.8
2 1 24 1 25 1 1 26 0.001
1 1 26 2 1 24 1 25 0.3636
1 2 26 1 1 27 0.02
1 1 27 1 2 26 0.433
2 1 27 1 28 1 1 29 0.1
1 1 29 2 1 27 1 28 4
```

```
1 1 29 2 1 29 1 30 0.3
1 1 30 0 0.006
1 1 30 2 1 30 1 31 0.03
1 1 31 0 0.0006
2 1 31 1 0 3 1 0 1 31 1 1 0.006
1 1 32 2 1 32 1 33 5.55e-3
1 1 33 0 2.78e-5
1 2 33 1 1 34 .00003
1 1 34 1 2 33 .01
1 2 34 1 1 35 .0006
1 1 35 1 2 34 .01
2 1 36 1 35 1 1 37 .02
1 1 37 2 1 36 1 35 .01
1 1 37 2 1 37 1 38 0.06
1 1 38 0 .006
1 1 38 2 1 38 1 39 .03
1 1 39 0 0.0006
1 1 1 1 1 40 4.7e-7
1 1 40 1 1 1 0.8
2 1 39 1 40 1 1 41 0.001
1 1 41 2 1 39 1 40 0.3636
1 2 41 1 1 42 0.02
1 1 42 1 2 41 0.433
2 1 42 1 43 1 1 44 0.1
1 1 44 2 1 42 1 43 4
1 1 44 2 1 44 1 45 0.3
1 1 45 0 0.006
1 1 45 2 1 45 1 46 0.03
1 1 46 0 0.0006
2 1 46 1 0 3 1 0 1 46 1 1 0.006
1 1 47 2 1 47 1 48 5.55e-3
1 1 48 0 2.78e-5
1 2 48 1 1 49 .00003
1 1 49 1 2 48 .01
1 2 49 1 1 50 .0006
1 1 50 1 2 49 .01
2 1 51 1 50 1 1 52 .02
1 1 52 2 1 51 1 50 .01
1 1 52 2 1 52 1 53 0.06
1 1 53 0 .006
1 1 53 2 1 53 1 54 .03
1 1 54 0 0.0006
1 1 1 1 1 55 4.7e-7
1 1 55 1 1 1 0.8
2 1 54 1 55 1 1 56 0.001
1 1 56 2 1 54 1 55 0.3636
1 2 56 1 1 57 0.02
1 1 57 1 2 56 0.433
2 1 57 1 58 1 1 59 0.1
1 1 59 2 1 57 1 58 4
1 1 59 2 1 59 1 60 0.3
1 1 60 0 0.006
1 1 60 2 1 60 1 61 0.03
1 1 61 0 0.0006
2 1 61 1 0 3 1 0 1 61 1 1 0.006
```

```
1 1 62 2 1 62 1 63 5.55e-3
1 1 63 0 2.78e-5
1 2 63 1 1 64 .00003
1 1 64 1 2 63 .01
1 2 64 1 1 65 .0006
1 1 65 1 2 64 .01
2 1 66 1 65 1 1 67 .02
1 1 67 2 1 66 1 65 .01
1 1 67 2 1 67 1 68 0.06
1 1 68 0 .006
1 1 68 2 1 68 1 69 .03
1 1 69 0 0.0006
1 1 1 1 1 70 4.7e-7
1 1 70 1 1 1 0.8
2 1 69 1 70 1 1 71 0.001
1 1 71 2 1 69 1 70 0.3636
1 2 71 1 1 72 0.02
1 1 72 1 2 71 0.433
2 1 72 1 73 1 1 74 0.1
1 1 74 2 1 72 1 73 4
1 1 74 2 1 74 1 75 0.3
1 1 75 0 0.006
1 1 75 2 1 75 1 76 0.03
1 1 76 0 0.0006
2 1 76 1 0 3 1 0 1 76 1 1 0.006
1 1 77 2 1 77 1 78 5.55e-3
1 1 78 0 2.78e-5
1 2 78 1 1 79 .00003
1 1 79 1 2 78 .01
1 2 79 1 1 80 .0006
1 1 80 1 2 79 .01
2 1 81 1 80 1 1 82 .02
1 1 82 2 1 81 1 80 .01
1 1 82 2 1 82 1 83 0.06
1 1 83 0 .006
1 1 83 2 1 83 1 84 .03
1 1 84 0 0.0006
1 1 1 1 1 85 4.7e-7
1 1 85 1 1 1 0.8
2 1 84 1 85 1 1 86 0.001
1 1 86 2 1 84 1 85 0.3636
1 2 86 1 1 87 0.02
1 1 87 1 2 86 0.433
2 1 87 1 88 1 1 89 0.1
1 1 89 2 1 87 1 88 4
1 1 89 2 1 89 1 90 0.3
1 1 90 0 0.006
1 1 90 2 1 90 1 91 0.03
1 1 91 0 0.0006
2 1 91 1 0 3 1 0 1 91 1 1 0.006
1 1 92 2 1 92 1 93 5.55e-3
1 1 93 0 2.78e-5
1 2 93 1 1 94 .00003
1 1 94 1 2 93 .01
1 2 94 1 1 95 .0006
```

```
1 1 95 1 2 94 .01
2 1 96 1 95 1 1 97 .02
1 1 97 2 1 96 1 95 .01
1 1 97 2 1 97 1 98 0.06
1 1 98 0 .006
1 1 98 2 1 98 1 99 .03
1 1 99 0 0.0006
1 1 1 1 1 100 4.7e-7
1 1 100 1 1 1 0.8
2 1 99 1 100 1 1 101 0.001
1 1 101 2 1 99 1 100 0.3636
1 2 101 1 1 102 0.02
1 1 102 1 2 101 0.433
2 1 102 1 103 1 1 104 0.1
1 1 104 2 1 102 1 103 4
1 1 104 2 1 104 1 105 0.3
1 1 105 0 0.006
1 1 105 2 1 105 1 106 0.03
1 1 106 0 0.0006
2 1 106 1 0 3 1 0 1 106 1 1 0.006
1 1 107 2 1 107 1 108 5.55e-3
1 1 108 0 2.78e-5
1 2 108 1 1 109 .00003
1 1 109 1 2 108 .01
1 2 109 1 1 110 .0006
1 1 110 1 2 109 .01
2 1 111 1 110 1 1 112 .02
1 1 112 2 1 111 1 110 .01
1 1 112 2 1 112 1 113 0.06
1 1 113 0 .006
1 1 113 2 1 113 1 114 .03
1 1 114 0 0.0006
1 1 1 1 1 115 4.7e-7
1 1 115 1 1 1 0.8
2 1 114 1 115 1 1 116 0.001
1 1 116 2 1 114 1 115 0.3636
1 2 116 1 1 117 0.02
1 1 117 1 2 116 0.433
2 1 117 1 118 1 1 119 0.1
1 1 119 2 1 117 1 118 4
1 1 119 2 1 119 1 120 0.3
1 1 120 0 0.006
1 1 120 2 1 120 1 121 0.03
1 1 121 0 0.0006
2 1 121 1 0 3 1 0 1 121 1 1 0.006
17
1 8 15 23 30 38 45 53 60 68 75 83 90 98 105 113 120
```

Twenty Gene Model

```
100
1 0 0 1 0 0 0   0 0 0
1 0 0 1 0 0 0   0 0 0
1 0 0 1 0 0 0   0 0 0
1 0 0 1 0 0 0   0 0 0
1 0 0 1 0 0 0   0 0 0
1 0 0 1 0 0 0   0 0 0
1 0 0 1 0 0 0   0 0 0
1 0 0 1 0 0 0   0 0 0
1 0 0 1 0 0 0   0 0 0
1 0 0 1 0 0 0   0 0 0

100
1 1 0 2 1 0 1 1 10.0
1 1 1 2 1 1 1 2 10.0
1 1 1 0 1.0
1 1 2 0 1.0
1 1 3 2 1 3 1 4 5.0
1 1 4 2 1 4 1 5 5.0
1 1 4 0 1.0
1 1 5 0 1.0
2 1 3 1 2 1 1 6 0.0001
1 1 6 2 1 6 1 4 100.0

1 1 10 2 1 10 1 11 10.0
1 1 11 2 1 11 1 12 10.0
1 1 11 0 1.0
1 1 12 0 1.0
1 1 13 2 1 13 1 14 5.0
1 1 14 2 1 14 1 15 5.0
1 1 14 0 1.0
1 1 15 0 1.0
2 1 13 1 11 1 1 16 0.0001
1 1 16 2 1 16 1 14 100.0

1 1 20 2 1 20 1 21 10.0
1 1 21 2 1 21 1 22 10.0
1 1 21 0 1.0
1 1 22 0 1.0
1 1 23 2 1 23 1 24 5.0
1 1 24 2 1 24 1 25 5.0
1 1 24 0 1.0
1 1 25 0 1.0
2 1 23 1 21 1 1 26 0.0001
1 1 26 2 1 26 1 24 100.0

1 1 30 2 1 30 1 31 10.0
1 1 31 2 1 31 1 32 10.0
1 1 31 0 1.0
1 1 32 0 1.0
1 1 33 2 1 33 1 34 5.0
1 1 34 2 1 34 1 35 5.0
```

```
1 1 34 0 1.0
1 1 35 0 1.0
2 1 33 1 31 1 1 36 0.0001
1 1 36 2 1 36 1 34 100.0

1 1 40 2 1 40 1 41 10.0
1 1 41 2 1 41 1 42 10.0
1 1 41 0 1.0
1 1 42 0 1.0
1 1 43 2 1 43 1 44 5.0
1 1 44 2 1 44 1 45 5.0
1 1 44 0 1.0
1 1 45 0 1.0
2 1 43 1 41 1 1 46 0.0001
1 1 46 2 1 46 1 44 100.0

1 1 50 2 1 50 1 51 10.0
1 1 51 2 1 51 1 52 10.0
1 1 51 0 1.0
1 1 52 0 1.0
1 1 53 2 1 53 1 54 5.0
1 1 54 2 1 54 1 55 5.0
1 1 54 0 1.0
1 1 55 0 1.0
2 1 53 1 51 1 1 56 0.0001
1 1 56 2 1 56 1 54 100.0

1 1 60 2 1 60 1 61 10.0
1 1 61 2 1 61 1 62 10.0
1 1 61 0 1.0
1 1 62 0 1.0
1 1 63 2 1 63 1 64 5.0
1 1 64 2 1 64 1 65 5.0
1 1 64 0 1.0
1 1 65 0 1.0
2 1 63 1 61 1 1 66 0.0001
1 1 66 2 1 66 1 64 100.0

1 1 70 2 1 70 1 71 10.0
1 1 71 2 1 71 1 72 10.0
1 1 71 0 1.0
1 1 72 0 1.0
1 1 73 2 1 73 1 74 5.0
1 1 74 2 1 74 1 75 5.0
1 1 74 0 1.0
1 1 75 0 1.0
2 1 73 1 71 1 1 76 0.0001
1 1 76 2 1 76 1 74 100.0

1 1 80 2 1 80 1 81 10.0
1 1 81 2 1 81 1 82 10.0
1 1 81 0 1.0
1 1 82 0 1.0
1 1 83 2 1 83 1 84 5.0
1 1 84 2 1 84 1 85 5.0
```

```
1 1 84 0 1.0
1 1 85 0 1.0
2 1 83 1 81 1 1 86 0.0001
1 1 86 2 1 86 1 84 100.0

1 1 90 2 1 90 1 91 10.0
1 1 91 2 1 91 1 92 10.0
1 1 91 0 1.0
1 1 92 0 1.0
1 1 93 2 1 93 1 94 5.0
1 1 94 2 1 94 1 95 5.0
1 1 94 0 1.0
1 1 95 0 1.0
2 1 93 1 91 1 1 96 0.0001
1 1 96 2 1 96 1 94 100.0

10
6 16 26 36 46 56 66 76 86 96
```

Transcription Translation Single Gene Model

```
4
0 0 0 0
8
0 1 1 0 784.0
1 1 0 0 2.0
1 1 0 2 1 0 1 1 2.0
1 1 1 0 58.0
0 1 1 2 78.0
1 1 2 0 2.0
2 1 1 1 2 3 1 1 1 2 1 3 22.0
1 1 3 0 3.0
4
0
1
2
3
```

Transcription Translation Fifteen Gene Model

```
32

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

64

0 1 1 0 784.0
1 1 0 0 2.0
0 1 1 1 78.0
1 1 1 0 2.0

1 1 0 2 1 0 1 2 2.0
1 1 2 0 58.0
2 1 1 1 2 3 1 1 1 2 1 3 22.0
1 1 3 0 3.0

1 1 0 2 1 0 1 4 2.0
1 1 4 0 58.0
2 1 1 1 4 3 1 1 1 4 1 5 22.0
1 1 5 0 3.0

1 1 0 2 1 0 1 6 2.0
1 1 6 0 58.0
2 1 1 1 6 3 1 1 1 6 1 7 22.0
1 1 7 0 3.0

1 1 0 2 1 0 1 8 2.0
1 1 8 0 58.0
2 1 1 1 8 3 1 1 1 8 1 9 22.0
1 1 9 0 3.0

1 1 0 2 1 0 1 10 2.0
1 1 10 0 58.0
2 1 1 1 10 3 1 1 1 10 1 11 22.0
1 1 11 0 3.0

1 1 0 2 1 0 1 12 2.0
1 1 12 0 58.0
2 1 1 1 12 3 1 1 1 12 1 13 22.0
1 1 13 0 3.0

1 1 0 2 1 0 1 14 2.0
1 1 14 0 58.0
2 1 1 1 14 3 1 1 1 14 1 15 22.0
1 1 15 0 3.0

1 1 0 2 1 0 1 16 2.0
1 1 16 0 58.0
2 1 1 1 16 3 1 1 1 16 1 17 22.0
```

```
1 1 17 0 3.0

1 1 0 2 1 0 1 18 2.0
1 1 18 0 58.0
2 1 1 1 18 3 1 1 1 18 1 19 22.0
1 1 19 0 3.0

1 1 0 2 1 0 1 20 2.0
1 1 20 0 58.0
2 1 1 1 20 3 1 1 1 20 1 21 22.0
1 1 21 0 3.0

1 1 0 2 1 0 1 22 2.0
1 1 22 0 58.0
2 1 1 1 22 3 1 1 1 22 1 23 22.0
1 1 23 0 3.0

1 1 0 2 1 0 1 24 2.0
1 1 24 0 58.0
2 1 1 1 24 3 1 1 1 24 1 25 22.0
1 1 25 0 3.0

1 1 0 2 1 0 1 26 2.0
1 1 26 0 58.0
2 1 1 1 26 3 1 1 1 26 1 27 22.0
1 1 27 0 3.0

1 1 0 2 1 0 1 28 2.0
1 1 28 0 58.0
2 1 1 1 28 3 1 1 1 28 1 29 22.0
1 1 29 0 3.0

1 1 0 2 1 0 1 30 2.0
1 1 30 0 58.0
2 1 1 1 30 3 1 1 1 30 1 31 22.0
1 1 31 0 3.0

4
3 5 7 9
```

Diffusion Model

```
33
10000
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
64
1 1 0 1 1 1 10.0
1 1 0 1 1 2 10.0
1 1 0 1 1 3 10.0
1 1 0 1 1 4 10.0
1 1 0 1 1 5 10.0
1 1 0 1 1 6 10.0
1 1 0 1 1 7 10.0
1 1 0 1 1 8 10.0
1 1 0 1 1 9 10.0
1 1 0 1 1 10 10.0
1 1 0 1 1 11 10.0
1 1 0 1 1 12 10.0
1 1 0 1 1 13 10.0
1 1 0 1 1 14 10.0
1 1 0 1 1 15 10.0
1 1 0 1 1 16 10.0
1 1 0 1 1 17 10.0
1 1 0 1 1 18 10.0
1 1 0 1 1 19 10.0
1 1 0 1 1 20 10.0
1 1 0 1 1 21 10.0
1 1 0 1 1 22 10.0
1 1 0 1 1 23 10.0
1 1 0 1 1 24 10.0
1 1 0 1 1 25 10.0
1 1 0 1 1 26 10.0
1 1 0 1 1 27 10.0
1 1 0 1 1 28 10.0
1 1 0 1 1 29 10.0
1 1 0 1 1 30 10.0
1 1 0 1 1 31 10.0
1 1 0 1 1 32 10.0
1 1 1 1 1 0 10.0
1 1 2 1 1 0 10.0
1 1 3 1 1 0 10.0
1 1 4 1 1 0 10.0
1 1 5 1 1 0 10.0
1 1 6 1 1 0 10.0
1 1 7 1 1 0 10.0
1 1 8 1 1 0 10.0
1 1 9 1 1 0 10.0
1 1 10 1 1 0 10.0
1 1 11 1 1 0 10.0
1 1 12 1 1 0 10.0
1 1 13 1 1 0 10.0
1 1 14 1 1 0 10.0
```

```
1 1 15 1 1 0 10.0
1 1 16 1 1 0 10.0
1 1 17 1 1 0 10.0
1 1 18 1 1 0 10.0
1 1 19 1 1 0 10.0
1 1 20 1 1 0 10.0
1 1 21 1 1 0 10.0
1 1 22 1 1 0 10.0
1 1 23 1 1 0 10.0
1 1 24 1 1 0 10.0
1 1 25 1 1 0 10.0
1 1 26 1 1 0 10.0
1 1 27 1 1 0 10.0
1 1 28 1 1 0 10.0
1 1 29 1 1 0 10.0
1 1 30 1 1 0 10.0
1 1 31 1 1 0 10.0
1 1 32 1 1 0 10.0
3
0 1 2
```

Common Source Code

The following source code is shared between the serial and parallel stochastic simulation codes and is located in the "common/" directory during compilation.

Exception.cpp

```
#include "Exception.h"

Exception::Exception(char *errorClass, char *errorFunction, char
*errorMessage) {
  d_class = errorClass;
  d_function = errorFunction;
  d_message = errorMessage;
}

char *Exception::getClass() const {
  return d_class;
}

char *Exception::getFunction() const {
  return d_function;
}

char *Exception::getMessage() const {
  return d_message;
}
```

Exception.h

```
#ifndef EXCEPTION_H
#define EXCEPTION_H

class Exception {
 private:
  char *d_class;
  char *d_function;
  char *d_message;

 public:
  Exception(char *errorClass, char *errorFunction, char *errorMessage);
  char *getMessage() const;
  char *getFunction() const;
  char *getClass() const;
};

#endif
```

Infinity.h

```
#ifndef INFINITY_H
#define INFINITY_H

#include <math.h>

const double MYINFINITY = -log(0.0);

#endif
```

InputFile.cpp

```
#include "InputFile.h"
#include "Exception.h"

InputFile::InputFile(char *filename) {
  d_in = fopen(filename,"r");
  if (d_in == NULL) {
    throw Exception("InputFile","InputFile","Unable to open file.");
  }
}

InputFile::~InputFile() {
  if (d_in != NULL) {
    fclose(d_in);
  }
}

int InputFile::getInt() {
  int num;
  if (fscanf(d_in,"%d",&num) != 1)
    throw Exception("InputFile","getInt","Error reading int.");
  return num;
}

unsigned int InputFile::getUnsignedInt() {
  unsigned int num;
  if (fscanf(d_in,"%u",&num) != 1)
    throw Exception("InputFile","getUnsignedInt","Error reading
unsigned int.");
  return num;
}

long InputFile::getLong() {
  long num;
  if (fscanf(d_in,"%ld",&num) != 1)
    throw Exception("InputFile","getLong","Error reading long.");
  return num;
}

unsigned long InputFile::getUnsignedLong() {
  unsigned long num;
  if (fscanf(d_in,"%lu",&num) != 1)
```

```
    throw Exception("InputFile","getUnsignedLong","Error reading
unsigned long.");
  return num;
}

char InputFile::getChar() {
  char c;
  if (fscanf(d_in,"%c",&c) != 1)
    throw Exception("InputFile","getChar","Error reading char.");
  return c;
}

float InputFile::getFloat() {
  float num;
  if (fscanf(d_in,"%f",&num) != 1)
    throw Exception("InputFile","getFloat","Error reading float.");
  return num;
}

double InputFile::getDouble() {
  double num;
  if (fscanf(d_in,"%lf",&num) != 1)
    throw Exception("InputFile","getDouble","Error reading double.");
  return num;
}

long double InputFile::getLongDouble() {
  long double num;
  if (fscanf(d_in,"%Lf",&num) != 1)
    throw Exception("InputFile","getLongDouble","Error reading
double.");
  return num;
}
```

InputFile.h

```
#ifndef INPUTFILE_H
#define INPUTFILE_H

#include <stdio.h>

class InputFile {
 private:
  FILE *d_in;

 public:
  InputFile(char *filename);
  ~InputFile();

  int getInt();
  unsigned int getUnsignedInt();
  long getLong();
  unsigned long getUnsignedLong();
  char getChar();
```

```
  float getFloat();
  double getDouble();
  long double getLongDouble();
};

#endif
```

makefile

```
CPPFLAGS = -O3
CXX = c++

all: Exception.o InputFile.o Model.o Reaction.o ReactionElement.o

Exception.o: Exception.h Exception.cpp
InputFile.o: InputFile.h InputFile.cpp
Reaction.o: Exception.h Reaction.h ReactionElement.h InputFile.h
Vector.h
ReactionElement.o: ReactionElement.h Exception.h
Model.o: Model.h Reaction.h ReactionElement.h Exception.h InputFile.h

clean:
   -rm *.o *~
```

Model.cpp

```
#include "Model.h"
#include "Exception.h"
#include <stdlib.h>
#include <iostream>

using namespace std;

Model::Model(char *filename) {
#ifdef DEBUG
  if (filename == NULL) {
    throw Exception("Model","Model","filename == NULL");
  }
#endif

  d_areReactionDependenciesBuilt = false;

  InputFile in(filename);
  unsigned long i;

  try {
    d_species.resize(in.getUnsignedLong());
    for(i=0; i<d_species.size(); i++) {
      d_species.set(i,in.getUnsignedLong());
    }
  } catch (Exception e) {
    throw Exception("Model","Model","Error reading species list from
input file.");
  }
```

```
  try {
    d_reactions.resize(in.getUnsignedLong());
    for(i=0; i<d_reactions.size(); i++) {
      d_reactions.getPtr(i)->read(in);
    }
  } catch (Exception e) {
    throw Exception("Model","Model","Error reading reaction list from
input file.");
  }

  try {
    d_outputs.resize(in.getUnsignedLong());
    for(i=0; i<d_outputs.size(); i++) {
      d_outputs.set(i,in.getUnsignedLong());
      if (d_outputs.get(i) >= d_species.size()) {
  throw Exception("Model","Model","Invalid species specified in output
list.");
      }
    }
  } catch (Exception e) {
    throw Exception("Model","Model","Error reading output list from
input file.");
  }

  getReactionDependencies();
}

void Model::buildReactionDependencies() {
  d_reactionDependencies.resize(d_reactions.size());
  Vector< Vector<unsigned long> >
speciesDependencies(d_species.size());
  Vector< Vector<unsigned long> > speciesAffectors(d_species.size());

  for(unsigned long reactionIndex=0; reactionIndex<d_reactions.size();
reactionIndex++) {
    Reaction* reaction = d_reactions.getPtr(reactionIndex);
    for(unsigned long reactantIndex=0; reactantIndex<reaction-
>getReactants()->size(); reactantIndex++) {
      unsigned long speciesIndex = reaction->getReactants()-
>getPtr(reactantIndex)->getSpeciesIndex();
      speciesDependencies.getPtr(speciesIndex)->add(reactionIndex);
      if (reaction->affectsSpecies(speciesIndex)) {
       speciesAffectors.getPtr(speciesIndex)->add(reactionIndex);
      }
    }
    for(unsigned long productIndex=0; productIndex<reaction-
>getProducts()->size(); productIndex++) {
      unsigned long speciesIndex = reaction->getProducts()-
>getPtr(productIndex)->getSpeciesIndex();
      if (reaction->affectsSpecies(speciesIndex)) {
       speciesAffectors.getPtr(speciesIndex)->add(reactionIndex);
      }
    }
  }
```

```
  for(unsigned long speciesIndex=0; speciesIndex < d_species.size();
speciesIndex++) {
    for(unsigned long i=0; i<speciesAffectors.getPtr(speciesIndex)-
>size(); i++) {
      unsigned long executedReactionIndex =
speciesAffectors.getPtr(speciesIndex)->get(i);
      for(unsigned long j=0;
j<speciesDependencies.getPtr(speciesIndex)->size(); j++) {
                  unsigned long affectedReactionIndex =
speciesDependencies.getPtr(speciesIndex)->get(j);
       if (!d_reactionDependencies.getPtr(executedReactionIndex)-
>contains(affectedReactionIndex)) {
          d_reactionDependencies.getPtr(executedReactionIndex)-
>add(affectedReactionIndex);
                                    }
      }
    }
  }

  d_areReactionDependenciesBuilt = true;
}

Vector< Vector<unsigned long> >* Model::getReactionDependencies() {
  if (!d_areReactionDependenciesBuilt) {
    buildReactionDependencies();
  }
  return &d_reactionDependencies;
}

void Model::output() {
  for(long i=0; i<d_species.size(); i++) {
    cout << "s" << i << " = " << d_species.get(i) << endl;
  }
  cout << endl;

  for(long i=0; i<d_reactions.size(); i++) {
    d_reactions.getPtr(i)->output();
    cout << endl;
  }
  cout << endl;

  cout << "output";
  for(long i=0; i<d_outputs.size(); i++) {
    cout << " s" << d_outputs.get(i);
  }
  cout << endl;
}
```

Model.h

```
#ifndef MODEL_H
#define MODEL_H
```

```
#include "Reaction.h"
#include <fstream>

class Model {
 public:
  Model() { };
  Model(char *filename);
  Vector<long>* getSpecies() { return &d_species; };
  Vector<Reaction>* getReactions() { return &d_reactions; };
  Vector<unsigned long>* getOutputs() { return &d_outputs; };
  Vector< Vector<unsigned long> >* getReactionDependencies();
  void output();

 private:
  Vector<long> d_species;
  Vector<Reaction> d_reactions;
  Vector<unsigned long> d_outputs;
  Vector< Vector<unsigned long> > d_reactionDependencies;
  void buildReactionDependencies();
  bool d_areReactionDependenciesBuilt;
};

#endif
```

Reaction.cpp

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include "Exception.h"
#include "Reaction.h"

using namespace std;

void Reaction::read(InputFile &in) {
  unsigned int i, j;

  d_reactants.resize(in.getUnsignedInt());
  for(i=0; i<d_reactants.size(); i++) {
    ReactionElement* reactant = d_reactants.getPtr(i);
    reactant->setCoefficient(in.getUnsignedInt());
    reactant->setSpeciesIndex(in.getUnsignedLong());
  }

  d_products.resize(in.getUnsignedInt());
  for(i=0; i<d_products.size(); i++) {
    ReactionElement* product = d_products.getPtr(i);
    product->setCoefficient(in.getUnsignedInt());
    product->setSpeciesIndex(in.getUnsignedLong());
  }

  d_rateConstant = in.getDouble();
  if (d_rateConstant <= 0.0) {
    throw Exception("Reaction","init","Invalid rate constant value.");
```

```
  }

  d_scaledRate = d_rateConstant;
  for(i=0; i<d_reactants.size(); i++) {
    for(j=d_reactants.getPtr(i)->getCoefficient(); j>=2; j--) {
      d_scaledRate /= (double)j;
    }
  }
}

void Reaction::execute(Vector<long>* species) {
  unsigned int i;

  for(i=0; i<d_reactants.size(); i++) {
    unsigned long speciesIndex = d_reactants.getPtr(i)-
>getSpeciesIndex();
    unsigned long coefficient = d_reactants.getPtr(i)-
>getCoefficient();
    species->set(speciesIndex,species->get(speciesIndex) -
coefficient);
  }

  for(i=0; i<d_products.size(); i++) {
    unsigned long speciesIndex = d_products.getPtr(i)-
>getSpeciesIndex();
    unsigned long coefficient = d_products.getPtr(i)->getCoefficient();
    species->set(speciesIndex,species->get(speciesIndex) +
coefficient);
  }
}

void Reaction::unexecute(Vector<long>* species) {
  unsigned int i;
  for(i=0; i<d_reactants.size(); i++) {
    unsigned long speciesIndex = d_reactants.getPtr(i)-
>getSpeciesIndex();
    unsigned long coefficient = d_reactants.getPtr(i)-
>getCoefficient();
    species->set(speciesIndex,species->get(speciesIndex) +
coefficient);
  }
  for(i=0; i<d_products.size(); i++) {
    unsigned long speciesIndex = d_products.getPtr(i)-
>getSpeciesIndex();
    unsigned long coefficient = d_products.getPtr(i)->getCoefficient();
    species->set(speciesIndex,species->get(speciesIndex) -
coefficient);
  }
}

double Reaction::getPropensity(Vector<long>* species) const {
  unsigned int i,j;
  double propensity = d_scaledRate;
  for(i=0; i<d_reactants.size(); i++) {
    ReactionElement *reactant = d_reactants.getPtr(i);
```

```
    for(j=0; j<reactant->getCoefficient(); j++) {
      if (species->get(reactant->getSpeciesIndex()) - j == 0) return
0.0;
      propensity *= (double)(species->get(reactant->getSpeciesIndex())
- j);
    }
  }
  return propensity;
}

bool Reaction::affectsSpecies(unsigned long speciesIndex) const {
  int delta = 0;
  unsigned int i;

  for(i=0; i<d_reactants.size(); i++) {
    if (d_reactants.getPtr(i)->getSpeciesIndex() == speciesIndex) {
      delta -= d_reactants.getPtr(i)->getCoefficient();
      break;
    }
  }
  for(i=0; i<d_products.size(); i++) {
    if (d_products.getPtr(i)->getSpeciesIndex() == speciesIndex) {
      delta += d_products.getPtr(i)->getCoefficient();
      break;
    }
  }

  if (delta == 0) {
    return false;
  } else {
    return true;
  }
}

void Reaction::output() const {
  if (d_reactants.size() == 0) {
    cout << "* ";
  } else {
    for(long i=0; i<d_reactants.size(); i++) {
      if (d_reactants.getPtr(i)->getCoefficient() > 1) {
        cout << d_reactants.getPtr(i)->getCoefficient() << " ";
      }
      cout << "s" << d_reactants.getPtr(i)->getSpeciesIndex() << " ";
    }
  }
  cout << "=>";
  if (d_products.size() == 0) {
    cout << " *";
  } else {
    for(long i=0; i<d_products.size(); i++) {
      if (d_products.getPtr(i)->getCoefficient() > 1) {
        cout << " " << d_products.getPtr(i)->getCoefficient();
      }
      cout << " s" << d_products.getPtr(i)->getSpeciesIndex();
    }
```

```
  }
  cout << " k=" << d_rateConstant;
}
```

ReactionElement.cpp

```
#include "ReactionElement.h"
#include "Exception.h"

ReactionElement::ReactionElement(unsigned int coefficient, unsigned
long speciesIndex) {
  setCoefficient(coefficient);
  setSpeciesIndex(speciesIndex);
}
```

ReactionElement.h

```
#ifndef REACTIONELEMENT_H
#define REACTIONELEMENT_H

#include "Exception.h"

class ReactionElement {
 public:
  ReactionElement() { };
  ReactionElement(unsigned int coefficient, unsigned long
speciesIndex);
  unsigned int getCoefficient() const { return d_coefficient; };
  unsigned long getSpeciesIndex() const { return d_speciesIndex; };

  void setCoefficient(unsigned int coefficient) {
#ifdef DEBUG
    if (coefficient < 0) throw
Exception("ReactionElement","setCoefficient","Invalid coefficient.");
#endif
    d_coefficient = coefficient;
  };

  void setSpeciesIndex(unsigned long speciesIndex) {
#ifdef DEBUG
    if (speciesIndex < 0) throw
Exception("ReactionElement","setSpeciesIndex","Invalid species
index.");
#endif
    d_speciesIndex = speciesIndex;
  };

 private:
  ReactionElement(ReactionElement &element) { };
  unsigned int d_coefficient;
  unsigned long d_speciesIndex;
};
```

```
#endif
```

Reaction.h

```
#ifndef REACTION_H
#define REACTION_H

#include "ReactionElement.h"
#include "InputFile.h"
#include "Vector.h"

class Reaction {
 private:
  Vector<ReactionElement> d_reactants;
  Vector<ReactionElement> d_products;
  double d_rateConstant;
  double d_scaledRate;

 public:
  Reaction() { };
  void read(InputFile &in);

  Vector<ReactionElement>* getReactants() { return &d_reactants; };
  Vector<ReactionElement>* getProducts() { return &d_products; };
  double getRateConstant() const { return d_rateConstant; } ;

  void execute(Vector<long>* species);
  void unexecute(Vector<long>* species);
  double getPropensity(Vector<long>* species) const;
  bool affectsSpecies(unsigned long speciesIndex) const;

  void output() const;

 private:
  Reaction(Reaction &reaction) { };
};

#endif
```

Vector.h

```
#ifndef VECTOR_H
#define VECTOR_H

#include "Exception.h"
#include <string.h>

template <class T>
class Vector {
 public:
  Vector() {
    d_size = 0;
```

```
    d_data = NULL;
  }

  Vector(unsigned long size) {
    d_size = 0;
    d_data = NULL;
    resize(size);
  }

  ~Vector() {
  }

  void resize(unsigned long size) {
    if (d_size == 0) {
      d_size = size;
      d_data = new T[size];
      if (d_data == NULL) {
          throw Exception("Vector","resize","Out of memory.");
      }
    } else {
      unsigned long oldSize = d_size;
      T* oldData = d_data;
      d_size = size;
      d_data = new T[size];
      if (d_data == NULL) {
          throw Exception("Vector","resize","Out of memory.");
      }
      memcpy(d_data,oldData,sizeof(T)*oldSize);
      delete [] oldData;
    }
  }

  void add(T value) {
    resize(d_size+1);
    set(d_size-1,value);
  }

  void swap(unsigned long index1, unsigned long index2) {
#ifdef DEBUG
    if ((index1 < 0) || (index1 >= d_size)) throw
Exception("Vector","get","Index out of range.");
    if ((index2 < 0) || (index2 >= d_size)) throw
Exception("Vector","get","Index out of range.");
#endif
    T temp = d_data[index1];
    d_data[index1] = d_data[index2];
    d_data[index2] = temp;
  }

  T get(unsigned long index) const {
#ifdef DEBUG
    if ((index < 0) || (index >= d_size)) throw
Exception("Vector","get","Index out of range.");
#endif
    return d_data[index];
```

```
  }

  T* getPtr(unsigned long index) const {
#ifdef DEBUG
    if ((index < 0) || (index >= d_size)) throw
Exception("Vector","getPtr","Index out of range.");
#endif
    return &(d_data[index]);
  }

  void set(unsigned long index, T value) {
#ifdef DEBUG
    if ((index < 0) || (index >= d_size)) throw
Exception("Vector","set","Index out of range.");
#endif
    d_data[index] = value;
  }

  unsigned long size() const {
    return d_size;
  }

  bool contains(T value) {
    for(unsigned long i=0; i<d_size; i++) {
      if (value == d_data[i]) {
                               return true;
      }
    }
    return false;
  }

  long find(T value) {
    for(unsigned long i=0; i<d_size; i++) {
      if (value == d_data[i]) {
                                return i;
      }
    }
    return -1;
  }

 private:
  unsigned long d_size;
  T* d_data;
};

#endif
```

Serial Source Code

The following source code is implements the serial versions of the stochastic simulation algorithms and is located in the "serial/" directory during compilation.

DirectMethod.cpp

```
#include "DirectMethod.h"
#include <stdlib.h>
#include <math.h>
#include "../common/Exception.h"
#include "../common/Infinity.h"

DirectMethod::DirectMethod() {
  uniformRandomBuffer = NULL;
  exponentialRandomBuffer = NULL;
}

DirectMethod::~DirectMethod() {
  if (uniformRandomBuffer != NULL) {
    delete uniformRandomBuffer;
  }
  if (exponentialRandomBuffer != NULL) {
    delete exponentialRandomBuffer;
  }
}

void DirectMethod::init(Model *model, Random *random, double
*currentTime, Vector<long> *speciesVector) {
#ifdef DEBUG
  if (model == NULL) {
    throw Exception("DirectMethod","init","model == NULL");
  }
  if (random == NULL) {
    throw Exception("DirectMethod","init","random == NULL");
  }
  if (currentTime == NULL) {
    throw Exception("DirectMethod","init","currentTime == NULL");
  }
#endif
  this->model = model;
  this->currentTime = currentTime;

  uniformRandomBuffer = new
RandomBuffer(random,100000,RANDOM_BUFFER_UNIFORM);
  if (uniformRandomBuffer == NULL) {
    throw Exception("DirectMethod","init","Out of memory.");
  }

  exponentialRandomBuffer = new
RandomBuffer(random,100000,RANDOM_BUFFER_EXPONENTIAL);
  if (exponentialRandomBuffer == NULL) {
    throw Exception("DirectMethod","init","Out of memory.");
```

```
  }

  propensities.resize(model->getReactions()->size());
  species = speciesVector;

  reactionCounts.resize(model->getReactions()->size());
  for(int i=0; i<reactionCounts.size(); i++) {
    reactionCounts.set(i,0);
  }
}

void DirectMethod::step() {
#ifdef DEBUG
  if (model == NULL) {
    throw Exception("DirectMethod","step","model == NULL");
  }
  if (uniformRandomBuffer == NULL) {
    throw Exception("DirectMethod","step","uniformRandomBuffer ==
NULL");
  }
  if (exponentialRandomBuffer == NULL) {
    throw Exception("DirectMethod","step","exponentialRandomBuffer ==
NULL");
  }
#endif

  totalPropensity = 0.0;
  unsigned long i;

  for(i=0; i<model->getReactions()->size(); i++) {
    propensities.set(i,model->getReactions()->getPtr(i)-
>getPropensity(species));
    totalPropensity += propensities.get(i);
  }

  double scaledRand = uniformRandomBuffer->getNumber() *
totalPropensity;
  reactionIndex = 0;
  for(i=0; i<model->getReactions()->size(); i++) {
    if (propensities.get(i) != 0.0) {
      reactionIndex = i;
      scaledRand -= propensities.get(i);
      if (scaledRand <= 0.0) break;
    }
  }

  if (totalPropensity == 0.0) {
    *currentTime = MYINFINITY;
  } else {
    *currentTime = *currentTime + exponentialRandomBuffer->getNumber()
/ totalPropensity;
  }
}

void DirectMethod::execute() {
```

```
#ifdef DEBUG
  if (model == NULL) {
    throw Exception("DirectMethod","execute","model == NULL");
  }
  if ((reactionIndex < 0) || (reactionIndex >= model-
>getReactionCount())) {
    throw Exception("DirectMethod","execute","reactionIndex out of
range");
  }
#endif

reactionCounts.set(reactionIndex,reactionCounts.get(reactionIndex)+1);
  model->getReactions()->getPtr(reactionIndex)->execute(species);
}

Vector<unsigned long>* DirectMethod::getReactionCounts() {
  return &reactionCounts;
}
```

DirectMethod.h

```
#ifndef DIRECTMETHOD_H
#define DIRECTMETHOD_H

#include "../common/Model.h"
#include "../common/Vector.h"
#include "RandomBuffer.h"

class DirectMethod {
 private:
  Model *model;
  double *currentTime;
  unsigned long reactionIndex;
  double totalPropensity;
  RandomBuffer *uniformRandomBuffer;
  RandomBuffer *exponentialRandomBuffer;
  Vector<double> propensities;
  Vector<long> *species;
  Vector<unsigned long> reactionCounts;

 public:
  DirectMethod();
  ~DirectMethod();
  void init(Model *model, Random *random, double *currentTime,
Vector<long> *speciesVector);
  void step();
  void execute();
  Vector<unsigned long>* getReactionCounts();
};

#endif
```

ess.cpp

```
#include <sys/time.h>
#include <iostream>
#include <math.h>
#include "../common/Model.h"
#include "../common/Exception.h"
#include "../common/Infinity.h"
#include "Parameters.h"
#include "Random.h"
#ifdef FIRST
#include "FirstReactionMethod.h"
#endif
#ifdef DIRECT
#include "DirectMethod.h"
#endif
#ifdef OPTIMIZEDDIRECT
#include "OptimizedDirectMethod.h"
#endif
#ifdef SORTINGDIRECT
#include "SortingDirectMethod.h"
#endif
#ifdef NEXT
#include "NextReactionMethod.h"
#endif
#ifdef ADAPTIVE
#include "AdaptiveMethod.h"
#endif

using namespace std;

void printHelpMessage(char *executableName) {
  cout << "usage: " << executableName << " <parameters>" << endl;
  cout << " valid parameters:" << endl;
  cout << "  -i <filename> or --input" << endl;
  cout << "     Sets input model filename (required)." << endl;
  cout << "  -e <double> or --endtime" << endl;
  cout << "     Sets simulator end time (required if reaction limit not
set)." << endl;
  cout << "  -r <integer> or --reactionlimit" << endl;
  cout << "     Sets max number of reactions to execute (required if
end time not set)." << endl;
  cout << "  -p <double> or --printinterval" << endl;
  cout << "     Tells simulator how often to report output (optional)."
<< endl;
  cout << "  -o on|off or --output" << endl;
  cout << "     Turns output on or off (optional)." << endl;
  cout << "  -s <integer> or --seed" << endl;
  cout << "     Sets the  random seed (optional)." << endl;
  cout << "  -prxn on|off or --profilereactions" << endl;
  cout << "     Turns reaction profiling (reaction.profile) on or off
(optional)." << endl;
  cout << "  -prun on|off or --profilerun" << endl;
  cout << "     Turns run profiling (run.profile) on or off
(optional)." << endl;
  cout << "  -h or --help" << endl;
```

```
  cout << "     Prints this message." << endl;
}

void printState(double time, Model *model,Vector<long> *species) {
  cout << time;
  for(unsigned long i=0; i<model->getOutputs()->size(); i++) {
    cout << " ";
    cout << species->get(model->getOutputs()->get(i));
  }
  cout << endl;
}

int main(int argc, char **argv) {
  try {
    timeval before;
    timeval after;
    gettimeofday(&before,NULL);

    Parameters *parameters;
    try {
      parameters = new Parameters(argc,argv);
    } catch (Exception e) {
      cerr << e.getMessage() << endl;
      cerr << "type " << argv[0] << " -h for help." << endl;
      return -1;
    }
    if (parameters == NULL) {
      throw Exception("main","main","Out of memory.");
    }

    if (parameters->isHelpSet()) {
      printHelpMessage(argv[0]);
      return 1;
    }

    Model model(parameters->getInputFilename());
    Random random(parameters->getSeed());

    Vector<long> species;
    species.resize(model.getSpecies()->size());
    for(unsigned int i=0; i<model.getSpecies()->size(); i++) {
      species.set(i,model.getSpecies()->get(i));
    }

    double currentTime = 0.0;
    unsigned long reactionCount = 0;
    double nextPrintTime;
    if (parameters->isPrintIntervalSet()) {
      nextPrintTime = parameters->getPrintInterval();
    } else {
      nextPrintTime = MYINFINITY;
    }

#ifdef FIRST
    FirstReactionMethod algorithm;
```

```
#endif
#ifdef DIRECT
    DirectMethod algorithm;
#endif
#ifdef OPTIMIZEDDIRECT
    OptimizedDirectMethod algorithm;
#endif
#ifdef NEXT
    NextReactionMethod algorithm;
#endif
#ifdef ADAPTIVE
    AdaptiveMethod algorithm;
#endif
#ifdef SORTINGDIRECT
    SortingDirectMethod algorithm;
#endif

    double finalPrintTime;
    if (parameters->isPrintIntervalSet()) {
      finalPrintTime = parameters->getEndTime() + parameters-
>getPrintInterval()*0.01;
    }
    algorithm.init(&model,&random,&currentTime,&species);
    if (parameters->isOutputOn())
printState(currentTime,&model,&species);
    while (true) {
      if (parameters->isReactionLimitSet() && (reactionCount >
parameters->getReactionLimit())) break;
      algorithm.step();
      if (parameters->isOutputOn()) {
                if (parameters->isPrintIntervalSet()) {
        while ((nextPrintTime < finalPrintTime) && (currentTime >
nextPrintTime)) {
                 printState(nextPrintTime,&model,&species);
              nextPrintTime += parameters->getPrintInterval();
                                    }
                                   }
      }
      if (parameters->isEndTimeSet() && (currentTime > parameters-
>getEndTime())) break;
      if (currentTime == MYINFINITY) break;
      algorithm.execute();
      if ((!parameters->isPrintIntervalSet()) && (parameters-
>isOutputOn()))
                printState(currentTime,&model,&species);
      reactionCount++;
    }

    gettimeofday(&after,NULL);
    double totalTime = after.tv_sec - before.tv_sec;
    totalTime += ((double)after.tv_usec)/1000000.0;
    totalTime -= ((double)before.tv_usec)/1000000.0;


    if (parameters->isRunProfileOn()) {
```

```
      ofstream out("run.profile");
      out << reactionCount << " " << totalTime << endl;
    }

    if (parameters->isReactionProfileOn()) {
      ofstream rc("reaction.profile");
      Vector<unsigned long> *reactionCounts =
algorithm.getReactionCounts();
      for(unsigned long i=0; i<reactionCounts->size(); i++) {
                  rc << reactionCounts->get(i) << endl;
      }
      rc.close();
    }

    delete parameters;
    return 1;
  } catch (Exception e) {
#ifdef DEBUG
    cout << e.getClass() << endl;
    cout << e.getFunction() << endl;
#endif
    cout << e.getMessage() << endl;
    return -1;
  }
}
```

FirstReactionMethod.cpp

```
#include <math.h>
#include "FirstReactionMethod.h"
#include "../common/Exception.h"
#include "../common/Infinity.h"

FirstReactionMethod::FirstReactionMethod() {
  exponentialRandomBuffer = NULL;
  initialized = false;
}

FirstReactionMethod::~FirstReactionMethod() {
  if (exponentialRandomBuffer != NULL) {
    delete exponentialRandomBuffer;
  }
}

void FirstReactionMethod::init(Model *model, Random *random, double
*currentTime, Vector<long> *speciesVector) {
#ifdef DEBUG
  if (model == NULL) {
    throw Exception("FirstReactionMethod","init","model == NULL");
  }
  if (random == NULL) {
    throw Exception("FirstReactionMethod","init","random == NULL");
  }
  if (currentTime == NULL) {
```

```
    throw Exception("FirstReactionMethod","init","currentTime ==
NULL");
  }
  stepShouldBeNext = true;
#endif
  this->model = model;
  this->currentTime = currentTime;
  this->species = speciesVector;
  exponentialRandomBuffer = new
RandomBuffer(random,100000,RANDOM_BUFFER_EXPONENTIAL);
  if (exponentialRandomBuffer == NULL) {
    throw Exception("DirectMethod","init","Out of memory.");
  }

  reactionCounts.resize(model->getReactions()->size());
  for(int i=0; i<reactionCounts.size(); i++) {
    reactionCounts.set(i,0);
  }

  initialized = true;
}

void FirstReactionMethod::step() {
#ifdef DEBUG
  if (!initialized) {
    throw Exception("FirstReactionMethod","step","step called before
initialization");
  }
  if (!stepShouldBeNext) {
    throw Exception("FirstReactionMethod","step","step called out of
sequence");
  }
  stepShouldBeNext = false;
#endif
  initialized = true;
  double time = MYINFINITY;
  reactionIndex = 0;
  for(unsigned long i=0; i<model->getReactions()->size(); i++) {
    double temp = exponentialRandomBuffer->getNumber() /
      model->getReactions()->getPtr(i)->getPropensity(species);
    if (temp < time) {
      time = temp;
      reactionIndex = i;
    }
  }
  *currentTime += time;
}

void FirstReactionMethod::execute() {
#ifdef DEBUG
  if (!initialized) {
    throw Exception("FirstReactionMethod","execute","execute called
before initialization");
  }
  if (stepShouldBeNext) {
```

```
    throw Exception("FirstReactionMethod","execute","execute called out
of sequence");
  }
  stepShouldBeNext = true;
#endif

reactionCounts.set(reactionIndex,reactionCounts.get(reactionIndex)+1);
  model->getReactions()->getPtr(reactionIndex)->execute(species);
}

Vector<unsigned long>* FirstReactionMethod::getReactionCounts() {
  return &reactionCounts;
}
```

FirstReactionMethod.h

```
#ifndef FIRSTREACTIONMETHOD_H
#define FIRSTREACTIONMETHOD_H

#include "../common/Vector.h"
#include "../common/Model.h"
#include "RandomBuffer.h"

class FirstReactionMethod {
 private:
  Model *model;
  double *currentTime;
  unsigned long reactionIndex;
  RandomBuffer *exponentialRandomBuffer;
  bool initialized;
  bool stepShouldBeNext;
  Vector<long> *species;
  Vector<unsigned long> reactionCounts;

 public:
  FirstReactionMethod();
  ~FirstReactionMethod();
  void init(Model *model, Random *random, double *currentTime,
Vector<long> *species);
  void step();
  void execute();
  Vector<unsigned long>* getReactionCounts();
};

#endif
```

makefile

```
OBJS = FirstReactionMethod.o DirectMethod.o NextReactionMethod.o \
            OptimizedDirectMethod.o Random.o RandomBuffer.o \
               Parameters.o SortingDirectMethod.o MinHeap.o

CXX = c++
```

```
CPPFLAGS = -O3


all: first direct next optimized_direct sorting_direct reaction_sort

first: ${OBJS} ess.cpp
	$(CXX) ${CPPFLAGS} -o first ess.cpp Random.o RandomBuffer.o
Parameters.o FirstReactionMethod.o ../common/*.o -DFIRST

direct: ${OBJS} ess.cpp
	$(CXX) ${CPPFLAGS} -o direct ess.cpp Random.o RandomBuffer.o
Parameters.o DirectMethod.o ../common/*.o -DDIRECT

next: ${OBJS} ess.cpp
 $(CXX) ${CPPFLAGS} -o next ess.cpp Random.o RandomBuffer.o Parameters.o
MinHeap.o NextReactionMethod.o ../common/*.o -DNEXT

optimized_direct: ${OBJS} ess.cpp
 $(CXX) ${CPPFLAGS} -o optimized_direct ess.cpp Random.o RandomBuffer.o
Parameters.o OptimizedDirectMethod.o ../common/*.o -DOPTIMIZEDDIRECT

sorting_direct: ${OBJS} ess.cpp
  $(CXX) ${CPPFLAGS} -o sorting_direct ess.cpp Random.o RandomBuffer.o
Parameters.o SortingDirectMethod.o ../common/*.o -DSORTINGDIRECT

reaction_sort: ${OBJS} ReactionSort.cpp
   $(CXX) ${CPPFLAGS} -o reaction_sort ReactionSort.cpp ../common/*.o

clean:
   -rm *.o *~
   -rm first direct next optimized_direct sorting_direct reaction_sort
```

MinHeap.cpp

```
#include "MinHeap.h"
#include "../common/Exception.h"
#include <iostream>
#include <string.h>

using namespace std;

MinHeap::MinHeap(unsigned long size) {
  d_top = NULL;
  d_nodes = new Node*[size];
  if (d_nodes == NULL) {
    throw Exception("MinHeap","MinHeap","Out of memory");
  }
  d_nodeCount = size;
  for(unsigned long i=0; i<d_nodeCount; i++) {
    d_nodes[i] = new Node();
    if (d_nodes[i] == NULL) {
      throw Exception("MinHeap","MinHeap","Out of memory");
    }
    d_nodes[i]->id = i;
```

```
    if (d_top == NULL) {
      d_top = d_nodes[i];
    } else {
      add(d_nodes[i],d_top);
    }
  }
}

MinHeap::~MinHeap() {
  if (d_top != NULL) delete d_top;
  if (d_nodes != NULL) delete [] d_nodes;
}

void MinHeap::output() {
  output(d_top);
}

void MinHeap::output(Node *n) {
  if (n != NULL) {
    cout << n->id << "(";
    output(n->left);
    cout << ",";
    output(n->right);
    cout << ")";
  }
}

void MinHeap::update(unsigned long id, double value) {
#ifdef DEBUG
  if ((id < 0) || (id >= d_nodeCount)) {
    throw Exception("MinHeap","update","Index out of range");
  }
#endif
  d_nodes[id]->value = value;
  update_aux(d_nodes[id]);
}

unsigned long MinHeap::getMin() const {
#ifdef DEBUG
  if (d_top == NULL) {
    throw Exception("MinHeap","getMin","d_top == NULL");
  }
#endif
  return d_top->id;
}

void MinHeap::update_aux(Node *n) {
#ifdef DEBUG
  if (n == NULL) {
    throw Exception("MinHeap","update_aux","n == NULL");
  }
#endif
  if ((n->parent != NULL) && (n->value < n->parent->value)) {
    swap(n->id,n->parent->id);
```

```
    update_aux(n->parent);
  }

  Node *minchild = NULL;
  if ((n->left != NULL) && (n->right != NULL)) {
    if (n->left->value < n->right->value) {
      minchild = n->left;
    } else {
      minchild = n->right;
    }
  } else if (n->left != NULL) {
    minchild = n->left;
  } else if (n->right != NULL) {
    minchild = n->right;
  } else {
    return;
  }

  if (minchild->value < n->value) {
    swap(minchild->id,n->id);
    update_aux(minchild);
  }
}

unsigned long MinHeap::nodeCount(Node *tree) const {
  if (tree == NULL) {
    return 0;
  } else {
    return 1+nodeCount(tree->left)+nodeCount(tree->right);
  }
}

void MinHeap::add(Node *n, Node *tree) {
#ifdef DEBUG
  if (n == NULL) {
    throw Exception("MinHeap","add","n == NULL");
  }
  if (tree == NULL) {
    throw Exception("MinHeap","add","tree == NULL");
  }
#endif

  if (tree->left == NULL) {
    tree->left = n;
    n->parent = tree;

    Node *temp = tree;
    while (temp != NULL) {
      temp->treeSize++;
      temp = temp->parent;
    }

    return;
  }
  if (tree->right == NULL) {
```

```
    tree->right = n;
    n->parent = tree;

    Node *temp = tree;
    while (temp != NULL) {
      temp->treeSize++;
      temp = temp->parent;
    }

    return;
  }

  unsigned long leftCount = tree->left->treeSize;
  unsigned long rightCount = tree->right->treeSize;

  if (leftCount <= rightCount) {
    add(n,tree->left);
  } else {
    add(n,tree->right);
  }
}

void MinHeap::swap(unsigned long id1, unsigned long id2) {
#ifdef DEBUG
  if ((id1 < 0) || (id1 >= d_nodeCount)) {
    throw Exception("MinHeap","swap","id1 out of range");
  }
  if ((id2 < 0) || (id2 >= d_nodeCount)) {
    throw Exception("MinHeap","swap","id2 out of range");
  }
  if (d_nodes == NULL) {
    throw Exception("MinHeap","swap","d_nodes == NULL");
  }
#endif
  double temp = d_nodes[id1]->value;
  d_nodes[id1]->value = d_nodes[id2]->value;
  d_nodes[id2]->value = temp;
  d_nodes[id1]->id = id2;
  d_nodes[id2]->id = id1;

  Node *n = d_nodes[id1];
  d_nodes[id1] = d_nodes[id2];
  d_nodes[id2] = n;
}
```

MinHeap.h

```
#ifndef MINHEAP_H
#define MINHEAP_H

#include <stdlib.h>

class MinHeap {
 private:
```

```
  class Node;

 public:
  MinHeap(unsigned long size);
  ~MinHeap();
  void update(unsigned long id, double value);
  unsigned long getMin() const;
  void output();

 private:
  void add(Node *n, Node *tree);
  void swap(unsigned long id1, unsigned long id2);
  void update_aux(Node *n);
  unsigned long nodeCount(Node *tree) const;
  void output(Node *n);

 private:
  Node *d_top;
  Node **d_nodes;
  unsigned long d_nodeCount;
};

class MinHeap::Node {
 public:
  Node *left;
  Node *right;
  Node *parent;
  double value;
  unsigned long id;
  unsigned long treeSize;

  Node() {
    left = NULL;
    right = NULL;
    parent = NULL;
    value = 0.0;
    id = 0;
    treeSize = 1;
  }

  ~Node() {
    if (left != NULL) delete left;
    if (right != NULL) delete right;
  }
};

#endif
```

NextReactionMethod.cpp

```
#include "NextReactionMethod.h"
#include "../common/Exception.h"
#include <stdlib.h>
```

```
NextReactionMethod::NextReactionMethod() {
  minHeap = NULL;
  exponentialRandomBuffer = NULL;
}

NextReactionMethod::~NextReactionMethod() {
  if (minHeap != NULL) delete minHeap;
  if (exponentialRandomBuffer != NULL) delete exponentialRandomBuffer;
}

void NextReactionMethod::init(Model *model, Random *random, double
*currentTime, Vector<long> *speciesVector) {
#ifdef DEBUG
  if (model == NULL) {
    throw Exception("NextReactionMethod","init","model == NULL");
  }
  if (random == NULL) {
    throw Exception("NextReactionMethod","init","random == NULL");
  }
  if (currentTime == NULL) {
    throw Exception("NextReactionMethod","init","currentTime == NULL");
  }
#endif
  this->model = model;
  this->currentTime = currentTime;
  this->species = speciesVector;

  exponentialRandomBuffer = new
RandomBuffer(random,100000,RANDOM_BUFFER_EXPONENTIAL);
  if (exponentialRandomBuffer == NULL) {
    throw Exception("NextReactionMethod","init","Out of memory.");
  }

  propensities.resize(model->getReactions()->size());
  putativeTimes.resize(model->getReactions()->size());

  minHeap = new MinHeap(model->getReactions()->size());
  if (minHeap == NULL) {
    throw Exception("NextReactionMethod","init","Out of memory.");
  }

  for(unsigned long i=0; i<model->getReactions()->size(); i++) {
    propensities.set(i,model->getReactions()->getPtr(i)-
>getPropensity(species));
    putativeTimes.set(i,random->getExponential() /
propensities.get(i));
    minHeap->update(i,putativeTimes.get(i));
  }

  reactionCounts.resize(model->getReactions()->size());
  for(int i=0; i<reactionCounts.size(); i++) {
    reactionCounts.set(i,0);
  }
}
```

```
void NextReactionMethod::reinit() {
  for(unsigned long i=0; i<model->getReactions()->size(); i++) {
    propensities.set(i,model->getReactions()->getPtr(i)-
>getPropensity(species));
    putativeTimes.set(i,exponentialRandomBuffer->getNumber() /
propensities.get(i));
    minHeap->update(i,putativeTimes.get(i));
  }
}

void NextReactionMethod::step() {
  reactionIndex = minHeap->getMin();

reactionCounts.set(reactionIndex,reactionCounts.get(reactionIndex)+1);
  *currentTime = putativeTimes.get(reactionIndex);
}

void NextReactionMethod::execute() {
  model->getReactions()->getPtr(reactionIndex)->execute(species);
  Vector<unsigned long> *dependencies = model-
>getReactionDependencies()->getPtr(reactionIndex);
  for(unsigned long i=0; i<dependencies->size(); i++) {
    unsigned long index = dependencies->get(i);
    double oldPropensity = propensities.get(index);
    propensities.set(index,model->getReactions()->getPtr(index)-
>getPropensity(species));
    if (index != reactionIndex) {
      if (oldPropensity == 0.0) {
     putativeTimes.set(index,*currentTime + exponentialRandomBuffer-
>getNumber()/propensities.get(index));
      } else {
         putativeTimes.set(index,*currentTime + oldPropensity /
propensities.get(index) * (putativeTimes.get(index) - *currentTime));
      }
      minHeap->update(index,putativeTimes.get(index));
    }
  }
  putativeTimes.set(reactionIndex,*currentTime +
exponentialRandomBuffer->getNumber()/propensities.get(reactionIndex));
  minHeap->update(reactionIndex,putativeTimes.get(reactionIndex));
}

unsigned long NextReactionMethod::getCurrentUpdateFactor() {
  return model->getReactionDependencies()->getPtr(reactionIndex)-
>size();
}
```

NextReactionMethod.h

```
#ifndef NEXTREACTIONMETHOD_H
#define NEXTREACTIONMETHOD_H

#include "../common/Model.h"
#include "RandomBuffer.h"
```

```
#include "MinHeap.h"
#include "../common/Vector.h"

class NextReactionMethod {
 private:
  Model *model;
  RandomBuffer *exponentialRandomBuffer;
  double *currentTime;
  unsigned long reactionIndex;
  Vector<double> propensities;
  Vector<double> putativeTimes;
  MinHeap *minHeap;
  Vector<long> *species;
  Vector<unsigned long> reactionCounts;

 public:
  NextReactionMethod();
  ~NextReactionMethod();
  void init(Model *model, Random *random, double *currentTime,
Vector<long> *speciesVector);
  void reinit();
  void step();
  void execute();
  unsigned long getCurrentUpdateFactor();
  Vector<unsigned long>* getReactionCounts() { return &reactionCounts;
};
};

#endif
```

OptimizedDirectMethod.cpp

```
#include "OptimizedDirectMethod.h"
#include "../common/Exception.h"
#include <stdlib.h>
#include <iostream>

using namespace std;

OptimizedDirectMethod::OptimizedDirectMethod() {
  uniformRandomBuffer = NULL;
  exponentialRandomBuffer = NULL;
}

OptimizedDirectMethod::~OptimizedDirectMethod() {
  if (uniformRandomBuffer != NULL)
    delete uniformRandomBuffer;
  if (exponentialRandomBuffer != NULL)
    delete exponentialRandomBuffer;
}

void OptimizedDirectMethod::init(Model *model, Random *random, double
*currentTime, Vector<long>* species) {
```

```
#ifdef DEBUG
  if (model == NULL) {
    throw Exception("OptimizedDirectMethod","init","model == NULL");
  }
  if (random == NULL) {
    throw Exception("OptimizedDirectMethod","init","random == NULL");
  }
  if (currentTime == NULL) {
    throw Exception("OptimizedDirectMethod","init","currentTime ==
NULL");
  }
#endif

  this->model = model;
  this->currentTime = currentTime;
  this->species = species;
  totalPropensity = 0.0;

  uniformRandomBuffer = new
RandomBuffer(random,100000,RANDOM_BUFFER_UNIFORM);
  if (uniformRandomBuffer == NULL) {
    throw Exception("OptimizedDirectMethod","init","Out of memory.");
  }

  exponentialRandomBuffer = new
RandomBuffer(random,100000,RANDOM_BUFFER_EXPONENTIAL);
  if (exponentialRandomBuffer == NULL) {
    throw Exception("OptimizedDirectMethod","init","Out of memory.");
  }

  propensities.resize(model->getReactions()->size());

  reactionCounts.resize(model->getReactions()->size());
  for(int i=0; i<reactionCounts.size(); i++) {
    reactionCounts.set(i,0);
  }
  reinit();
}

void OptimizedDirectMethod::reinit() {
  totalPropensity = 0.0;
  for(unsigned long i=0; i<model->getReactions()->size(); i++) {
    propensities.set(i,model->getReactions()->getPtr(i)-
>getPropensity(species));
    totalPropensity += propensities.get(i);
  }
}

void OptimizedDirectMethod::step() {
  double scaledRand = uniformRandomBuffer->getNumber() *
totalPropensity;
  reactionIndex = 0;
  for(unsigned long i=0; i<model->getReactions()->size(); i++) {
    if (propensities.get(i) != 0.0) {
      reactionIndex = i;
```

```
      scaledRand -= propensities.get(i);
      if (scaledRand <= 0.0) break;
    }
  }

  double time = exponentialRandomBuffer->getNumber() / totalPropensity;
  *currentTime += time;
}

void OptimizedDirectMethod::execute() {
  model->getReactions()->getPtr(reactionIndex)->execute(species);

reactionCounts.set(reactionIndex,reactionCounts.get(reactionIndex)+1);
  Vector<unsigned long>* dependencies = model-
>getReactionDependencies()->getPtr(reactionIndex);
  for(unsigned long i=0; i<dependencies->size(); i++) {
    unsigned long index = dependencies->get(i);
    totalPropensity -= propensities.get(index);
    propensities.set(index,model->getReactions()->getPtr(index)-
>getPropensity(species));
    totalPropensity += propensities.get(index);
  }
  if (totalPropensity < 0) totalPropensity = 0.0;
}

unsigned long OptimizedDirectMethod::getCurrentUpdateFactor() {
  return model->getReactionDependencies()->getPtr(reactionIndex)-
>size();
}

Vector<unsigned long>* OptimizedDirectMethod::getReactionCounts() {
  return &reactionCounts;
}
```

OptimizedDirectMethod.h

```
#ifndef FASTDIRECTMETHOD_H
#define FASTDIRECTMETHOD_H

#include "../common/Model.h"
#include "RandomBuffer.h"

class OptimizedDirectMethod {
 private:
  Model *model;
  double *currentTime;
  unsigned long reactionIndex;
  Vector<double> propensities;
  Vector<long>* species;
  double totalPropensity;
  RandomBuffer *uniformRandomBuffer;
  RandomBuffer *exponentialRandomBuffer;
  Vector<unsigned long> reactionCounts;
```

```
 public:
  OptimizedDirectMethod();
  ~OptimizedDirectMethod();
  void init(Model *model, Random *random, double *currentTime,
                          Vector<long>* species);
  void reinit();
  void step();
  void execute();
  unsigned long getCurrentUpdateFactor();
  Vector<unsigned long>* getReactionCounts();
};

#endif
```

Parameters.cpp

```
#include "Parameters.h"
#include "../common/Exception.h"
#include <stdlib.h>
#include <string.h>
#include <iostream>

using namespace std;

Parameters::Parameters(int argc, char **argv) {
  if (argc < 1) {
    throw Exception("Parameters","Parameters","argc < 1");
  }
  if (argv == NULL) {
    throw Exception("Parameters","Parameters","argv == NULL");
  }
  d_inputFilename    = NULL;
  d_inputFilenameSet = false;
  d_endTime          = 0.0;
  d_endTimeSet       = false;
  d_printInterval    = 0.0;
  d_printIntervalSet = false;
  d_reactionLimit    = 0;
  d_reactionLimitSet = false;
  d_seed             = 1;
  d_seedSet          = false;
  d_output           = true;
  d_outputSet        = false;
  d_helpSet          = false;
  d_profileReactions    = false;
  d_profileReactionsSet = false;
  d_profileRun          = false;
  d_profileRunSet       = false;


  if (argc < 2) throw Exception("Parameters","Parameters","Invalid
argument list.");

  if (argc == 2) {
```

```
    if ((strcmp(argv[1],"--help") == 0) || (strcmp(argv[1],"-h") == 0))
{
      d_helpSet = true;
      return;
    } else {
      throw Exception("Parameters","Parameters","Invalid argument
list.");
    }
  }

  if ((argc % 2) != 1) {
    throw Exception("Parameters","Parameters","Invalid argument
list.");
  }

  for(int i=1; i<argc; i+=2) {
    if ((strcmp(argv[i],"--output") == 0) ||
                        (strcmp(argv[i],"-o") == 0)) {
      setOutput(argv[i+1]);
    } else if ((strcmp(argv[i],"--input") == 0) ||
                        (strcmp(argv[i],"-i") == 0)) {
      setInputFilename(argv[i+1]);
    } else if ((strcmp(argv[i],"--seed") == 0) ||
                        (strcmp(argv[i],"-s") == 0)) {
      setSeed(argv[i+1]);
    } else if ((strcmp(argv[i],"--reactionlimit") == 0) ||
                        (strcmp(argv[i],"-r") == 0)) {
      setReactionLimit(argv[i+1]);
    } else if ((strcmp(argv[i],"--endtime") == 0) ||
                        (strcmp(argv[i],"-e") == 0)) {
      setEndTime(argv[i+1]);
    } else if ((strcmp(argv[i],"--printinterval") == 0) ||
                        (strcmp(argv[i],"-p") == 0)) {
      setPrintInterval(argv[i+1]);
    } else if ((strcmp(argv[i],"--profilereactions") == 0) ||
                       (strcmp(argv[i],"-prxn") == 0)) {
      setProfileReactions(argv[i+1]);
    } else if ((strcmp(argv[i],"--profilerun") == 0) ||
                       (strcmp(argv[i],"-prun") == 0)) {
      setProfileRun(argv[i+1]);
    } else {
      throw Exception("Parameters","Parameters","Unknown parameter.");
    }
  }

  if (!d_inputFilenameSet)
    throw Exception("Parameters","Parameters","Input filename not
set.");
  if ((!d_endTimeSet) && (!d_reactionLimitSet))
    throw Exception("Parameters","Parameters","Both end time and
reaction limit are not set.  At least one must be set.");
}

void Parameters::setOutput(char *arg) {
  if (arg == NULL) {
```

```
    throw Exception("Parameters","setOutput","arg == NULL");
  }
  if (d_outputSet) throw Exception("Parameters","setOutput","'-o' or '-
-output' option specified multiple times.");
  if (strcmp(arg,"on") == 0) d_output = true;
  else if (strcmp(arg,"off") == 0) d_output = false;
  else throw Exception("Parameters","setOutput","Invalid output
parameter.  Must be 'on' or 'off'.");
  d_outputSet = true;
}

void Parameters::setProfileReactions(char *arg) {
  if (arg == NULL) {
    throw Exception("Parameters","setProfileReactions","arg == NULL");
  }
  if (d_profileReactionsSet) throw
Exception("Parameters","setProfileReactions","'-prxn' or '--
profilereactions' option specified multiple times.");
  if (strcmp(arg,"on") == 0) d_profileReactions = true;
  else if (strcmp(arg,"off") == 0) d_profileReactions = false;
  else throw Exception("Parameters","setProfileReactions","Invalid
profile reactions parameter.  Must be 'on' or 'off'.");
  d_profileReactionsSet = true;
}

void Parameters::setProfileRun(char *arg) {
  if (arg == NULL) {
    throw Exception("Parameters","setProfileRun","arg == NULL");
  }
  if (d_profileRunSet) throw Exception("Parameters","setProfileRun","'-
prun' or '--profilerun' option specified multiple times.");
  if (strcmp(arg,"on") == 0) d_profileRun = true;
  else if (strcmp(arg,"off") == 0) d_profileRun = false;
  else throw Exception("Parameters","setProfileRun","Invalid profile
run parameter.  Must be 'on' or 'off'.");
  d_profileRunSet = true;
}

void Parameters::setInputFilename(char *arg) {
  if (arg == NULL) {
    throw Exception("Parameters","setInputFilename","arg == NULL");
  }
  if (d_inputFilenameSet) throw
Exception("Parameters","setInputFilename","'-i' or '--input' option
specified multiple times.");
  d_inputFilename = arg;
  d_inputFilenameSet = true;
}

void Parameters::setSeed(char *arg) {
  if (arg == NULL) {
    throw Exception("Parameters","setSeed","arg == NULL");
  }
  if (d_seedSet) throw Exception("Parameters","setSeed","'-s' or '--
seed' option specified multiple times.");
```

```
  try {
    d_seed = getUnsignedLong(arg);
  } catch (Exception e) {
    e.getClass();
    throw Exception("Parameters","setSeed","Invalid seed value.");
  }
  d_seedSet = true;
}

void Parameters::setReactionLimit(char *arg) {
  if (arg == NULL) {
    throw Exception("Parameters","setReactionLimit","arg == NULL");
  }
  if (d_reactionLimitSet)
    throw Exception("Parameters","setReactionLimit","'-r' or '--
reactionlimit' option specified multiple times.");
  try {
    d_reactionLimit = getUnsignedLong(arg);
  } catch (Exception e) {
    e.getClass();
    throw Exception("Parameters","setReactionLimit","Invalid reaction
limit value.");
  }
  d_reactionLimitSet = true;
}

void Parameters::setEndTime(char *arg) {
  if (arg == NULL) {
    throw Exception("Parameters","setEndTime","arg == NULL");
  }
  if (d_endTimeSet) throw Exception("Parameters","setEndTime","'-e' or
'--endtime' option specified multiple times.");
  try {
    d_endTime = getDouble(arg);
  } catch (Exception e) {
    e.getClass();
    throw Exception("Parameters","setEndTime","Invalid end time
value.");
  }
  if (d_endTime < 0) {
    throw Exception("Parameters","setEndTime","End time must be
positive.");
  }
  d_endTimeSet = true;
}

void Parameters::setPrintInterval(char *arg) {
  if (arg == NULL) {
    throw Exception("Parameters","setPrintInterval","arg == NULL");
  }
  if (d_printIntervalSet)
    throw Exception("Parameters","setPrintInterval","'-p' or '--
printinterval' option specified multiple times.");

  try {
```

```
    d_printInterval = getDouble(arg);
  } catch (Exception e) {
    e.getClass();
    throw Exception("Parameters","setPrintInterval","Invalid print
interval value.");
  }
  if (d_printInterval < 0)
    throw Exception("Parameters","setPrintInterval","Print interval
must be positive.");
  d_printIntervalSet = true;
}

double Parameters::getDouble(char *arg) {
  if (arg == NULL) {
    throw Exception("Parameters","getDouble","arg == NULL");
  }
  char *endPtr = NULL;
  double returnValue = strtod(arg,&endPtr);
  if (*endPtr != '\0') {
    throw Exception("Parameters","getDouble","Invalid double value.");
  }
  return returnValue;
}

unsigned long Parameters::getUnsignedLong(char *arg) {
  if (arg == NULL) {
    throw Exception("Parameters","getUnsignedLong","arg == NULL");
  }
  char *endPtr = NULL;
  unsigned long returnValue = strtoul(arg,&endPtr,10);
  if (*endPtr != '\0') {
    throw Exception("Parameters","getUnsignedLong","Invalid unsigned
long value.");
  }
  return returnValue;
}

double Parameters::getEndTime() {
  if (!d_endTimeSet) {
    throw Exception("Parameters","getEndTime","End time not set");
  }
  return d_endTime;
}

double Parameters::getPrintInterval() {
  if (!d_printIntervalSet) {
    throw Exception("Parameters","getPrintInterval","Print interval not
set");
  }
  return d_printInterval;
}

unsigned long Parameters::getReactionLimit() {
  if (!d_reactionLimitSet) {
```

```
    throw Exception("Parameters","getReactionLimit","Reaction limit not
set");
  }
  return d_reactionLimit;
}

unsigned long Parameters::getSeed() {
  return d_seed;
}
```

Parameters.h

```
#ifndef PARAMETERS_H
#define PARAMETERS_H

class Parameters {
 public:
  Parameters(int argc, char **argv);

  double getEndTime();
  double getPrintInterval();
  unsigned long getReactionLimit();
  unsigned long getSeed();
  bool isOutputOn() { return d_output; };
  bool isRunProfileOn() { return d_profileRun; };
  bool isReactionProfileOn() { return d_profileReactions; };
  char *getInputFilename() { return d_inputFilename; };

  bool isEndTimeSet() { return d_endTimeSet; };
  bool isPrintIntervalSet() { return d_printIntervalSet; };
  bool isReactionLimitSet() { return d_reactionLimitSet; };
  bool isHelpSet() { return d_helpSet; };

 private:
  char* d_inputFilename;
  bool d_inputFilenameSet;
  double d_endTime;
  bool d_endTimeSet;
  double d_printInterval;
  bool d_printIntervalSet;
  unsigned long d_reactionLimit;
  bool d_reactionLimitSet;
  unsigned long d_seed;
  bool d_seedSet;
  bool d_output;
  bool d_outputSet;
  bool d_helpSet;

  bool d_profileReactions;
  bool d_profileReactionsSet;
  bool d_profileRun;
  bool d_profileRunSet;

 private:
```

```
  void setInputFilename(char *arg);
  void setEndTime(char *arg);
  void setPrintInterval(char *arg);
  void setReactionLimit(char *arg);
  void setSeed(char *arg);
  void setOutput(char *arg);
  void setProfileReactions(char *arg);
  void setProfileRun(char *arg);
  double getDouble(char *arg);
  unsigned long getUnsignedLong(char *arg);
};

#endif
```

RandomBuffer.cpp

```
#include "RandomBuffer.h"
#include "../common/Exception.h"
#include <iostream>
#include <stdlib.h>

using namespace std;

RandomBuffer::RandomBuffer(Random *random, unsigned int bufferSize, int
bufferType) {
#ifdef DEBUG
  if (random == NULL) {
    throw Exception("RandomBuffer","RandomBuffer","random == NULL");
  }
  if (bufferSize == 0) {
    throw Exception("RandomBuffer","RandomBuffer","bufferSize == 0");
  }
  if ((bufferType != RANDOM_BUFFER_EXPONENTIAL) && (bufferType !=
RANDOM_BUFFER_UNIFORM)) {
    throw Exception("RandomBuffer","RandomBuffer","invalid
bufferType");
  }
#endif

  d_random = random;
  d_bufferSize = bufferSize;
  d_bufferType = bufferType;

  d_buffer = new double[d_bufferSize];
  if (d_buffer == NULL) {
    cerr << "Out of memory." << endl;
    exit(-1);
  }

  d_currentIndex = d_bufferSize;
}

RandomBuffer::~RandomBuffer() {
  if (d_buffer != NULL) delete [] d_buffer;
```

```
}

double RandomBuffer::getNumber() {
  d_currentIndex++;
  if (d_currentIndex >= d_bufferSize) {
    if (d_bufferType == RANDOM_BUFFER_EXPONENTIAL) {
      for(int i=0; i<d_bufferSize; i++) {
                d_buffer[i] = d_random->getExponential();
      }
    } else {
      for(int i=0; i<d_bufferSize; i++) {
                  d_buffer[i] = d_random->getUniform();
      }
    }
    d_currentIndex = 0;
  }
  return d_buffer[d_currentIndex];
}

void RandomBuffer::reuse() {
  if (d_currentIndex != 0) {
    d_currentIndex--;
  }
}
```

RandomBuffer.h

```
#ifndef RANDOMBUFFER_H
#define RANDOMBUFFER_H

#include "Random.h"

#define RANDOM_BUFFER_EXPONENTIAL 1
#define RANDOM_BUFFER_UNIFORM 2

class RandomBuffer {
 private:
  Random *d_random;
  double *d_buffer;
  unsigned int d_bufferSize;
  unsigned int d_currentIndex;
  int d_bufferType;

 public:
  RandomBuffer(Random *random, unsigned int bufferSize, int
bufferType);
  ~RandomBuffer();
  double getNumber();
  void reuse();
};

#endif
```

Random.cpp

```
#include <math.h>
#include "Random.h"

Random::Random(unsigned long seed) {
  d_myRandomNumber = (unsigned long)seed;
}

double Random::getUniform() {
  d_myRandomNumber = d_myRandomNumber * 0x41c64e6d + 0x3039;
  long temp = d_myRandomNumber & 0x7fffffff;
  return (double)temp/2147483647.0;
}

double Random::getExponential() {
  double temp = getUniform();
  while (temp == 0.0) {
    temp = getUniform();
  }
  return -log(temp);
}
```

Random.h

```
#ifndef RANDOM_H
#define RANDOM_H

class Random {
 private:
  unsigned long d_myRandomNumber;

 public:
  Random(unsigned long seed);
  double getUniform();
  double getExponential();
};

#endif
```

ReactionSort.cpp

```
#include "../common/Model.h"
#include <fstream>
#include <iostream>

using namespace std;

int main(int argc, char **argv) {
  if (argc != 3) {
    cerr << "usage: rxnsort <model> <rxncount>" << endl;
    return -1;
  }
```

```
  Model model(argv[1]);

  Vector <unsigned long> reactionCounts;
  Vector <unsigned long> reactionList;

  reactionCounts.resize(model.getReactions()->size());
  reactionList.resize(model.getReactions()->size());

  ifstream in(argv[2]);
  unsigned long value;
  for(unsigned long i=0; i<model.getReactions()->size(); i++) {
    in >> value;
    reactionCounts.set(i,value);
    reactionList.set(i,i);
  }

  for(unsigned long i=0; i<model.getReactions()->size(); i++) {
    for(unsigned long j=0; j<model.getReactions()->size(); j++) {
      if (reactionCounts.get(i) > reactionCounts.get(j)) {
               unsigned long temp = reactionCounts.get(i);
              reactionCounts.set(i,reactionCounts.get(j));
                      reactionCounts.set(j,temp);
               unsigned long temp2 = reactionList.get(i);
                reactionList.set(i,reactionList.get(j));
                       reactionList.set(j,temp2);
      }
    }
  }

  cout << model.getSpecies()->size() << endl;
  for(unsigned long i=0; i<model.getSpecies()->size(); i++) {
    cout << " " << model.getSpecies()->get(i);
  }
  cout << endl << endl;


  cout << model.getReactions()->size() << endl;
  for(unsigned long i=0; i<model.getReactions()->size(); i++) {
    Reaction *reaction = model.getReactions()-
>getPtr(reactionList.get(i));
    cout << reaction->getReactants()->size() << " ";
    for(unsigned long j=0; j<reaction->getReactants()->size(); j++) {
      cout << reaction->getReactants()->getPtr(j)->getCoefficient() <<
" ";
      cout << reaction->getReactants()->getPtr(j)->getSpeciesIndex() <<
" ";
    }
    cout << reaction->getProducts()->size() << " ";
    for(unsigned long j=0; j<reaction->getProducts()->size(); j++) {
      cout << reaction->getProducts()->getPtr(j)->getCoefficient() << "
";
      cout << reaction->getProducts()->getPtr(j)->getSpeciesIndex() <<
" ";
    }
```

```
    cout << reaction->getRateConstant() << endl;
  }
  cout << endl;

  cout << model.getOutputs()->size() << endl;
  for(unsigned long i=0; i<model.getOutputs()->size(); i++) {
    cout << " " << model.getOutputs()->get(i);
  }
  cout << endl;
}
```

SortingDirectMethod.cpp

```
#include "SortingDirectMethod.h"
#include "../common/Exception.h"
#include <stdlib.h>
#include <iostream>

using namespace std;

SortingDirectMethod::SortingDirectMethod() {
  uniformRandomBuffer = NULL;
   exponentialRandomBuffer = NULL;
}

SortingDirectMethod::~SortingDirectMethod() {
  if (uniformRandomBuffer != NULL)
    delete uniformRandomBuffer;
   if (exponentialRandomBuffer != NULL)
     delete exponentialRandomBuffer;
}

void SortingDirectMethod::init(Model *model, Random *random, double
*currentTime, Vector<long>* species) {
#ifdef DEBUG
  if (model == NULL) {
    throw Exception("SortingDirectMethod","init","model == NULL");
  }
  if (random == NULL) {
    throw Exception("SortingDirectMethod","init","random == NULL");
  }
  if (currentTime == NULL) {
    throw Exception("SortingDirectMethod","init","currentTime ==
NULL");
  }
#endif

  this->model = model;
  this->currentTime = currentTime;
  this->species = species;
  totalPropensity = 0.0;

  uniformRandomBuffer = new
RandomBuffer(random,100000,RANDOM_BUFFER_UNIFORM);
```

```
  if (uniformRandomBuffer == NULL) {
    throw Exception("SortingDirectMethod","init","Out of memory.");
  }

   exponentialRandomBuffer = new
RandomBuffer(random,100000,RANDOM_BUFFER_EXPONENTIAL);
   if (exponentialRandomBuffer == NULL) {
     throw Exception("SortingDirectMethod","init","Out of memory.");
   }

  propensities.resize(model->getReactions()->size());

  reactionList.resize(model->getReactions()->size());
  for(unsigned long i=0; i<model->getReactions()->size(); i++) {
    reactionList.set(i,i);
  }

  reactionCounts.resize(model->getReactions()->size());
  for(int i=0; i<reactionCounts.size(); i++) {
    reactionCounts.set(i,0);
  }
  reinit();
}

void SortingDirectMethod::reinit() {
  totalPropensity = 0.0;
  for(unsigned long i=0; i<model->getReactions()->size(); i++) {
    propensities.set(i,model->getReactions()->getPtr(i)-
>getPropensity(species));
    totalPropensity += propensities.get(i);
  }
}

void SortingDirectMethod::step() {
  double scaledRand = uniformRandomBuffer->getNumber() *
totalPropensity;
  reactionIndex = 0;
  unsigned long i;
  for(i=0; i<model->getReactions()->size(); i++) {
    reactionIndex = reactionList.get(i);
    if (propensities.get(reactionIndex) != 0.0) {
      scaledRand -= propensities.get(reactionIndex);
      if (scaledRand <= 0.0) break;
    }
  }

  if (i != 0) {
    reactionList.swap(i,i-1);
  }

   double time = exponentialRandomBuffer->getNumber() /
totalPropensity;
   *currentTime += time;
}
```

```
void SortingDirectMethod::execute() {
  model->getReactions()->getPtr(reactionIndex)->execute(species);

reactionCounts.set(reactionIndex,reactionCounts.get(reactionIndex)+1);
  Vector<unsigned long>* dependencies = model-
>getReactionDependencies()->getPtr(reactionIndex);
  for(unsigned long i=0; i<dependencies->size(); i++) {
    unsigned long index = dependencies->get(i);
    totalPropensity -= propensities.get(index);
    propensities.set(index,model->getReactions()->getPtr(index)-
>getPropensity(species));
    totalPropensity += propensities.get(index);
  }
  if (totalPropensity < 0) totalPropensity = 0.0;
}

unsigned long SortingDirectMethod::getCurrentUpdateFactor() {
  return model->getReactionDependencies()->getPtr(reactionIndex)-
>size();
}
```

SortingDirectMethod.h

```
#ifndef SORTINGDIRECTMETHOD_H
#define SORTINGDIRECTMETHOD_H

#include "../common/Model.h"
#include "RandomBuffer.h"

class SortingDirectMethod {
 private:
  Model *model;
  double *currentTime;
  unsigned long reactionIndex;
  Vector<double> propensities;
  Vector<long>* species;
  double totalPropensity;
  RandomBuffer *uniformRandomBuffer;
  RandomBuffer *exponentialRandomBuffer;
  Vector<unsigned long> reactionList;
  Vector<unsigned long> reactionCounts;

 public:
  SortingDirectMethod();
  ~SortingDirectMethod();
  void init(Model *model, Random *random, double *currentTime,
                          Vector<long>* species);
  void reinit();
  void step();
  void execute();
  unsigned long getCurrentUpdateFactor();
  Vector<unsigned long>* getReactionCounts() { return &reactionCounts;
};
};
```

```
#endif
```

Parallel Source Code

The following source code implements the Parallel Reaction Method. The implementation depends on a pthread library, the sprng parallel random number generator library, and the common code.

Event.cpp

```
#include "Event.h"

bool Event::cancels(Event event) {
  if ((event.getThreadId() == getThreadId()) &&
      (event.getIdentifier() == getIdentifier()) &&
      (event.isAntiMessage() != isAntiMessage())) {
    return true;
  } else {
    return false;
  }
}
```

Event.h

```
#ifndef EVENT_H
#define EVENT_H
#include <fstream>
#include <iostream>

using namespace std;

class Event {
 private:
  unsigned int d_threadId;
  double d_time;
  unsigned long d_reactionIndex;
  bool d_isAntiMessage;
  unsigned long d_identifier;

 public:
  Event() { };

  Event(unsigned int threadId, double time, unsigned long
reactionIndex,
                bool isAntiMessage, unsigned long id) {
    d_threadId = threadId;
    d_time = time;
    d_reactionIndex = reactionIndex;
    d_isAntiMessage = isAntiMessage;
    d_identifier = id;
  }

  Event(const Event &e) {
    d_threadId = e.d_threadId;
```

```
    d_time = e.d_time;
    d_reactionIndex = e.d_reactionIndex;
    d_isAntiMessage = e.d_isAntiMessage;
    d_identifier = e.d_identifier;
  }

  void setThreadId(unsigned int threadId) { d_threadId = threadId; }
  void setTime(double time) { d_time = time; }
  void setReactionIndex(unsigned long reactionIndex) { d_reactionIndex
= reactionIndex; }
  void setAntiMessage(bool value) { d_isAntiMessage = value; }
  void setIdentifier(unsigned long id) { d_identifier = id; }

  unsigned int getThreadId() { return d_threadId; }
  double getTime() { return d_time; }
  unsigned long getReactionIndex() { return d_reactionIndex; }
  bool isAntiMessage() { return d_isAntiMessage; }
  unsigned long getIdentifier() { return d_identifier; }

  bool cancels(Event e);

  void print() {
    cout << d_threadId << " " << d_time << " " << d_reactionIndex <<
endl;
  }
};

#endif
```

EventNode.h

```
#ifndef EVENTNODE_H
#define EVENTNODE_H

#include "Event.h"

class EventNode {
 public:
  Event event;
  EventNode *next;
};

#endif
```

EventStack.cpp

```
#include "EventStack.h"
#include "../common/Exception.h"
#include <iostream>

using namespace std;

EventStack::EventStack() {
```

```
  top = NULL;
  itemCount = 0;
  deleteCount = 0;

  eventPool = new EventNode();
  eventPool->next = NULL;
}

EventStack::~EventStack() {
  while (!isEmpty()) {
    pop();
  }
}

void EventStack::push(Event e) {
  EventNode* node = eventPool;
  eventPool = eventPool->next;
  if (eventPool == NULL) {
    eventPool = new EventNode();
    eventPool->next = NULL;
  }

  if (node == NULL) {
    throw Exception("EventStack","push","out of memory.");
  }
  node->event = e;
  node->next = top;
  top = node;
}

void EventStack::trim(double time, unsigned long threadId) {
  EventNode* node = top;
  if (node == NULL) return;
  while (node->event.getTime() > time) {
    node = node->next; if (node == NULL) return;
  }
  while (node->event.getThreadId() != threadId) {
    node = node->next; if (node == NULL) return;
  }
  node = node->next; if (node == NULL) return;
  while (node->event.getThreadId() != threadId) {
    node = node->next; if (node == NULL) return;
  }

  EventNode* temp = node->next;
  node->next = NULL;
  node = temp;
  while (node != NULL) {
    temp = node->next;


    deleteCount++;
    itemCount--;
    node->next = eventPool;
    eventPool = node;
```

```
    node = temp;
  }
}

Event EventStack::pop() {
  deleteCount++;
  itemCount--;
#ifdef DEBUG
  if (isEmpty()) {
    throw Exception("EventStack","pop","popped an empty stack.");
  }
#endif
  EventNode* node = top;
  Event e = top->event;
  top = top->next;
  deleteCount++;

  node->next = eventPool;
  eventPool = node;

  return e;
}

Event EventStack::peek() {
#ifdef DEBUG
  if (isEmpty()) {
    throw Exception("EventStack","pop","peeked an empty stack.");
  }
#endif
  return top->event;
}

bool EventStack::isEmpty() {
  return (top == NULL);
}

void EventStack::print() {
  EventNode *n = top;
  while (n != NULL) {
    cout << n->event.getThreadId() << "-"
                  << n->event.getIdentifier() << " ";
    n = n->next;
  }
  cout << endl;
}
```

EventStack.h

```
#ifndef EVENTSTACK_H
#define EVENTSTACK_H

#include "Event.h"
#include "EventNode.h"
```

```
#include <pthread.h>

class EventStack {
 private:
  EventNode *top;
  int itemCount;
  int deleteCount;
  int mallocCount;

  EventNode *eventPool;

 public:
  EventStack();
  ~EventStack();
  void push(Event e);
  Event pop();
  Event peek();
  bool isEmpty();
  void print();
  void trim(double time, unsigned long threadId);
};

#endif
```

makefile

```
OBJS = Event.o EventStack.o OutputData.o SimulationThread.o \
          SortedEventList.o ParallelRandom.o ThreadControl.o \
                                SumTree.o

CXX = c++
CPPFLAGS = -O3

all: pess

pess: $(OBJS) Simulator.o pess.cpp
 $(CXX) $(CPPFLAGS) -o pess pess.cpp Simulator.o $(OBJS) ../common/*.o -
lpthread

clean:
   -rm *.o
   -rm pess
   -rm *~
```

OutputData.cpp

```
#include "OutputData.h"
#include "../common/Exception.h"
#include <fstream>
#include <iostream>
#include <math.h>

OutputData::OutputData() {
```

```
}

void OutputData::init(Model *model, double printInterval, double
endTime) {
  unsigned long stepCount = 0;
  double temp = 0.0;
  while (temp < (endTime+printInterval*0.01)) {
    stepCount++;
    temp += printInterval;
  }

  d_stepTimes.resize(stepCount);
  d_outputPopulations.resize(stepCount);
  for(int i=0; i<stepCount; i++) {
    d_stepTimes.set(i,((double)i)*printInterval);
    d_outputPopulations.getPtr(i)->resize(model->getOutputs()->size());
  }

  d_speciesToOutputMap.resize(model->getSpecies()->size());
  for(unsigned long i=0; i<model->getOutputs()->size(); i++) {
    d_speciesToOutputMap.set(model->getOutputs()->get(i),i);
  }
}

void OutputData::set(unsigned long step, unsigned long speciesIndex,
unsigned long value) {
#ifdef DEBUG
  if (step >= d_stepTimes.size()) {
    throw Exception("OutputData","output","step out of range");
  }
#endif
  unsigned long item = d_speciesToOutputMap.get(speciesIndex);
  d_outputPopulations.getPtr(step)->set(item,value);
}

void OutputData::write(char *filename) {
  ofstream out(filename);
  for(unsigned long step=0; step<d_stepTimes.size(); step++) {
    out << d_stepTimes.get(step);
    Vector<unsigned long>* outputData =
d_outputPopulations.getPtr(step);
    for(unsigned long i=0; i<outputData->size(); i++) {
      out << " " << outputData->get(i);
    }
    out << endl;
  }
  out.close();
}
```

OutputData.h

```
#ifndef OUTPUTDATA_H
#define OUTPUTDATA_H
```

```
#include "../common/Model.h"
#include <fstream>
#include <iostream>

using namespace std;

class OutputData {
 public:
  OutputData();
  void init(Model *model, double printInterval, double endTime);
  void set(unsigned long step, unsigned long speciesIndex, unsigned
long value);
  void write(char *filename);
  int getStepCount() { return d_stepTimes.size(); };

 private:
  Vector<double> d_stepTimes;
  Vector< Vector<unsigned long> > d_outputPopulations;
  Vector<unsigned long> d_speciesToOutputMap;
};

#endif
```

ParallelRandom.cpp

```
#include <math.h>
#include <stdlib.h>
#include "ParallelRandom.h"
#include <iostream>

using namespace std;

ParallelRandom::ParallelRandom() {

}

void ParallelRandom::init(unsigned int threadId, unsigned int
threadCount,int seed) {
  d_currentIndex = 0;
  int random_seed = 100;
  for(unsigned long i=0; i<RANDOM_BUFFER_SIZE; i++) {
    // To use the SPRNG, re-add the calls to sprng here
    // Currently using a home-grown rng to resolve compilation
    // issues with certain platforms
    // The numbers for the dissertation were gathered using calls to
    // SPRNG
    random_seed = ((random_seed * 1103515245 + 12345) / 65546) % 32768;
    while (random_seed == 0) {
      random_seed = ((random_seed * 1103515245 + 12345) / 65546) %
32768;
    }
    d_numbers[i] = abs(random_seed / 32768.0);
  }
```

```
}

double ParallelRandom::getUniform() {
  return d_numbers[d_currentIndex];
}

double ParallelRandom::getExponential() {
  return -log(getUniform());
}

void ParallelRandom::next() {
  d_currentIndex++;
  if (d_currentIndex == RANDOM_BUFFER_SIZE) {
    d_currentIndex = 0;
    for(unsigned long i=0; i<RANDOM_BUFFER_SIZE; i++) {
      d_numbers[i] = (double)rand()/(double)RAND_MAX;
    }
  }
  while ((d_numbers[d_currentIndex] == 0.0) &&
(d_numbers[d_currentIndex] == 1.0)) {
    d_currentIndex++;
  }
}

void ParallelRandom::back() {
  if (d_currentIndex != 0) d_currentIndex--;
}
```

ParallelRandom.h

```
#ifndef RANDOM_H
#define RANDOM_H

#include "../../sprng/sprng2.0/include/sprng.h"

#define RANDOM_BUFFER_SIZE 1000000

class ParallelRandom {
 private:
  int *d_sprngPtr;
  unsigned long d_currentIndex;
  double d_numbers[RANDOM_BUFFER_SIZE];

 public:
  ParallelRandom();
  void init(unsigned int threadId, unsigned int threadCount, int seed);
  double getUniform();
  double getExponential();
  void next();
  void back();

  int getCurrentIndex() { return d_currentIndex; };
};
```

```
#endif
```

pess.cpp

```
#include <sys/time.h>
#include <unistd.h>
#include <pthread.h>
#include <fstream>
#include <iostream>
#include <stdlib.h>
#include "SharedVariables.h"
#include "../common/Infinity.h"

using namespace std;

typedef Vector<int> IntVector;

void parseCommandLineArguments(int argc, char **argv, SharedVariables
&shared) {
  if (argc != 8) {
    cerr << "usage: pess <modelfile> <threadCount> <printinterval>
<endtime> <seed> [<profile-file> [<assign-file>]]" << endl;
    exit(-1);
  }

  shared.modelFilename = argv[1];

  if (sscanf(argv[2], "%u", &shared.threadCount) != 1)
    throw Exception("", "main", "Invalid thread count.");

  if (sscanf(argv[3], "%lf", &shared.printInterval) != 1)
    throw Exception("", "main", "Invalid print interval.");

  if (sscanf(argv[4], "%lf", &shared.endTime) != 1)
    throw Exception("", "main", "Invalid end time.");

  if (sscanf(argv[5], "%u", &shared.seed) != 1)
    throw Exception("", "main", "Invalid seed.");

  shared.profileFilename = argv[6];
  shared.assignmentsFilename = argv[7];
}

void assignOutputsToThreads(SharedVariables &shared) {
  IntVector species(shared.model->getSpecies()->size());
  for (unsigned long i = 0; i < shared.model->getSpecies()->size();
i++) {
    species.set(i,-1);
  }

  for (unsigned long i = 0; i < shared.model->getReactions()->size();
i++) {
    Reaction *reaction = shared.model->getReactions()->getPtr(i);
```

```
    for (unsigned int j = 0; j < reaction->getReactants()->size(); j++)
{
      unsigned long reactantIndex = reaction->getReactants()-
>getPtr(j)->getSpeciesIndex();

species.set(reactantIndex,shared.threadAssignedToReaction.get(i));
    }
  }

  for (unsigned long i = 0; i < shared.model->getReactions()->size();
i++) {
    Reaction *reaction = shared.model->getReactions()->getPtr(i);
    for (unsigned int j = 0; j < reaction->getProducts()->size(); j++)
{
      unsigned long productIndex = reaction->getProducts()->getPtr(j)-
>getSpeciesIndex();
      if (species.get(productIndex) == -1) {
    species.set(productIndex,shared.threadAssignedToReaction.get(i));
      }
    }
  }

  for (unsigned long i = 0; i < shared.model->getSpecies()->size();
i++) {
    if (species.get(i) == -1) species.set(i,0);
  }

  for (unsigned long i = 0; i < shared.model->getSpecies()->size();
i++) {
    if (shared.model->getOutputs()->contains(i)) {
      shared.simulationThreads.getPtr(species.get(i))-
>getOutputAssignments()->add(i);
    }
  }
}

void assignThreadsToReactions(SharedVariables &shared) {
  shared.threadAssignedToReaction.resize(shared.model->getReactions()-
>size());
  int thread;
  ifstream in(shared.assignmentsFilename);
  for(int i=0; i<shared.model->getReactions()->size(); i++) {
    in >> thread;
    shared.threadAssignedToReaction.set(i,thread);
  }

  for(unsigned long reactionIndex = 0; reactionIndex<shared.model-
>getReactions()->size(); reactionIndex++) {
    int threadIndex =
shared.threadAssignedToReaction.get(reactionIndex);
    shared.simulationThreads.getPtr(threadIndex)-
>getReactionAssignments()->add(reactionIndex);
  }
  in.close();
}
```

```
void *run(void *thread)
{
  SimulationThread *t = (SimulationThread *) thread;
  t->run();
  pthread_exit(NULL);
}

void startSimulationThreads(SharedVariables &shared, Vector<pthread_t>
&pthreads) {
  for (unsigned int threadIndex=0; threadIndex < shared.threadCount;
threadIndex++) {
    SimulationThread *currentThread =
shared.simulationThreads.getPtr(threadIndex);
    currentThread->init(threadIndex,&shared);
    int rc = pthread_create(pthreads.getPtr(threadIndex), NULL, run,
currentThread);
    if (rc) throw Exception("", "main", "Error starting thread.");
  }
}

double getExecutionTime(timeval startTime) {
  timeval endTime;
  gettimeofday(&endTime,NULL);
  double totalTime = endTime.tv_sec - startTime.tv_sec;
  totalTime += ((double)endTime.tv_usec)/1000000.0;
  totalTime -= ((double)startTime.tv_usec)/1000000.0;
  return totalTime;
}

double getSimulationTime(SharedVariables &shared) {
  double minTime = MYINFINITY;
  for(int i=0; i<shared.threadCount; i++) {
    double time = shared.simulationThreads.getPtr(i)->getCurrentTime();
    if (time < minTime) minTime = time;
  }
  return minTime;
}

unsigned long getReactionsExecutedCount(SharedVariables &shared) {
  unsigned long count = 0;
  for(int i=0; i<shared.threadCount; i++) {
    count += shared.simulationThreads.getPtr(i)-
>getReactionsExecutedCount();
  }
  return count;
}

unsigned long getReactionsUnexecutedCount(SharedVariables &shared) {
  unsigned long count = 0;
  for(int i=0; i<shared.threadCount; i++) {
    count += shared.simulationThreads.getPtr(i)-
>getReactionsUnexecutedCount();
  }
  return count;
```

```
}

unsigned long getCommunicationCount(SharedVariables &shared) {
  unsigned long count = 0;
  for(int i=0; i<shared.threadCount; i++) {
    count += shared.simulationThreads.getPtr(i)-
>getCommunicationCount();
  }
  return count;
}

void printStatistics(timeval startTime, double simulationTime, unsigned
long reactionsExecutedCount, unsigned long reactionsUnexecutedCount,
unsigned long communicationCount) {
  double executionTime = getExecutionTime(startTime);

  cout << endl;
  cout << "Simulator Time = " << simulationTime << endl;
  cout << "Execution Time = " << executionTime << endl;
  cout << "Reactions Executed = " << reactionsExecutedCount
       << " (" << (reactionsExecutedCount/executionTime) << "
rxns/sec)" << endl;
  cout << "Reactions Unexecuted = " << reactionsUnexecutedCount
       << " (" << (reactionsUnexecutedCount/executionTime) << "
rxns/sec)" << endl;
  cout << "Communications Executed = " << communicationCount
  << " (" << (communicationCount/executionTime) << " comms/sec)" <<
endl;
}

void monitorProgress(SharedVariables &shared, timeval startTime) {
  while(shared.globalMinimumTime < shared.endTime) {
    sleep(5);

    cout << "halt start time = " << getExecutionTime(startTime) <<
endl;

    for(int i=0; i<shared.threadCount; i++) {
      ThreadControl *tc = shared.simulationThreads.getPtr(i)-
>getThreadControl();
      tc->halt();
    }

    for(int i=0; i<shared.threadCount; i++) {
      ThreadControl *tc = shared.simulationThreads.getPtr(i)-
>getThreadControl();
      tc->waitForHalt();
    }

    shared.globalMinimumTime = getSimulationTime(shared);
    cout << "halt complete time = " << getExecutionTime(startTime) <<
endl;

    unsigned long executedReactionsCount =
getReactionsExecutedCount(shared);
```

```
    unsigned long unexecutedReactionsCount =
getReactionsUnexecutedCount(shared);
    unsigned long communicationCount = getCommunicationCount(shared);

    for(int i=0; i<shared.threadCount; i++) {
      ThreadControl *tc = shared.simulationThreads.getPtr(i)-
>getThreadControl();
      tc->resume();
    }

    for(int i=0; i<shared.threadCount; i++) {
      ThreadControl *tc = shared.simulationThreads.getPtr(i)-
>getThreadControl();
      tc->waitForResume();
    }

    printStatistics(startTime, shared.globalMinimumTime,
executedReactionsCount,  unexecutedReactionsCount, communicationCount);
    if (getExecutionTime(startTime) > 120) {
      break;
    }
  }
}

void stopSimulationThreads(SharedVariables &shared, Vector<pthread_t>
pthreads) {
    int status;
    for(int i=0; i<shared.threadCount; i++) {
      ThreadControl *tc = shared.simulationThreads.getPtr(i)-
>getThreadControl();
      tc->exit();
//       pthread_join(pthreads.get(i), (void **) &status);
    }
}

int main(int argc, char **argv) {
  timeval startTime;
  gettimeofday(&startTime,NULL);

  try {
    SharedVariables shared;
    parseCommandLineArguments(argc,argv,shared);
    shared.globalMinimumTime = 0.0;
    shared.model = new Model(shared.modelFilename);
    shared.simulationThreads.resize(shared.threadCount);
    assignThreadsToReactions(shared);
    assignOutputsToThreads(shared);

shared.outputData.init(shared.model,shared.printInterval,shared.endTime
);
    Vector<pthread_t> pthreads(shared.threadCount);
    startSimulationThreads(shared,pthreads);
    monitorProgress(shared,startTime);
    stopSimulationThreads(shared,pthreads);
    shared.outputData.write("output.txt");
```

```
    double executionTime = getExecutionTime(startTime);
    shared.globalMinimumTime = getSimulationTime(shared);
    unsigned long executedReactionsCount =
getReactionsExecutedCount(shared);
    unsigned long unexecutedReactionsCount =
getReactionsUnexecutedCount(shared);
    unsigned long communicationCount = getCommunicationCount(shared);
    printStatistics(startTime, shared.globalMinimumTime,
executedReactionsCount, unexecutedReactionsCount, communicationCount);
    cout << "Model Filename: " << shared.modelFilename << endl;
    cout << "Thread Count: " << shared.threadCount << endl;

    ofstream out;
    out.open(shared.profileFilename);
    out << shared.modelFilename << " ";
    out << shared.endTime << " ";
    out << shared.threadCount << " ";
    out << executionTime << " ";
    out << executedReactionsCount / executionTime << " ";
    out << executedReactionsCount << " ";
    out << unexecutedReactionsCount << " ";
    out << communicationCount <<  endl;
    out.close();

  } catch(Exception e) {
    cout << e.getClass() << "::" << e.getFunction() << "() - " << e.
      getMessage() << endl;
    return -1;
  }
}
```

SharedVariables.h

```
#ifndef SHAREDVARIABLES_H
#define SHAREDVARIABLES_H

#include "../common/Vector.h"
#include "../common/Model.h"
#include "SimulationThread.h"
#include "OutputData.h"

class SimulationThread;
class Simulation;

class SharedVariables {
 public:
  char *assignmentsFilename;
  char *modelFilename;
  char *profileFilename;
  int threadCount;
  double printInterval;
  double endTime;
  Model *model;
```

```
  Vector<SimulationThread> simulationThreads;
  Vector<int> threadAssignedToReaction;
  OutputData outputData;
  int seed;
  double globalMinimumTime;
};

#endif
```

SimulationThread.cpp

```
#include <iostream>
#include "SimulationThread.h"

using namespace std;

void SimulationThread::init(int threadIndex, SharedVariables* shared) {
  d_threadIndex = threadIndex;
  d_shared = shared;
}

double SimulationThread::getCurrentTime() {
  double currentTime = d_simulator.getCurrentTime();
  if (!d_localIncomingEvents.isEmpty()) {
    if (d_localIncomingEvents.peekEarliestEvent().getTime() <
currentTime) {
      currentTime =
d_localIncomingEvents.peekEarliestEvent().getTime();
    }
  }
  d_globalIncomingEvents.lock();
  if (!d_globalIncomingEvents.isEmpty()) {
    if (d_globalIncomingEvents.peekEarliestEvent().getTime() <
currentTime) {
      currentTime =
d_globalIncomingEvents.peekEarliestEvent().getTime();
    }
  }
  d_globalIncomingEvents.unlock();
  return currentTime;
}

void SimulationThread::output(double time) {

  for(int step =
   (int)(d_simulator.getCurrentTime() / d_shared->printInterval) + 1;
      ((step < d_shared->outputData.getStepCount()) &&
       (step < (int)(time / d_shared->printInterval) + 1));
      step++) {
    for(int i=0; i<d_outputAssignments.size(); i++) {
      d_shared->outputData.set(step,d_outputAssignments.get(i),
    d_simulator.getCurrentSpeciesValue(d_outputAssignments.get(i)));
    }
  }
```

```
}

void SimulationThread::notifyAffectedThreads(Event event) {
  if (event.getTime() < d_shared->endTime) {
    UnsignedLongVector *dependencies = d_shared->model-
>getReactionDependencies()->getPtr(event.getReactionIndex());
    Vector<unsigned int> notifiedThreads;
    for (unsigned long i=0; i < dependencies->size(); i++) {
      unsigned int remoteThreadId = d_shared-
>threadAssignedToReaction.get(dependencies->get(i));
      if ((remoteThreadId != d_threadIndex) &&
(!notifiedThreads.contains(remoteThreadId))) {
                          d_communicationCount++;
                  SortedEventList *eventList = d_shared-
>simulationThreads.getPtr(remoteThreadId)->getIncomingEventList();
                            eventList->lock();
                       eventList->addEvent(event);
                          eventList->unlock();
                   notifiedThreads.add(remoteThreadId);
      }
    }
  }
}

void SimulationThread::executeLocalEvent() {
  if (d_simulator.getCurrentTime() < d_shared->endTime) {
    Event localEvent = d_simulator.getNextEvent();

#ifdef DEBUG
    cout << d_threadIndex << " executing local event time="
   << localEvent.getTime() << " id=" << localEvent.getIdentifier() << "
rxn="
                << localEvent.getReactionIndex() << endl;
#endif

    d_eventStack.push(localEvent);
    notifyAffectedThreads(localEvent);
    output(localEvent.getTime());
    d_simulator.executeLocalEvent();
  }
}

void SimulationThread::rollback(double time) {
  while (true) {
    if (d_eventStack.isEmpty()) {
      d_simulator.rollbackComplete(0.0);
      break;
    }
    Event event = d_eventStack.pop();
    if (event.getThreadId() == d_threadIndex) {
      if (event.getTime() < time) {
                        d_eventStack.push(event);
              d_simulator.rollbackComplete(event.getTime());
                                  break;
      } else {
```

```
                    d_simulator.unexecuteEvent(event);
                        event.setAntiMessage(true);
                       notifyAffectedThreads(event);
      }
    } else {
      d_simulator.unexecuteEvent(event);
      d_localIncomingEvents.addEvent(event);
    }
  }

#ifdef DEBUG
  cout << d_threadIndex << " rolled back to time " <<
d_simulator.getCurrentTime() << endl;
#endif
}

void SimulationThread::run() {
  d_simulator.init(d_threadIndex,d_shared->threadCount,   d_shared-
>model, &d_reactionAssignments,d_shared->seed);
  for(int i=0; i<d_outputAssignments.size(); i++)
    d_shared->outputData.set(0,d_outputAssignments.get(i),
d_simulator.getCurrentSpeciesValue(d_outputAssignments.get(i)));

  while (true) {
    if (d_threadControl.haltCheck()) {
      d_eventStack.trim(d_shared->globalMinimumTime, d_threadIndex);
    }
    d_threadControl.exitCheck();

#ifdef DEBUG
//     if (d_simulator.getCurrentTime() < d_shared->endTime) {
//       cout << d_threadIndex << " EventStack: ";
//       d_eventStack.print();
//     }
#endif

    d_globalIncomingEvents.lock();
    while(!d_globalIncomingEvents.isEmpty()) {

d_localIncomingEvents.addEvent(d_globalIncomingEvents.getEarliestEvent(
));
    }
    d_globalIncomingEvents.unlock();

    Event localEvent = d_simulator.getNextEvent();
    if (d_localIncomingEvents.isEmpty()) {
      executeLocalEvent();
    } else {
      Event remoteEvent = d_localIncomingEvents.peekEarliestEvent();
      if (localEvent.getTime() < remoteEvent.getTime()) {
                          executeLocalEvent();
      } else if (remoteEvent.getTime() < d_simulator.getCurrentTime())
{
                     rollback(remoteEvent.getTime());
      } else {
```

```
          while(localEvent.getTime() > remoteEvent.getTime()) {
#ifdef DEBUG
                cout << d_threadIndex << " local event time="
         << localEvent.getTime() << " id=" << localEvent.getIdentifier()
<< " rxn="
                    << localEvent.getReactionIndex() << endl;
#endif
          remoteEvent = d_localIncomingEvents.getEarliestEvent();
                       d_eventStack.push(remoteEvent);
                        output(remoteEvent.getTime());
                d_simulator.executeRemoteEvent(remoteEvent);

#ifdef DEBUG
          cout << d_threadIndex << " executing remote event time="
                       << remoteEvent.getTime() << " id=" <<
remoteEvent.getIdentifier() << " rxn="
                    << remoteEvent.getReactionIndex() << endl;
#endif

                  localEvent = d_simulator.getNextEvent();
                 if (d_localIncomingEvents.isEmpty()) break;
          remoteEvent = d_localIncomingEvents.peekEarliestEvent();

                                      }
                             executeLocalEvent();
       }
     }
   }
}
```

SimulationThread.h

```
#ifndef SIMULATIONTHREAD_H
#define SIMULATIONTHREAD_H

#include "SharedVariables.h"
#include "ThreadControl.h"
#include "../common/Vector.h"
#include "Simulator.h"
#include "SortedEventList.h"
#include "EventStack.h"

typedef Vector<unsigned long> UnsignedLongVector;

class SharedVariables;

class SimulationThread {
 public:
  void init(int threadIndex, SharedVariables* shared);
  void run();

  ThreadControl* getThreadControl() { return &d_threadControl; };
  UnsignedLongVector* getReactionAssignments() { return
&d_reactionAssignments; };
```

```
  double getCurrentTime();
  unsigned long getReactionsExecutedCount() { return
d_simulator.getReactionsExecutedCount(); };
  unsigned long getReactionsUnexecutedCount() { return
d_simulator.getReactionsUnexecutedCount(); };
  unsigned long getCommunicationCount() { return d_communicationCount;
};
  SortedEventList* getIncomingEventList() { return
&d_globalIncomingEvents; };
  UnsignedLongVector *getOutputAssignments() { return
&d_outputAssignments; };

 private:
  void executeLocalEvent();
  void rollback(double time);
  void output(double time);
  void notifyAffectedThreads(Event e);
  int d_threadIndex;
  SharedVariables* d_shared;
  UnsignedLongVector d_reactionAssignments;
  ThreadControl d_threadControl;
  Simulator d_simulator;
  EventStack d_eventStack;
  unsigned long d_communicationCount;
  SortedEventList d_globalIncomingEvents;
  SortedEventList d_localIncomingEvents;
  UnsignedLongVector d_outputAssignments;
};

#endif
```

Simulator.cpp

```
#include "Simulator.h"
#include "../common/Infinity.h"
#include "SharedVariables.h"

Simulator::Simulator() {

}

void Simulator::init(int threadIndex, int threadCount,     Model
*model, Vector<unsigned long> *assignedReactions, int seed) {
  d_sumTree = new SumTree(assignedReactions->size());
  d_threadIndex = threadIndex;
  d_model = model;
  d_assignedReactions = assignedReactions;
  d_currentTime = 0.0;
  d_randomNumber.init(threadIndex,threadCount,seed);
  d_event.setThreadId(threadIndex);
  d_event.setAntiMessage(false);
  d_event.setIdentifier(0);
  d_reactionsExecutedCount = 0;
  d_reactionsUnexecutedCount = 0;
```

```
  d_species.resize(model->getSpecies()->size());
  for(unsigned long i=0; i<model->getSpecies()->size(); i++) {
    d_species.set(i,model->getSpecies()->get(i));
  }

  d_localDependencies.resize(model->getReactions()->size());
  for(unsigned long i=0; i<model->getReactions()->size(); i++) {
    Vector<unsigned long> *dependencies = model-
>getReactionDependencies()->getPtr(i);
    for(unsigned long j=0; j<dependencies->size(); j++) {
      unsigned long reactionIndex = dependencies->get(j);
      if (d_assignedReactions->contains(reactionIndex)) {
        d_localDependencies.getPtr(i)->add(d_assignedReactions-
>find(reactionIndex));
      }
    }
  }

  calculatePropensities();
  generateNewEvent();
}

void Simulator::calculatePropensities() {
  for(int i=0; i<d_assignedReactions->size(); i++) {
    unsigned long reactionIndex = d_assignedReactions->get(i);
    Reaction *reaction = d_model->getReactions()-
>getPtr(reactionIndex);
    d_sumTree->set(i,reaction->getPropensity(&d_species));
  }
}

unsigned long Simulator::selectReaction(double uniformRand) {
  return d_assignedReactions->get(d_sumTree-
>selectReaction(uniformRand));
}

void Simulator::generateNewEvent() {
  d_randomNumber.next();
  d_event.setTime(d_currentTime + d_randomNumber.getExponential() /
d_sumTree->getTotal());
  d_randomNumber.next();

d_event.setReactionIndex(selectReaction(d_randomNumber.getUniform()));
  d_event.setIdentifier(d_event.getIdentifier() + 1);
}

void Simulator::executeLocalEvent() {
//   event.print();
  d_currentTime = d_event.getTime();
  d_model->getReactions()->getPtr(d_event.getReactionIndex())-
>execute(&d_species);
  updatePropensities(d_event.getReactionIndex());
  generateNewEvent();
  d_reactionsExecutedCount++;
```

```
}

void Simulator::executeRemoteEvent(Event event) {
//    event.print();
  double oldTotalPropensity = d_sumTree->getTotal();

  d_model->getReactions()->getPtr(event.getReactionIndex())-
>execute(&d_species);
  d_currentTime = event.getTime();
  updatePropensities(event.getReactionIndex());
  if (d_sumTree->getTotal() == 0.0) {
    d_event.setTime(MYINFINITY);
  } else if (oldTotalPropensity == 0.0) {
    d_randomNumber.back();
    d_event.setTime(d_currentTime + d_randomNumber.getExponential() /
d_sumTree->getTotal());
    d_randomNumber.next();
  } else {
    d_event.setTime(oldTotalPropensity/d_sumTree-
>getTotal()*(d_event.getTime()-d_currentTime)+d_currentTime);
  }

d_event.setReactionIndex(selectReaction(d_randomNumber.getUniform()));
}

void Simulator::unexecuteEvent(Event e) {
  if (e.getThreadId() == d_threadIndex) {
    d_randomNumber.back();
    d_randomNumber.back();
    d_reactionsExecutedCount--;
    d_reactionsUnexecutedCount++;
  }
  d_model->getReactions()->getPtr(e.getReactionIndex())-
>unexecute(&d_species);
  updatePropensities(e.getReactionIndex());
  d_currentTime = e.getTime();
}

void Simulator::updatePropensities(unsigned long reactionIndex) {
  Vector<unsigned long> *dependencies =
d_localDependencies.getPtr(reactionIndex);
  for(int i=0; i<dependencies->size(); i++) {
    unsigned long localIndex = dependencies->get(i);
    unsigned long globalIndex = d_assignedReactions->get(localIndex);
    Reaction *reaction = d_model->getReactions()->getPtr(globalIndex);
    d_sumTree->set(localIndex,reaction->getPropensity(&d_species));
  }
}

void Simulator::rollbackComplete(double time) {
  d_currentTime = time;
  d_randomNumber.back();
  d_randomNumber.back();
  generateNewEvent();
}
```

```
Event Simulator::getNextEvent() {
  return d_event;
}
```

Simulator.h

```
#ifndef SIMULATOR_H
#define SIMULATOR_H

#include "ParallelRandom.h"
#include "../common/Model.h"
#include "Event.h"
#include "SumTree.h"

class Simulator {
 public:
  Simulator();
  void init(int threadIndex, int threadCount,
            Model *model, Vector<unsigned long> *assignedReactions,
                                    int seed);
  void executeLocalEvent();
  void executeRemoteEvent(Event event);
  void unexecuteEvent(Event e);
  void rollbackComplete(double time);
  Event getNextEvent();
  double getCurrentTime() { return d_currentTime; }
  unsigned long getReactionsExecutedCount() { return
d_reactionsExecutedCount; }
  unsigned long getReactionsUnexecutedCount() { return
d_reactionsUnexecutedCount; }
  unsigned long getCurrentSpeciesValue(int speciesIndex) { return
d_species.get(speciesIndex); }

 private:
  void calculatePropensities();
  void updatePropensities(unsigned long reactionIndex);
  unsigned long selectReaction(double uniformRand);
  void generateNewEvent();
  ParallelRandom d_randomNumber;
  Model *d_model;
  Vector<unsigned long> *d_assignedReactions;
  double d_currentTime;
  Event d_event;
  Vector<long> d_species;
  unsigned long d_threadIndex;
  unsigned long d_reactionsExecutedCount;
  unsigned long d_reactionsUnexecutedCount;

  Vector< Vector<unsigned long> > d_localDependencies;
  SumTree *d_sumTree;
};
```

```
#endif
```

SortedEventList.cpp

```
#include "SortedEventList.h"
#include "../common/Exception.h"
#include <iostream>
#include <stdlib.h>

using namespace std;

SortedEventList::SortedEventList() {
  head = NULL;
  pthread_mutex_init(&mutex, NULL);
}

SortedEventList::~SortedEventList() {
  pthread_mutex_destroy(&mutex);
}

void SortedEventList::lock() {
  pthread_mutex_lock(&mutex);
}

void SortedEventList::unlock() {
  pthread_mutex_unlock(&mutex);
}

void SortedEventList::addEvent(Event event) {
  EventNode *temp = head;
  while (temp != NULL) {
    if (temp->event.cancels(event))
      break;
    temp = temp->next;
  }

  if (temp != NULL) {
    if (temp == head) {
      head = head->next;
      delete temp;
    } else {
      EventNode *temp2 = head;
      while (temp2->next != temp) {
                          temp2 = temp2->next;
      }
      temp2->next = temp->next;
      delete temp;
    }
  } else {
    EventNode *newNode = new EventNode();
    newNode->event = event;
    newNode->next = NULL;

    if (head == NULL) {
```

```
      head = newNode;
    } else if (head->event.getTime() > event.getTime()) {
      newNode->next = head;
      head = newNode;
    } else {
      EventNode *currentNode = head;
      while ((currentNode->next != NULL) &&
          (currentNode->next->event.getTime() < event.getTime())) {
                    currentNode = currentNode->next;
      }
      // The following code is added to allow antimessages to be
processed
      // ahead of messages with identical times.
      if (event.isAntiMessage()) {
                  newNode->next = currentNode->next;
                     currentNode->next = newNode;
      } else {
                while ((currentNode->next != NULL) &&
           (currentNode->next->event.getTime() == event.getTime())) {
                     currentNode = currentNode->next;
                                  }
                  newNode->next = currentNode->next;
                     currentNode->next = newNode;
      }
    }
  }
}

Event SortedEventList::getEarliestEvent() {
  EventNode *temp = head;
  Event event = head->event;
  head = head->next;
  delete temp;
  return event;
}

Event SortedEventList::peekEarliestEvent() {
  return head->event;
}

bool SortedEventList::isEmpty() {
  bool tmp = (head == NULL);
  return tmp;
}

unsigned long SortedEventList::getSize() {
  unsigned long i=0;
  EventNode *temp = head;
  while (temp != NULL) {
    i++;
    temp = temp->next;
  }
  return i;
}
```

SortedEventList.h

```
#ifndef SORTEDEVENTLIST_H
#define SORTEDEVENTLIST_H

#include "Event.h"
#include "EventNode.h"
#include <pthread.h>

class SortedEventList {
 public:
  SortedEventList();
  ~SortedEventList();
  void lock();
  void unlock();
  void addEvent(Event event);
  Event peekEarliestEvent();
  Event getEarliestEvent();
  bool isEmpty();
  unsigned long getSize();

 private:
  EventNode *head;
  pthread_mutex_t mutex;
};

#endif
```

SumTree.cpp

```
#include <stdlib.h>
#include <iostream>
#include "SumTree.h"

using namespace std;

SumTree::SumTree(unsigned long reactions) {
  reactionCount = reactions;

  int i = 1;
  size = 1;
  while (i < reactionCount) {
    i *= 2;
    size += i;
  }

  data = new double[size + 1];
  for(unsigned long i = 1; i <= size; i++) {
    data[i] = 0.0;
  }
}

double SumTree::getTotal() const {
```

```
  return data[1];
}

void SumTree::set(unsigned long reactionIndex, double value) {
  unsigned long index = size - reactionIndex;
  data[index] = value;
  index = index >> 1;
  while (index > 0) {
    data[index] = data[index*2] + data[index*2+1];
    index = index >> 1;
  }
}

unsigned long SumTree::selectReaction(double uniform) const {
  double selector = uniform*getTotal();
  int currentIndex = 1;
  while(2*currentIndex <= size) {
    if (selector < data[currentIndex*2]) {
      currentIndex = 2*currentIndex;
    } else {
      selector -= data[currentIndex*2];
      currentIndex = 2*currentIndex+1;
    }
  }
  return size - currentIndex;
}

void SumTree::print() const {
  int x = 1;
  int y = x;
  for(int i=1; i <= size; i++) {
    cout << data[i] << " ";
    y--;
    if (y == 0) {
      cout << endl;
      x *= 2;
      y = x;
    }
  }
  cout << endl;
}

// int main() {
//   SumTree x(5);
//   for(int i=1; i<=3; i++) {
//     x.set(i,0.33);
//   }

//   x.print();
//   for(int i=0; i<11; i++) {
//     double uni = (double)i/10.0;
//     cout << uni << " " << x.selectReaction(uni) << endl;
//   }
// }
```

SumTree.h

```
class SumTree {
 public:
  SumTree(unsigned long reactions);
  void set(unsigned long reactionIndex, double value);
  void print() const;
  unsigned long selectReaction(double uniform) const;
  double getTotal() const;

 private:
  double *data;
  unsigned long reactionCount;
  unsigned long size;
};
```

ThreadControl.cpp

```
#include "ThreadControl.h"
#include <iostream>

using namespace std;


ThreadControl::ThreadControl() {
  d_exit = 0;

  d_halt = 0;
  d_halted = 0;
}

void ThreadControl::halt() {
  d_halt = 1;
}

void ThreadControl::waitForHalt() {
  while (!d_halted) { }
}

void ThreadControl::resume() {
  d_halt = 0;
}

void ThreadControl::waitForResume() {
  while (d_halted) { }
}

int ThreadControl::haltCheck() {
  if (d_halt) {
    d_halted = 1;
    while (d_halt) { }
    d_halted = 0;
    return 1;
```

```
  }
  return 0;
}

void ThreadControl::exit() {
  d_exit = 1;
}

void ThreadControl::exitCheck() {
  if (d_exit) {
    pthread_exit(NULL);
  }
}
```

ThreadControl.h

```
#include <pthread.h>

class ThreadControl {
 public:
  ThreadControl();
  void halt();
  void waitForHalt();
  void waitForResume();
  void resume();
  void exit();
  int haltCheck();
  void exitCheck();

 private:
  volatile int d_halt;
  volatile int d_exit;
  volatile int d_halted;
  pthread_mutex_t d_haltLock;
  pthread_mutex_t d_resumeLock;
  pthread_mutex_t d_haltedLock;
  pthread_cond_t d_haltCond;
  pthread_cond_t d_resumeCond;
};
```

Model Generation Source Code

To generate model files programmatically for the Parallel Reaction Method's performance analysis, the following source code was written. To generate the ecoli model file, motif data must be downloaded from Uri Alon's website at http://www.weizmann.ac.il/mcb/UriAlon/

NewModelBuilder.java

```
// The following code is used to build performance analysis model files
// and assignments files for the Parallel Reaction Method.

import java.io.*;

class NewModelBuilder {
    public static void printReaction(PrintStream out,
                                     PrintStream assign,
                                     int r,
                                     int p,
                                     double rate,
                                     int assignment) {
        out.println("1 1 " + r + " 1 1 " + p + " " + rate);
        assign.println(assignment);
    }

    public static void main(String args[]) throws IOException {
        if (args.length != 5) {
            System.out.println("usage: java NewModelBuilder <erate> " +
                               "<irate1> <irate2> <model> <assign>");
            System.exit(1);
        }

        double erate = Double.parseDouble(args[0]);
        double irate1 = Double.parseDouble(args[1]);
        double irate2 = Double.parseDouble(args[2]);
        PrintStream model = new PrintStream(new
FileOutputStream(args[3]));
        PrintStream assign = new PrintStream(new
FileOutputStream(args[4]));

        model.println(18);
        for(int i=0; i<18; i++) {
            model.print(" 1000");
        }
        model.println();

        model.println(24*2+2);
        for(int cluster=0; cluster<2; cluster++) {
            int s = cluster*9;
            double rate;
            if (cluster == 0) {
                rate = irate1;
```

```
            } else {
                rate = irate2;
            }
            printReaction(model,assign,s,s+1,rate,cluster);
            printReaction(model,assign,s+1,s,rate,cluster);
            printReaction(model,assign,s,s+2,rate,cluster);
            printReaction(model,assign,s+2,s,rate,cluster);
            printReaction(model,assign,s+1,s+3,rate,cluster);
            printReaction(model,assign,s+3,s+1,rate,cluster);
            printReaction(model,assign,s+1,s+4,rate,cluster);
            printReaction(model,assign,s+4,s+1,rate,cluster);
            printReaction(model,assign,s+2,s+4,rate,cluster);
            printReaction(model,assign,s+4,s+2,rate,cluster);
            printReaction(model,assign,s+2,s+5,rate,cluster);
            printReaction(model,assign,s+5,s+2,rate,cluster);
            printReaction(model,assign,s+3,s+6,rate,cluster);
            printReaction(model,assign,s+6,s+3,rate,cluster);
            printReaction(model,assign,s+4,s+6,rate,cluster);
            printReaction(model,assign,s+6,s+4,rate,cluster);
            printReaction(model,assign,s+4,s+7,rate,cluster);
            printReaction(model,assign,s+7,s+4,rate,cluster);
            printReaction(model,assign,s+5,s+7,rate,cluster);
            printReaction(model,assign,s+7,s+5,rate,cluster);
            printReaction(model,assign,s+6,s+8,rate,cluster);
            printReaction(model,assign,s+8,s+6,rate,cluster);
            printReaction(model,assign,s+7,s+8,rate,cluster);
            printReaction(model,assign,s+8,s+7,rate,cluster);
        }
        printReaction(model,assign,0,9,erate,0);
        printReaction(model,assign,9,0,erate,1);


        model.println(2);
        model.println("0 9");

        model.close();
        assign.close();
    }
}
```

Ecoli.java

```
// The following code is used to generate the large scale ecoli model
// file and reaction assignments file

import java.io.*;
import java.util.Vector;

class EColi {
    public static void main(String args[]) throws IOException {
        if (args.length != 4) {
            System.out.println("usage: java EColi <unbindRate> <procs>
<modelfile> <assignmentfile>");
            System.exit(1);
        }
```

```
        double rate = Double.parseDouble(args[0]);
        int procs = Integer.parseInt(args[1]);
        String modelFilename = args[2];
        String assignmentsFilename = args[3];

        Vector names = new Vector();

        BufferedReader in = new BufferedReader(new
FileReader("coliInterFullNames.txt"));
        String str;
        while ((str = in.readLine()) != null) {
            String strs[] = str.split(" ");
            names.add(strs[1]);
        }
        in.close();

        Vector connections = new Vector();
        in = new BufferedReader(new
FileReader("coliInterFullVec.txt"));
        while ((str = in.readLine()) != null) {
            connections.add(str);
        }
        in.close();

        PrintStream assignments = new PrintStream(new
FileOutputStream(assignmentsFilename));
        PrintStream model = new PrintStream(new
FileOutputStream(modelFilename));

        Vector species = new Vector();
        Vector reactions = new Vector();

        for(int i=0; i<names.size(); i++) {
            int S = i*4;
            species.add(new Integer(1));
            int M = i*4+1;
            species.add(new Integer(2));
            int P = i*4+2;
            species.add(new Integer(50));
            int D = i*4+3;
            species.add(new Integer(500));

            reactions.add("1 1 " + S + " 2 1 " + S + " 1 " + M + "
0.0144405");
            assignments.println(i % procs);
            reactions.add("1 1 " + M + " 0 0.00577622");
            assignments.println(i % procs);
            reactions.add("1 1 " + M + " 2 1 " + M + " 1 " + P + "
0.11552453");
            assignments.println(i % procs);
            reactions.add("1 1 " + P + " 0 2.88811E-4");
            assignments.println(i % procs);
            reactions.add("1 2 " + P + " 1 1 " + D + " 2.88811E-4");
            assignments.println(i % procs);
```

```
            reactions.add("1 1 " + D + " 1 2 " + P + " 0.00288811");
            assignments.println(i % procs);
            reactions.add("1 1 " + D + " 0 2.88811E-4");
            assignments.println(i % procs);
        }

        for(int i=0; i<connections.size(); i++) {
            String strs[] = ((String)connections.get(i)).split(" ");
            int tf = (Integer.parseInt(strs[0])-1)*4 + 3;
            int src = (Integer.parseInt(strs[1])-1)*4;
            int m = src + 1;
            int type = Integer.parseInt(strs[2]);

            if ((type == 1) || (type == 3)) {
                int tfsrc = species.size();
                reactions.add("2 1 " + tf + " 1 " + src + " 1 1 " +
tfsrc + " " + (rate/100));
                assignments.println((Integer.parseInt(strs[0])-1) %
procs);
                reactions.add("1 1 " + tfsrc + " 2 1 " + tf + " 1 " +
src + " " + rate);
                assignments.println((Integer.parseInt(strs[0])-1) %
procs);
                reactions.add("1 1 " + tfsrc + " 2 1 " + tfsrc + " 1 "
+ m + " " + " 0.144405");
                assignments.println((Integer.parseInt(strs[1])-1) %
procs);
                species.add(new Integer(0));
            }

            if ((type == 2) || (type == 3)) {
                int tfsrc = species.size();
                reactions.add("2 1 " + tf + " 1 " + src + " 1 1 " +
tfsrc + " " + (rate/100));
                assignments.println((Integer.parseInt(strs[0])-1) %
procs);
                reactions.add("1 1 " + tfsrc + " 2 1 " + tf + " 1 " +
src + " " + rate);
                assignments.println((Integer.parseInt(strs[0])-1) %
procs);
                reactions.add("1 1 " + tfsrc + " 2 1 " + tfsrc + " 1 "
+ m + " " + " 0.000144405");
                assignments.println((Integer.parseInt(strs[1])-1) %
procs);
                species.add(new Integer(0));
            }
        }

        model.println(species.size());
        for(int i=0; i<species.size(); i++) {
            if (i % 10 == 0) model.println();
            model.print(" " + (Integer)species.get(i));
        }
        model.println();
        model.println(reactions.size());
```

```
        for(int i=0; i<reactions.size(); i++) {
            model.println(reactions.get(i));
        }
        model.println();
        model.println(20);
        for(int i=0; i<10; i++) {
            model.println(i*4);
            model.println(i*4+3);
        }
        model.println();
        model.println();

        assignments.close();
    }
}
```

Distributed Computing Engine

The following source code is used to implement the Distributed Computing Engine in a tool called BioGrid, which is used to perform parameter sweeps for biological systems. The BioGrid system also include an interface to the data post-processing program Octave.

Server/Server.java

```
import java.net.*;
import java.io.*;
import java.util.*;

public class Server {
    private ServerSocket serverSocket;
    ServerDatabase database;

    public Server() {
      database = new ServerDatabase();
    }

    public void start(int port) throws IOException {
      try {
          serverSocket = new ServerSocket(port);
          System.out.println("Server started at " +
                          InetAddress.getLocalHost().getHostName() +
                          "(" +
InetAddress.getLocalHost().getHostAddress() + ") " +
                          "port " + port);
      } catch (IOException e) {
          throw new IOException("Unable to start the server: " +
e.getMessage());
      }
    }

    public void acceptConnection() throws IOException {
      ServerUser user = new ServerUser(serverSocket.accept(),database);
      user.start();
    }

    public static void printUsage() {
      System.out.println("usage:  java Server <port>");
    }

    public static void main(String args[]) {
      if (args.length != 1) {
          printUsage();
      } else {
          int port = 0;
          try {
            port = Integer.parseInt(args[0]);
          } catch (NumberFormatException e) {
```

```
            printUsage();
            System.exit(1);
          }

          System.out.println();
          System.out.println("BioGrid Server v1.0");
          System.out.println("written by James M. McCollum
(jmccoll2@utk.edu)");
          System.out.println("University of Tennessee - Knoxville");
          System.out.println("http://biocomp.ece.utk.edu");
          System.out.println();

          Server server = new Server();
          try {
            server.start(port);
          } catch (IOException e) {
            System.out.println(e.getMessage());
            System.exit(1);
          }

          while (true) {
            try {
                server.acceptConnection();
            } catch (IOException e) {
                System.out.println(e.getMessage());
            }
          }
      }
    }
}
```

Server/FileInfo.java

```
import java.io.*;

public class FileInfo {
    private FileId id = null;
    private UserId owner = null;
    private UserId creator = null;
    private boolean exists = false;
    private JobId dependsOn = null;
    private String serverFilePath = "";

    public Object clone() {
      FileInfo temp = new FileInfo();
      temp.id = id;
      temp.owner = owner;
      temp.creator = creator;
      temp.exists = exists;
      temp.dependsOn = dependsOn;
      temp.serverFilePath = serverFilePath;
      return temp;
    }

    public void write(VerifiedOutputStream out) throws IOException {
```

```
      out.writeString("FileInfo");
      if (id == null) out.writeBoolean(false); else {
out.writeBoolean(true); id.write(out); }
      if (owner == null) out.writeBoolean(false); else {
out.writeBoolean(true); owner.write(out); }
      if (creator == null) out.writeBoolean(false); else {
out.writeBoolean(true); creator.write(out); }
      out.writeBoolean(exists);
      if (dependsOn == null) out.writeBoolean(false); else {
out.writeBoolean(true); dependsOn.write(out); }
      out.writeString(serverFilePath);
    }

    public static FileInfo read(VerifiedInputStream in) throws
IOException {
      if (!in.readString().equals("FileInfo")) {
          throw new IOException("Error reading FileInfo.");
      }
      FileInfo fileInfo = new FileInfo();
      if (in.readBoolean()) fileInfo.id = FileId.read(in);
      if (in.readBoolean()) fileInfo.owner = UserId.read(in);
      if (in.readBoolean()) fileInfo.creator = UserId.read(in);
      fileInfo.exists = in.readBoolean();
      if (in.readBoolean()) fileInfo.dependsOn = JobId.read(in);
      fileInfo.serverFilePath = in.readString();
      return fileInfo;
    }

    public void setId(FileId id) {
      this.id = id;
    }

    public void setOwner(UserId owner) {
      this.owner = owner;
    }

    public void setCreator(UserId creator) {
      this.creator = creator;
    }

    public void setExists(boolean exists) {
      this.exists = exists;
    }

    public void setJobDependency(JobId job) {
      this.dependsOn = job;
    }

    public void setServerFilePath(String path) {
      this.serverFilePath = path;
    }

    public FileId getId() {
      return this.id;
    }
```

```
    public UserId getOwner() {
      return this.owner;
    }

    public UserId getCreator() {
      return this.creator;
    }

    public boolean getExists() {
      return this.exists;
    }

    public JobId getJobDependency() {
      return this.dependsOn;
    }

    public String getServerFilePath() {
      return this.serverFilePath;
    }

    public String toString() {
      String returnval = "" + id;
      returnval += " owner:" + owner;
      returnval += " creator:" + creator;
      returnval += " exists:" + exists;
      returnval += " dependsOn:" + dependsOn;
      returnval += " filePath:" + serverFilePath;
      return returnval;
    }
}
```

Server/JobArray.java

```
public class JobArray {
    private Job array[] = new Job[0];

    public JobArray() {

    }

    public synchronized void add(Job item) {
      Job temp[] = new Job[array.length+1];
      for(int i=0; i<array.length; i++) {
          temp[i] = array[i];
      }
      temp[array.length] = item;
      array = temp;
    }

    public synchronized Job get(int index) {
      return array[index];
    }

    public synchronized void set(int index, Job job) {
```

```
      array[index] = job;
    }

    public synchronized int find(Object obj) {
      for(int i=0; i<array.length; i++) {
          if (array[i].equals(obj)) {
            return i;
          }
      }
      return -1;
    }

    public synchronized void remove(int index) {
      if ((index < 0) || (index >= array.length)) {
          throw new ArrayIndexOutOfBoundsException();
      }
      Job temp[] = new Job[array.length-1];
      for(int i=0; i<index; i++) {
          temp[i] = array[i];
      }
      for(int i=index+1; i<array.length; i++) {
          temp[i-1] = array[i];
      }
      array = temp;
    }

    public synchronized void remove(Job obj) {
      int index = find(obj);
      remove(index);
    }

    public synchronized String toString() {
      String returnValue = "[";
      for(int i=0; i<array.length; i++) {
          returnValue += array[i].toString() + " ";
      }
      returnValue += "]";
      return returnValue;
    }

    public synchronized int size() {
      return array.length;
    }
}
```

Server/UserInfo.java

```
import java.io.*;

public class UserInfo {
    private UserId id = null;
    private String hostName = "";
    private String hostAddress = "";
    private String type = "";
    private String action = "";
```

```
private JobId jobId = null;;
private double progress = 0.0;

public UserInfo() {

}

public void setId(UserId id) {
  this.id = id;
}

public void setHostName(String hostName) {
  this.hostName = hostName;
}

public void setHostAddress(String hostAddress) {
  this.hostAddress = hostAddress;
}

public void setType(String type) {
  this.type = type;
}

public void setAction(String action) {
  this.action = action;
}

public void setJobId(JobId jobId) {
  this.jobId = jobId;
}

public void setProgress(double progress) {
  this.progress = progress;
}

public UserId getId() {
  return id;
}

public String getHostName() {
  return hostName;
}

public String getHostAddress() {
  return hostAddress;
}

public String getType() {
  return type;
}

public String getAction() {
  return action;
}
```

```
    public JobId getJobId() {
      return jobId;
    }

    public double getProgress() {
      return progress;
    }

    public String toString() {
      String str = "";
      str += id.toString() + " " + getHostName() + "(" +
getHostAddress() + ") ";
      str += getType() + " " + getAction() + " " + getJobId() + " " +
getProgress();
      return str;
    }

    public static UserInfo read(VerifiedInputStream in) throws
IOException {
      if (!in.readString().equals("UserInfo")) {
          throw new IOException("Error reading UserInfo.");
      }
      UserInfo info = new UserInfo();
      if (in.readBoolean()) info.id = UserId.read(in);
      info.hostName = in.readString();
      info.hostAddress = in.readString();
      info.type = in.readString();
      info.action = in.readString();
      if (in.readBoolean()) info.jobId = JobId.read(in);
      info.progress = in.readDouble();
      return info;
    }

    public void write(VerifiedOutputStream out) throws IOException {
      out.writeString("UserInfo");
      if (id == null) { out.writeBoolean(false); } else {
out.writeBoolean(true); id.write(out); }
      out.writeString(hostName);
      out.writeString(hostAddress);
      out.writeString(type);
      out.writeString(action);
      if (jobId == null) { out.writeBoolean(false); } else {
out.writeBoolean(true); jobId.write(out); }
      out.writeDouble(progress);
    }
}
```

Server/Command.java

```
import java.io.*;

public abstract class Command {
    public abstract String getCommandName();
    public abstract int getInputCount();
    public abstract int getOutputCount();
```

```
    public abstract void run(Worker worker,
                        File inputs[],
                        File outputs[]) throws
ServerException,IOException;

    public CommandSignature getCommandSignature() {
      return new
CommandSignature(getCommandName(),getInputCount(),getOutputCount());
    }
}
```

Server/CommandSignature.java

```
import java.io.*;

public class CommandSignature {
    private String command;
    private int inputCount;
    private int outputCount;

    public CommandSignature(String command, int inputCount, int
outputCount) {
      this.command = command;
      this.inputCount = inputCount;
      this.outputCount = outputCount;
    }

    public String getCommand() {
      return command;
    }

    public int getInputCount() {
      return inputCount;
    }

    public int getOutputCount() {
      return outputCount;
    }

    public boolean equals(Object obj) {
      if (obj.getClass() == CommandSignature.class) {
          CommandSignature temp = (CommandSignature)obj;
          if (temp.command.equals(this.command) &&
            (temp.inputCount == this.inputCount) &&
            (temp.outputCount == this.outputCount)) {
            return true;
          }
      }
      return false;
    }

    public void write(VerifiedOutputStream out) throws IOException {
      out.writeString("CommandSignature");
      out.writeString(command);
      out.writeInt(inputCount);
```

```
      out.writeInt(outputCount);
    }

    public static CommandSignature read(VerifiedInputStream in) throws
IOException {
      if (!in.readString().equals("CommandSignature")) {
          throw new IOException("Error reading command signature.");
      }
      String tempCommand = in.readString();
      int tempInputCount = in.readInt();
      int tempOutputCount = in.readInt();
      return new
CommandSignature(tempCommand,tempInputCount,tempOutputCount);
    }
}
```

Server/FileId.java

```
import java.io.*;

public class FileId {
    long id;

    public FileId(long id) {
      this.id = id;
    }

    public void write(VerifiedOutputStream out) throws IOException {
      out.writeString("FileId");
      out.writeLong(id);
    }

    public static FileId read(VerifiedInputStream in) throws
IOException {
      String str = in.readString();
      if (!str.equals("FileId")) {
          throw new IOException("Error reading FileId.");
      }
      return new FileId(in.readLong());
    }

    public boolean equals(Object obj) {
      if (obj.getClass() == FileId.class) {
          if (((FileId)obj).id == this.id) {
            return true;
          }
      }
      return false;
    }

    public String toString() {
      return "file" + id;
    }
}
```

Server/FileInfoArray.java

```
public class FileInfoArray {
    private FileInfo array[] = new FileInfo[0];

    public FileInfoArray() {

    }

    public synchronized void add(FileInfo item) {
      FileInfo temp[] = new FileInfo[array.length+1];
      for(int i=0; i<array.length; i++) {
          temp[i] = array[i];
      }
      temp[array.length] = item;
      array = temp;
    }

    public synchronized FileInfo get(int index) {
      return array[index];
    }

    public synchronized int find(Object obj) {
      for(int i=0; i<array.length; i++) {
          if (array[i].equals(obj)) {
            return i;
          }
      }
      return -1;
    }

    public synchronized void remove(int index) {
      if ((index < 0) || (index >= array.length)) {
          throw new ArrayIndexOutOfBoundsException();
      }
      FileInfo temp[] = new FileInfo[array.length-1];
      for(int i=0; i<index; i++) {
          temp[i] = array[i];
      }
      for(int i=index+1; i<array.length; i++) {
          temp[i-1] = array[i];
      }
      array = temp;
    }

    public synchronized void remove(FileInfo obj) {
      int index = find(obj);
      remove(index);
    }

    public synchronized String toString() {
      String returnValue = "[";
      for(int i=0; i<array.length; i++) {
          returnValue += array[i].toString() + " ";
      }
```

```
      returnValue += "]";
      return returnValue;
    }

    public synchronized int size() {
      return array.length;
    }
}
```

Server/Job.java

```
import java.io.*;

public class Job {
    private JobId id;
    private String command;
    private FileId inputs[];
    private FileId outputs[];
    private UserId owner;
    private UserId worker = null;
    private JobStatus status = JobStatus.PENDING;
    private String errorMessage = "";
    private double progress = 0.0;

    public Job(JobId id, String command, FileId[] inputs, FileId[]
outputs, UserId owner) {
      this.id = id;
      this.command = command;
      this.inputs = inputs;
      this.outputs = outputs;
      this.owner = owner;
    }

    public static Job read(VerifiedInputStream in) throws IOException {
      if (!in.readString().equals("Job")) {
          throw new IOException("Error reading job.");
      }
      JobId tempId = JobId.read(in);
      String tempCommand = in.readString();
      int inputCount = in.readInt();
      FileId tempInput[] = new FileId[inputCount];
      for(int i=0; i<inputCount; i++) {
          tempInput[i] = FileId.read(in);
      }
      int outputCount = in.readInt();
      FileId tempOutput[] = new FileId[outputCount];
      for(int i=0; i<outputCount; i++) {
          tempOutput[i] = FileId.read(in);
      }
      UserId tempOwner = UserId.read(in);
      Job job = new
Job(tempId,tempCommand,tempInput,tempOutput,tempOwner);
      if (in.readBoolean()) { job.setWorker(UserId.read(in)); }
      job.setStatus(JobStatus.read(in));
      job.errorMessage = in.readString();
```

```
      job.progress = in.readDouble();
      return job;
    }

    public void write(VerifiedOutputStream out) throws IOException {
      out.writeString("Job");
      id.write(out);
      out.writeString(command);
      out.writeInt(inputs.length);
      for(int i=0; i<inputs.length; i++) {
          inputs[i].write(out);
      }
      out.writeInt(outputs.length);
      for(int i=0; i<outputs.length; i++) {
          outputs[i].write(out);
      }
      owner.write(out);
      if (worker == null) { out.writeBoolean(false); } else {
out.writeBoolean(true); worker.write(out); }
      status.write(out);
      out.writeString(errorMessage);
      out.writeDouble(progress);
    }

    public Object clone() {
      Job job = new Job(id,command,inputs,outputs,owner);
      job.setWorker(worker);
      job.setStatus(status);
      job.setErrorMessage(errorMessage);
      job.setProgress(progress);
      return job;
    }

    public void setWorker(UserId worker) {
      this.worker = worker;
    }

    public void setStatus(JobStatus status) {
      this.status = status;
    }

    public void setErrorMessage(String errorMessage) {
      this.errorMessage = errorMessage;
    }

    public void setProgress(double progress) {
      this.progress = progress;
    }

    public JobId getId() {
      return id;
    }

    public String getCommand() {
      return command;
```

```
    }

    public FileId[] getInputs() {
      return inputs;
    }

    public FileId[] getOutputs() {
      return outputs;
    }

    public UserId getOwner() {
      return owner;
    }

    public UserId getWorker() {
      return worker;
    }

    public JobStatus getStatus() {
      return status;
    }

    public String getErrorMessage() {
      return errorMessage;
    }

    public double getProgress() {
      return progress;
    }

    public String toString() {
      String str = "";
      str += id.toString() + " (";
      for(int i=0; i<outputs.length; i++) {
          if (i != 0) str += ",";
          str += outputs[i];
      }
      str += ")=" + command + "(";
      for(int i=0; i<inputs.length; i++) {
          if (i != 0) str += ",";
          str += inputs[i];
      }
      str += ") " + owner + " " + worker + " " + status;
      str += " " + progress + " " + errorMessage;
      return str;
    }

    public CommandSignature getCommandSignature() {
      return new
CommandSignature(command,inputs.length,outputs.length);
    }
}
```

Server/JobId.java

```
import java.io.*;

public class JobId {
    long id;

    public JobId(long id) {
      this.id = id;
    }

    public void write(VerifiedOutputStream out) throws IOException {
      out.writeString("JobId");
      out.writeLong(id);
    }

    public static JobId read(VerifiedInputStream in) throws IOException
{
      if (!in.readString().equals("JobId")) {
          throw new IOException("Error reading JobId.");
      }
      return new JobId(in.readLong());
    }

    public boolean equals(Object obj) {
      if (obj.getClass() == JobId.class) {
          if (((JobId)obj).id == this.id) {
            return true;
          }
      }
      return false;
    }

    public String toString() {
      return "job" + id;
    }
}
```

Server/JobStatus.java

```
import java.io.*;

public class JobStatus {
    String status;

    private JobStatus(String status) {
      this.status = status;
    }

    public String toString() {
      return status;
    }

    public void write(VerifiedOutputStream out) throws IOException {
      out.writeString("JobStatus");
      out.writeString(status);
    }
```

```
    public static JobStatus read(VerifiedInputStream in) throws
IOException {
      if (!in.readString().equals("JobStatus")) {
          throw new IOException("Error reading job status");
      }
      String temp = in.readString();
      if (temp.equals("Pending")) {
          return PENDING;
      } else if (temp.equals("Running")) {
          return RUNNING;
      } else if (temp.equals("Error")) {
          return ERROR;
      } else if (temp.equals("Complete")) {
          return COMPLETE;
      } else {
          throw new IOException("Error reading job status");
      }
    }

    public static final JobStatus PENDING = new JobStatus("Pending");
    public static final JobStatus RUNNING = new JobStatus("Running");
    public static final JobStatus ERROR = new JobStatus("Error");
    public static final JobStatus COMPLETE  = new
JobStatus("Complete");
}
```

Server/ServerDatabase.java

```
import java.io.*;

public class ServerDatabase {
    private ServerUserArray users = new ServerUserArray();
    private FileInfoArray files = new FileInfoArray();
    private JobArray jobs = new JobArray();
    private JobArray pendingJobs = new JobArray();
    private ServerUserArray waitingUsers = new ServerUserArray();
    private long currentFileNumber = 0;
    private long currentUserNumber = 0;
    private long currentJobNumber = 0;

    private synchronized UserId getNewUserId() {
      currentUserNumber = currentUserNumber % 10000000;
      return new UserId(++currentUserNumber);
    }

    private synchronized FileId getNewFileId() {
      currentFileNumber = currentFileNumber % 10000000;
      return new FileId(++currentFileNumber);
    }

    private synchronized JobId getNewJobId() {
      currentJobNumber = currentJobNumber % 10000000;
      return new JobId(++currentJobNumber);
    }
```

```
public synchronized void addUser(ServerUser user) {
  users.add(user);
  user.setId(getNewUserId());
}

public synchronized void deleteUser(UserId userId) {
  for(int i=0; i<users.size(); i++) {
      ServerUser temp = users.get(i);
      if (temp.getId().equals(userId)) {
        users.remove(temp);
        break;
      }
  }

  JobArray jobsToDelete = new JobArray();
  for(int i=0; i<jobs.size(); i++) {
      Job job = jobs.get(i);
      if (job.getOwner().equals(userId)) {
        jobsToDelete.add(job);
      }
  }

  FileInfoArray filesToDelete = new FileInfoArray();
  for(int i=0; i<files.size(); i++) {
      FileInfo fileInfo = files.get(i);
      if (fileInfo.getOwner().equals(userId)) {
        filesToDelete.add(fileInfo);
      }
  }

  for(int i=0; i<jobsToDelete.size(); i++) {
      deleteJob(jobsToDelete.get(i).getId());
  }

  for(int i=0; i<filesToDelete.size(); i++) {
      deleteFile(filesToDelete.get(i).getId());
  }
}

public synchronized FileId addFile(File file, UserId owner) {
  FileInfo fileInfo = new FileInfo();
  fileInfo.setId(getNewFileId());
  fileInfo.setOwner(owner);
  fileInfo.setCreator(owner);
  fileInfo.setExists(true);
  fileInfo.setJobDependency(null);
  fileInfo.setServerFilePath(file.getAbsolutePath());
  files.add(fileInfo);
  return fileInfo.getId();
}

public synchronized FileInfo getFile(FileId fileId) {
  for(int i=0; i<files.size(); i++) {
      FileInfo temp = files.get(i);
```

```
            if (temp.getId().equals(fileId)) {
              return (FileInfo)temp.clone();
            }
        }
        return null;
    }

    public synchronized void updateFile(FileId fileId, File file,UserId
creator) throws IOException {
        for(int i=0; i<files.size(); i++) {
            FileInfo fileInfo = files.get(i);
            if (fileInfo.getId().equals(fileId)) {
              fileInfo.setCreator(creator);
              fileInfo.setExists(true);
              fileInfo.setJobDependency(null);
              fileInfo.setServerFilePath(file.getAbsolutePath());
              return;
            }
        }
        assignWork();
    }

    public synchronized void deleteFile(FileId fileId) {
        for(int i=0; i<files.size(); i++) {
            FileInfo temp = files.get(i);
            if (temp.getId().equals(fileId)) {
              files.remove(temp);
              (new File(temp.getServerFilePath())).delete();
              return;
            }
        }
    }

    public synchronized FileInfo[] getAllFileInfo() {
        FileInfo[] fileInfo = new FileInfo[files.size()];
        for(int i=0; i<files.size(); i++) {
            fileInfo[i] = (FileInfo)files.get(i).clone();
        }
        return fileInfo;
    }

    public synchronized UserInfo[] getAllUserInfo() {
        UserInfo[] userInfo = new UserInfo[users.size()];
        for(int i=0; i<users.size(); i++) {
            userInfo[i] = (UserInfo)users.get(i).getInfo();
        }
        return userInfo;
    }

    public synchronized boolean isJobReady(Job job) {
        for(int i=0; i<job.getInputs().length; i++) {
            if (!getFile((job.getInputs())[i]).getExists()) {
              return false;
            }
        }
```

```
      return true;
    }

    public synchronized void assignWork() {
      boolean done = false;
      while(!done) {
          done = true;
          for(int i=0; i<pendingJobs.size(); i++) {
            Job job = pendingJobs.get(i);
            if (isJobReady(job)) {
                for(int j=0; j<waitingUsers.size(); j++) {
                  ServerUser user = waitingUsers.get(j);
                  if (user.canExecute(job)) {
                      pendingJobs.remove(job);
                      waitingUsers.remove(user);
                      try {
                        user.getPendingJobCallback(job);
                      } catch (IOException e) {

                      }
                      done = false;
                  }
                }
            }
          }
      }
    }

    public synchronized JobId addJob(String command, FileId[] inputs,
int outputCount, UserId owner) throws IOException {
      JobId id = getNewJobId();
      FileInfo[] outputFileInfo = new FileInfo[outputCount];
      FileId[] outputs = new FileId[outputCount];
      for(int i=0; i<outputFileInfo.length; i++) {
          outputFileInfo[i] = new FileInfo();
          outputFileInfo[i].setId(getNewFileId());
          outputFileInfo[i].setOwner(owner);
          outputFileInfo[i].setCreator(null);
          outputFileInfo[i].setExists(false);
          outputFileInfo[i].setJobDependency(id);
          outputFileInfo[i].setServerFilePath("");
          files.add(outputFileInfo[i]);
          outputs[i] = outputFileInfo[i].getId();
      }
      Job job = new Job(id,command,inputs,outputs,owner);
      jobs.add(job);
      pendingJobs.add(job);
      assignWork();
      return id;
    }

    public synchronized Job getJob(JobId jobId) {
      for(int i=0; i<jobs.size(); i++) {
          Job job = jobs.get(i);
          if (job.getId().equals(jobId)) {
```

```
            return (Job)job.clone();
          }
        }
        return null;
      }

      public synchronized Job[] getAllJobs() {
        Job[] temp = new Job[jobs.size()];
        for(int i=0; i<jobs.size(); i++) {
          temp[i] = (Job)jobs.get(i).clone();
        }
        return temp;
      }

      public synchronized void updateJob(Job job) {
        for(int i=0; i<jobs.size(); i++) {
          if (jobs.get(i).getId().equals(job.getId())) {
            jobs.set(i,job);
            break;
          }
        }
      }

      public synchronized void getPendingJob(ServerUser user) throws
IOException {
        waitingUsers.add(user);
        assignWork();
      }

      public synchronized void deleteJob(JobId id) {
        for(int i=0; i<jobs.size(); i++) {
          Job job = jobs.get(i);
          if (job.getId().equals(id)) {
            jobs.remove(job);
            break;
          }
        }
        for(int i=0; i<pendingJobs.size(); i++) {
          Job job = pendingJobs.get(i);
          if (job.getId().equals(id)) {
            pendingJobs.remove(job);
            break;
          }
        }
      }
}
```

Server/ServerException.java

```
public class ServerException extends Exception {
    public ServerException(String message) {
      super(message);
    }
}
```

Server/ServerUserArray.java

```
public class ServerUserArray {
    private ServerUser array[] = new ServerUser[0];

    public ServerUserArray() {

    }

    public synchronized void add(ServerUser item) {
      ServerUser temp[] = new ServerUser[array.length+1];
      for(int i=0; i<array.length; i++) {
          temp[i] = array[i];
      }
      temp[array.length] = item;
      array = temp;
    }

    public synchronized ServerUser get(int index) {
      return array[index];
    }

    public synchronized int find(Object obj) {
      for(int i=0; i<array.length; i++) {
          if (array[i].equals(obj)) {
            return i;
          }
      }
      return -1;
    }

    public synchronized void remove(int index) {
      if ((index < 0) || (index >= array.length)) {
          throw new ArrayIndexOutOfBoundsException();
      }
      ServerUser temp[] = new ServerUser[array.length-1];
      for(int i=0; i<index; i++) {
          temp[i] = array[i];
      }
      for(int i=index+1; i<array.length; i++) {
          temp[i-1] = array[i];
      }
      array = temp;
    }

    public synchronized void remove(ServerUser obj) {
      int index = find(obj);
      remove(index);
    }

    public synchronized String toString() {
      String returnValue = "[";
      for(int i=0; i<array.length; i++) {
          returnValue += array[i].toString() + " ";
      }
```

```
      returnValue += "]";
      return returnValue;
    }

    public synchronized int size() {
      return array.length;
    }
}
```

Server/UserId.java

```
import java.io.*;

public class UserId {
    long id;

    public UserId(long id) {
      this.id = id;
    }

    public void write(VerifiedOutputStream out) throws IOException {
      out.writeString("UserId");
      out.writeLong(id);
    }

    public static UserId read(VerifiedInputStream in) throws
IOException {
      if (!in.readString().equals("UserId")) {
          throw new IOException("Error reading UserId.");
      }
      return new UserId(in.readLong());
    }

    public boolean equals(Object obj) {
      if (obj.getClass() == UserId.class) {
          if (((UserId)obj).id == this.id) {
            return true;
          }
      }
      return false;
    }

    public String toString() {
      return "user" + id;
    }
}
```

Server/VerifiedInputStream.java

```
import java.io.*;

class VerifiedInputStream {
    private DataInputStream in;

    private final static int TYPE_BOOLEAN = 1;
```

```
    private final static int TYPE_CHAR = 2;
    private final static int TYPE_BYTE = 3;
    private final static int TYPE_SHORT = 4;
    private final static int TYPE_INT = 5;
    private final static int TYPE_LONG = 6;
    private final static int TYPE_FLOAT = 7;
    private final static int TYPE_DOUBLE = 8;
    private final static int TYPE_STRING = 9;
    private final static int TYPE_FILE = 10;

    public VerifiedInputStream(InputStream inputStream) throws
IOException {
      in = new DataInputStream(new BufferedInputStream(inputStream));
    }

    public void close() throws IOException {
      in.close();
    }

    public boolean readBoolean() throws IOException {
      verifyType(TYPE_BOOLEAN);
      return in.readBoolean();
    }

    public char readChar() throws IOException {
      verifyType(TYPE_CHAR);
      return in.readChar();
    }

    public byte readByte() throws IOException {
      verifyType(TYPE_BYTE);
      return in.readByte();
    }

    public short readShort() throws IOException {
      verifyType(TYPE_SHORT);
      return in.readShort();
    }

    public int readInt() throws IOException {
      verifyType(TYPE_INT);
      return in.readInt();
    }

    public long readLong() throws IOException {
      verifyType(TYPE_LONG);
      return in.readLong();
    }

    public float readFloat() throws IOException {
      verifyType(TYPE_FLOAT);
      return in.readFloat();
    }

    public double readDouble() throws IOException {
```

```
      verifyType(TYPE_DOUBLE);
      return in.readDouble();
    }

    public String readString() throws IOException {
      verifyType(TYPE_STRING);
      int stringLength = readInt();
      char string[] = new char[stringLength];
      for(int i=0; i<string.length; i++) {
          string[i] = in.readChar();
      }
      return new String(string);
    }

    public void readFile(File file) throws IOException {
      verifyType(TYPE_FILE);
      long fileLength = readLong();
      BufferedOutputStream out = new BufferedOutputStream(new
FileOutputStream(file));
      for(long i=0; i<fileLength; i++) {
          out.write(in.read());
      }
      out.flush();
      out.close();
    }

    private void verifyType(int expectedType) throws IOException {
      int type1 = in.readInt();
      int type2 = in.readInt();
      if ((type1 != type2) || (type1 < 1) || (type1 > 10)){
          throw new IOException("Error verifying the type of message to
receive.");
      } else if (type1 != expectedType) {
          throw new IOException("Attempted to receive data of type " +
getTypeText(expectedType) + ", but instead received data of type " +
getTypeText(type1) + ".");
      }
    }

    private String getTypeText(int type) {
      if (type == TYPE_BOOLEAN) return "boolean";
      else if (type == TYPE_CHAR) return "char";
      else if (type == TYPE_BYTE) return "byte";
      else if (type == TYPE_SHORT) return "short";
      else if (type == TYPE_INT) return "int";
      else if (type == TYPE_LONG) return "long";
      else if (type == TYPE_FLOAT) return "float";
      else if (type == TYPE_DOUBLE) return "double";
      else if (type == TYPE_STRING) return "string";
      else if (type == TYPE_FILE) return "file";
      else return "unknown";
    }
}
```

Server/VerifiedOutputStream.java

```
import java.io.*;

class VerifiedOutputStream {
    private DataOutputStream out;
    private BufferedOutputStream buffOut;

    private final static int TYPE_BOOLEAN = 1;
    private final static int TYPE_CHAR = 2;
    private final static int TYPE_BYTE = 3;
    private final static int TYPE_SHORT = 4;
    private final static int TYPE_INT = 5;
    private final static int TYPE_LONG = 6;
    private final static int TYPE_FLOAT = 7;
    private final static int TYPE_DOUBLE = 8;
    private final static int TYPE_STRING = 9;
    private final static int TYPE_FILE = 10;

    public VerifiedOutputStream(OutputStream outputStream) throws
IOException {
      buffOut = new BufferedOutputStream(outputStream);
      out = new DataOutputStream(buffOut);
    }

    public void close() throws IOException {
      out.close();
    }

    public void flush() throws IOException {
      buffOut.flush();
    }

    public void writeBoolean(boolean value) throws IOException {
      writeType(TYPE_BOOLEAN);
      out.writeBoolean(value);
    }

    public void writeChar(char value) throws IOException {
      writeType(TYPE_CHAR);
      out.writeChar(value);
    }

    public void writeByte(byte value) throws IOException {
      writeType(TYPE_BYTE);
      out.writeByte(value);
    }

    public void writeShort(short value) throws IOException {
      writeType(TYPE_SHORT);
      out.writeShort(value);
    }

    public void writeInt(int value) throws IOException {
      writeType(TYPE_INT);
      out.writeInt(value);
```

```
    }

    public void writeLong(long value) throws IOException {
      writeType(TYPE_LONG);
      out.writeLong(value);
    }

    public void writeFloat(float value) throws IOException {
      writeType(TYPE_FLOAT);
      out.writeFloat(value);
    }

    public void writeDouble(double value) throws IOException {
      writeType(TYPE_DOUBLE);
      out.writeDouble(value);
    }

    public void writeString(String value) throws IOException {
      writeType(TYPE_STRING);
      writeInt(value.length());
      out.writeChars(value);
    }

    public void writeFile(File file) throws IOException {
      FileInputStream fileIn = new FileInputStream(file);
      long fileLength = fileIn.getChannel().size();
      BufferedInputStream in = new BufferedInputStream(fileIn);
      writeType(TYPE_FILE);
      writeLong(fileLength);
      for(long i=0; i<fileLength; i++) {
          out.write(in.read());
      }
      in.close();
    }

    private void writeType(int type) throws IOException {
      out.writeInt(type);
      out.writeInt(type);
    }
}
```

Server/Worker.java

```
import java.io.*;

public class Worker {
    ServerConnection connection;
    Command[] commands = new Command[0];
    Job currentJob = null;

    public Worker(String host, int worker, String name) {
      try {
          connection = new ServerConnection(host,worker,name);
      } catch (IOException e) {
          if (e.getMessage() == null) {
```

```
            System.out.println("Unable to connect to the server.");
          } else {
            System.out.println("Unable to connect to the server: " +
e.getMessage());
          }
          System.exit(1);
      }
    }

    public void addCommand(Command command) {
      Command[] temp = new Command[commands.length+1];
      for(int i=0; i<commands.length; i++) {
          temp[i] = commands[i];
      }
      temp[commands.length] = command;
      commands = temp;
    }

    public void reportProgress(double progress) throws
IOException,ServerException {
      currentJob.setProgress(progress);
      connection.updateJob(currentJob);
    }

    private Command findCommand(Job job) throws IOException {
      Command command = null;
      for(int i=0; i<commands.length; i++) {
          if (job.getCommand().equals(commands[i].getCommandName())) {
            command = commands[i];
            break;
          }
      }
      if (command == null) {
          throw new IOException("Received an unknown command from the
server.");
      }
      return command;
    }

    private File[] createTempFileArray(int size) throws IOException {
      File[] files = new File[size];
      for(int i=0; i<files.length; i++) {
          files[i] = File.createTempFile("worker.","");
      }
      return files;
    }

    public void run() {
      CommandSignature signatures[] = new
CommandSignature[commands.length];
      for(int i=0; i<signatures.length; i++) {
          signatures[i] = commands[i].getCommandSignature();
      }
      while(true) {
          File[] inputFiles = null;
```

```
          File[] outputFiles = null;
          try {
            currentJob = connection.getPendingJob(signatures);
            currentJob.setStatus(JobStatus.RUNNING);
            currentJob.setWorker(connection.getId());
            reportProgress(0.0);
            System.out.println(currentJob.toString());
            Command command = findCommand(currentJob);
            FileId[] inputIds = currentJob.getInputs();
            FileId[] outputIds = currentJob.getOutputs();
            inputFiles = createTempFileArray(inputIds.length);
            outputFiles = createTempFileArray(outputIds.length);
            for(int i=0; i<inputFiles.length; i++) {
                connection.getFile(inputIds[i],inputFiles[i]);
            }
            try {
                command.run(this,inputFiles,outputFiles);
                for(int i=0; i<outputFiles.length; i++) {
                  connection.updateFile(outputIds[i],outputFiles[i]);
                }
                currentJob.setStatus(JobStatus.COMPLETE);
            } catch (ServerException e) {
                currentJob.setErrorMessage(e.getMessage());
                currentJob.setStatus(JobStatus.ERROR);
            }
            currentJob.setProgress(100.0);
            connection.updateJob(currentJob);
            deleteFiles(inputFiles);
            deleteFiles(outputFiles);
          } catch (IOException e) {
            deleteFiles(inputFiles);
            deleteFiles(outputFiles);
            if (e.getMessage() == null) {
                System.out.println("Connection to the server lost.");
            } else {
                System.out.println("Connection to the server lost: " +
e.getMessage());
            }
            System.exit(1);
          } catch (ServerException e) {
            deleteFiles(inputFiles);
            deleteFiles(outputFiles);
          }
      }
    }

    private void deleteFiles(File[] files) {
      if (files != null) {
          for(int i=0; i<files.length; i++) {
            files[i].delete();
          }
      }
    }
}
```

Server/ServerUser.java

```
import java.io.*;
import java.net.*;
import java.util.*;

public class ServerUser extends Thread {
    ServerDatabase database;
    Socket socket;
    VerifiedInputStream in;
    VerifiedOutputStream out;
    UserId id;
    String type = "Unknown";
    String action = "connect";
    Job job = null;
    CommandSignature signatures[];

    public ServerUser(Socket socket,ServerDatabase database) throws
IOException {
      this.database = database;
      this.socket = socket;
      in = new VerifiedInputStream(socket.getInputStream());
      out = new VerifiedOutputStream(socket.getOutputStream());
      database.addUser(this);
    }

    public void setId(UserId id) {
      this.id = id;
    }

    public synchronized UserInfo getInfo() {
      UserInfo info = new UserInfo();
      info.setId(id);
      info.setType(type);
      info.setAction(action);
      info.setHostName(socket.getInetAddress().getHostName());
      info.setHostAddress(socket.getInetAddress().getHostAddress());
      if (job == null) {
          info.setJobId(null);
          info.setProgress(0.0);
      } else {
          info.setJobId(job.getId());
          info.setProgress(job.getProgress());
      }
      return info;
    }

    public void run() {
      try {
          id.write(out);
          out.flush();
          type = in.readString();
          System.out.println(id.toString() + " connected as " + type +
".");
          while (true) {
```

```
            String command = in.readString();
            action = command;
            if (command.equals("addFile")) {
                addFile();
            } else if (command.equals("getFile")) {
                getFile();
            } else if (command.equals("updateFile")) {
                updateFile();
            } else if (command.equals("deleteFile")) {
                deleteFile();
            } else if (command.equals("getFileInfo")) {
                getFileInfo();
            } else if (command.equals("getAllFileInfo")) {
                getAllFileInfo();
            } else if (command.equals("getAllUserInfo")) {
                getAllUserInfo();
            } else if (command.equals("addJob")) {
                addJob();
            } else if (command.equals("getJob")) {
                getJob();
            } else if (command.equals("getAllJobs")) {
                getAllJobs();
            } else if (command.equals("updateJob")) {
                updateJob();
            } else if (command.equals("getPendingJob")) {
                getPendingJob();
            } else {
                throw new IOException("Unknown command: " + command);
            }
        }
    } catch (IOException e) {
        try { socket.close(); } catch (IOException junk) { }
        if (job != null) {
          job.setStatus(JobStatus.ERROR);
          job.setProgress(100.0);
          job.setErrorMessage("Connection to worker lost.");
          database.updateJob(job);
        }
        database.deleteUser(id);
        System.out.println("Connection to " + id + " lost.");
    }
}

private void addFile() throws IOException {
    File file = File.createTempFile("server.","");
    in.readFile(file);
    FileId fileId = database.addFile(file,this.id);
    out.writeString("Ok");
    fileId.write(out);
    out.flush();
}

private void getFile() throws IOException {
    FileId fileId = FileId.read(in);
    FileInfo fileInfo = database.getFile(fileId);
```

```
      if (fileInfo == null) {
          out.writeString("Invalid file id: " + fileId);
      } else if (fileInfo.getExists() == false) {
          out.writeString("Can not get " + fileId + " because it has
not been created yet.");
      } else {
          out.writeString("Ok");
          out.writeFile(new File(fileInfo.getServerFilePath()));
      }
      out.flush();
    }

    private void updateFile() throws IOException {
      FileId fileId = FileId.read(in);
      File file = File.createTempFile("server.","");
      in.readFile(file);
      FileInfo fileInfo = database.getFile(fileId);
      if (fileInfo == null) {
          out.writeString("Invalid file id: " + fileId);
          out.flush();
          file.delete();
      } else {
          database.updateFile(fileId,file,id);
          out.writeString("Ok");
          out.flush();
      }
    }

    private void deleteFile() throws IOException {
      FileId fileId = FileId.read(in);
      FileInfo fileInfo = database.getFile(fileId);
      if (fileInfo == null) {
          out.writeString("Invalid file id: " + fileId);
      } else {
          out.writeString("Ok");
          (new File(fileInfo.getServerFilePath())).delete();
          database.deleteFile(fileId);
      }
      out.flush();
    }

    private void getFileInfo() throws IOException {
      FileId[] fileId = new FileId[in.readInt()];
      for(int i=0; i<fileId.length; i++) {
          fileId[i] = FileId.read(in);
      }
      FileInfo[] fileInfo = new FileInfo[fileId.length];
      for(int i=0; i<fileInfo.length; i++) {
          fileInfo[i] = database.getFile(fileId[i]);
          if (fileInfo[i] == null) {
            out.writeString("Invalid file id: " + fileId[i]);
            out.flush();
            return;
          }
      }
```

```
  out.writeString("Ok");
  for(int i=0; i<fileInfo.length; i++) {
      fileInfo[i].write(out);
  }
  out.flush();
}

private void getAllFileInfo() throws IOException {
  FileInfo fileInfo[] = database.getAllFileInfo();
  out.writeString("Ok");
  out.writeInt(fileInfo.length);
  for(int i=0; i<fileInfo.length; i++) {
      fileInfo[i].write(out);
  }
  out.flush();
}

private void getAllUserInfo() throws IOException {
  UserInfo userInfo[] = database.getAllUserInfo();
  out.writeString("Ok");
  out.writeInt(userInfo.length);
  for(int i=0; i<userInfo.length; i++) {
      userInfo[i].write(out);
  }
  out.flush();
}

public UserId getId() {
  return id;
}

public void addJob() throws IOException {
  String command = in.readString();
  int inputCount = in.readInt();
  FileId[] inputs = new FileId[inputCount];
  for(int i=0; i<inputCount; i++) {
      inputs[i] = FileId.read(in);
  }
  int outputCount = in.readInt();
  for(int i=0; i<inputs.length; i++) {
      if (database.getFile(inputs[i]) == null) {
        out.writeString("Invalid input " + inputs[i]);
        out.flush();
        return;
      }
  }
  out.writeString("Ok");
  JobId jobId = database.addJob(command,inputs,outputCount,id);
  jobId.write(out);
  out.flush();
}

public void getJob() throws IOException {
  int jobCount = in.readInt();
  JobId[] jobId = new JobId[jobCount];
```

```
  for(int i=0; i<jobCount; i++) {
      jobId[i] = JobId.read(in);
  }
  Job[] job = new Job[jobCount];
  for(int i=0; i<jobCount; i++) {
      job[i] = database.getJob(jobId[i]);
      if (jobId[i] == null) {
        out.writeString("Invalid job id: " + jobId[i]);
        out.flush();
        return;
      }
  }
  out.writeString("Ok");
  for(int i=0; i<jobCount; i++) {
      job[i].write(out);
  }
  out.flush();
}

public void getAllJobs() throws IOException {
  out.writeString("Ok");
  Job[] job = database.getAllJobs();
  out.writeInt(job.length);
  for(int i=0; i<job.length; i++) {
      job[i].write(out);
  }
  out.flush();
}

public void getPendingJob() throws IOException {
  int signatureCount = in.readInt();
  signatures = new CommandSignature[signatureCount];
  for(int i=0; i<signatureCount; i++) {
      signatures[i] = CommandSignature.read(in);
  }
  database.getPendingJob(this);
}

public void getPendingJobCallback(Job job) throws IOException {
  this.job = job;
  out.writeString("Ok");
  job.write(out);
  out.flush();
}

public void updateJob() throws IOException {
  job = Job.read(in);
  database.updateJob(job);
  if (job.getStatus() != JobStatus.RUNNING) {
      job = null;
  }
  out.writeString("Ok");
  out.flush();
}
```

```
    public boolean canExecute(Job job) {
      for(int i=0; i<signatures.length; i++) {
          if (signatures[i].equals(job.getCommandSignature())) {
            return true;
          }
      }
      return false;
    }
}
```

Server/ServerConnection.java

```
import java.io.*;
import java.net.*;

public class ServerConnection {
    private Socket socket;
    private VerifiedInputStream in;
    private VerifiedOutputStream out;
    private UserId id;

    /// BEGIN -- FUNCTIONS THAT ARE VERY USEFUL TO THE CLIENT /////

    // host - The Moonshine server host.
    // port - The Moonshine server port.
    // type - A string that describes what is connecting (i.e.
"ESSWorker","MathWorker","Client",etc.)
    public ServerConnection(String host, int port, String type) throws
IOException {
      socket = new Socket(host,port);
      in = new VerifiedInputStream(socket.getInputStream());
      out = new VerifiedOutputStream(socket.getOutputStream());
      id = UserId.read(in);
      out.writeString(type);
      out.flush();
    }

    // close the connection to the server
    public void close() throws IOException {
      socket.close();
    }

    // Adds a file to the server and returns the corresponding file id
    public FileId addFile(File file) throws IOException,ServerException
{
      out.writeString("addFile");
      out.writeFile(file);
      out.flush();
      receiveOk();
      FileId id = FileId.read(in);
      return id;
    }

    // Reads the file with "fileId" from the server and writes the file
locally to "file"
```

```
    public void getFile(FileId fileId, File file) throws
IOException,ServerException {
      out.writeString("getFile");
      fileId.write(out);
      out.flush();
      receiveOk();
      in.readFile(file);
    }

    // Deletes a file from the server.  If you disconnect from the
server, the deletion will
    // be done automatically
    public void deleteFile(FileId fileId) throws
IOException,ServerException {
      out.writeString("deleteFile");
      fileId.write(out);
      out.flush();
      receiveOk();
    }

    // Used by cliends to add a job to the server.
    public JobId addJob(String command, FileId[] inputs, int
outputCount) throws IOException,ServerException {
      out.writeString("addJob");
      out.writeString(command);
      out.writeInt(inputs.length);
      for(int i=0; i<inputs.length; i++) {
          inputs[i].write(out);
      }
      out.writeInt(outputCount);
      out.flush();
      receiveOk();
      return JobId.read(in);
    }

    // Used by clients to monitor the status of a job.  NOTE: Instead
of making multiple calls to
    // this function to monitor several jobs, it is more efficient to
call the array version
    // given below.
    public Job getJob(JobId jobId) throws IOException,ServerException {
      JobId[] temp = new JobId[1];
      temp[0] = jobId;
      Job[] tempJob = getJob(temp);
      return tempJob[0];
    }

    // Efficient way for clients to monitor the status of multiple
jobs.
    public Job[] getJob(JobId jobId[]) throws
IOException,ServerException {
      out.writeString("getJob");
      out.writeInt(jobId.length);
      for(int i=0; i<jobId.length; i++) {
          jobId[i].write(out);
```

```
      }
      out.flush();
      receiveOk();
      Job[] jobs = new Job[jobId.length];
      for(int i=0; i<jobs.length; i++) {
          jobs[i] = Job.read(in);
      }
      return jobs;
    }


    /// BEGIN -- FUNCTIONS THAT THE CLIENT CAN USE, BUT AREN'T
NECESSARY /////

    // Get the UserId of this connection
    public UserId getId() {
      return id;
    }

    // Returns information about a specific file.
    public FileInfo getFileInfo(FileId fileId) throws
IOException,ServerException {
      FileId[] id = {fileId};
      FileInfo info[] = getFileInfo(id);
      return info[0];
    }

    // Returns information about multiple files.
    public FileInfo[] getFileInfo(FileId fileId[]) throws
IOException,ServerException {
      out.writeString("getFileInfo");
      out.writeInt(fileId.length);
      for(int i=0; i<fileId.length; i++) {
          fileId[i].write(out);
      }
      out.flush();
      receiveOk();
      FileInfo[] fileInfo = new FileInfo[fileId.length];
      for(int i=0; i<fileInfo.length; i++) {
          fileInfo[i] = FileInfo.read(in);
      }
      return fileInfo;
    }

    // Returns information about all of the users of the system (Used
by Server Monitor)
    public UserInfo[] getAllUserInfo() throws
IOException,ServerException {
      out.writeString("getAllUserInfo");
      out.flush();
      receiveOk();
      int count = in.readInt();
      UserInfo[] userInfo = new UserInfo[count];
      for(int i=0; i<count; i++) {
          userInfo[i] = UserInfo.read(in);
```

```
      }
      return userInfo;
    }

    // Returns information about all of the files in the system (Used
by Server Monitor)
    public FileInfo[] getAllFileInfo() throws
IOException,ServerException {
      out.writeString("getAllFileInfo");
      out.flush();
      receiveOk();
      int count = in.readInt();
      FileInfo[] fileInfo = new FileInfo[count];
      for(int i=0; i<count; i++) {
          fileInfo[i] = FileInfo.read(in);
      }
      return fileInfo;
    }

    // Returns information about all of the jobs in the system (Used by
Server Monitor)
    public Job[] getAllJobs() throws IOException,ServerException {
      out.writeString("getAllJobs");
      out.flush();
      receiveOk();
      int count = in.readInt();
      Job[] jobs = new Job[count];
      for(int i=0; i<count; i++) {
          jobs[i] = Job.read(in);
      }
      return jobs;
    }

    /// BEGIN - FUNCTIONS THAT SHOULD NEVER BE CALLED BY A CLIENT ////

    // Updates the status of a job (Called by the worker to report
progress)
    public void updateJob(Job job) throws IOException,ServerException {
      out.writeString("updateJob");
      job.write(out);
      out.flush();
      receiveOk();
    }

    // Gets a pending job from the server (Called by the worker when it
has nothing to do)
    public Job getPendingJob(CommandSignature commandSignatures[])
throws IOException,ServerException {
      out.writeString("getPendingJob");
      out.writeInt(commandSignatures.length);
      for(int i=0; i<commandSignatures.length; i++) {
          commandSignatures[i].write(out);
      }
      out.flush();
      receiveOk();
```

```
      return Job.read(in);
    }

    // Updates a non-existant file with new information (Called by the
worker to report outputs)
    public void updateFile(FileId fileId, File file) throws
IOException,ServerException {
      out.writeString("updateFile");
      fileId.write(out);
      out.writeFile(file);
      out.flush();
      receiveOk();
    }

    private void receiveOk() throws ServerException,IOException {
      String msg = in.readString();
      if (!msg.equals("Ok")) {
          throw new ServerException(msg);
      }
    }
}
```

VisualMonitor/VisualMonitor.java

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;
import java.io.*;

class VisualMonitor extends JFrame {
    ServerConnection connection = null;
    UserPanel userPanel;
    JobPanel jobPanel;
    FilePanel filePanel;
    java.util.Timer timer = new java.util.Timer();
    TimerTask runTask;

    public VisualMonitor(String name, int host) {
      super("BioGrid Visual Monitor v1.0 - " + name + ":" + host);
      try {
          connection = new ServerConnection(name,host,"VisualMonitor");
      } catch (IOException e) {
          if (e.getMessage() == null) {
            System.out.println("Error connecting to the server.");
          } else {
            System.out.println("Error connecting to the server: " +
e.getMessage());
          }
          System.exit(1);;
      }
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.BOTTOM);
      userPanel = new UserPanel(connection);
      jobPanel = new JobPanel(connection);
```

```
      filePanel = new FilePanel(connection);
      tabbedPane.addTab("Users",userPanel);
      tabbedPane.addTab("Jobs",jobPanel);
      tabbedPane.addTab("Files",filePanel);
      getContentPane().add(tabbedPane);
      setSize(600,400);
      show();

      runTask = new TimerTask() { public void run() { updateAll(); } };
      timer.schedule(runTask,0,300);
    }

    public void updateAll() {
      userPanel.updateAll();
      jobPanel.updateAll();
      filePanel.updateAll();
    }

    public static void main(String args[]) {
      if (args.length != 2) {
          System.out.println("usage: java VisualMonitor <host>
<port>");
          return;
      }

      System.out.println();
      System.out.println("BioGrid Visual Monitor v1.0");
      System.out.println("written by James M. McCollum
(jmccoll2@utk.edu)");
      System.out.println("University of Tennessee - Knoxville");
      System.out.println("http://biocomp.ece.utk.edu");
      System.out.println();

      try {
          VisualMonitor monitor = new
VisualMonitor(args[0],Integer.parseInt(args[1]));
      } catch (NumberFormatException e) {
          System.out.println("usage: java VisualMonitor <host>
<port>");
      }
    }
}
```

VisualMonitor/JobPanel.java

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;
import java.io.*;

class JobPanel extends JPanel {
    JobTableModel jobTableModel;
    JTable jobTable;
```

```
    public JobPanel(ServerConnection connection) {
      this.setLayout(new GridBagLayout());

      GridBagConstraints gbc = new GridBagConstraints();
      gbc.insets = new Insets(5,5,5,5);
      gbc.fill = GridBagConstraints.BOTH;
      gbc.weightx = 1;
      gbc.weighty = 1;

      jobTableModel = new JobTableModel(connection);
      jobTable = new JTable(jobTableModel);
      jobTable.setDefaultRenderer(Double.class,new
ProgressBarRenderer());
      jobTable.setRowSelectionAllowed(false);
      jobTable.setColumnSelectionAllowed(false);

      int tableWidths[] = {70,100,100,100,70,70,70,100,100};
      for(int i=0; i<tableWidths.length; i++) {

jobTable.getColumnModel().getColumn(i).setPreferredWidth(tableWidths[i]
);
      }
      this.add(new JScrollPane(jobTable),gbc);
    }

    public void updateAll() {
      jobTableModel.update();
      jobTableModel.fireTableDataChanged();
    }
}
```

VisualMonitor/ProgressBarRenderer.java

```
import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;

class ProgressBarRenderer extends JProgressBar implements
TableCellRenderer {
    public ProgressBarRenderer() {
      super(0,100);
    }

    public Component getTableCellRendererComponent(JTable table,
                                       Object value,
                                       boolean isSelected,
                                       boolean hasFocus,
                                       int row,
                                       int col) {
      int intValue = ((Double)value).intValue();
      setValue(intValue);
      if (intValue < 0) {
          setStringPainted(false);
      } else {
          setStringPainted(true);
```

```
      }
      return this;
    }
}
```

VisualMonitor/FilePanel.java

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;
import java.io.*;

class FilePanel extends JPanel {
    FileTableModel fileTableModel;
    JTable fileTable;

    public FilePanel(ServerConnection connection) {
      this.setLayout(new GridBagLayout());

      GridBagConstraints gbc = new GridBagConstraints();
      gbc.insets = new Insets(5,5,5,5);
      gbc.fill = GridBagConstraints.BOTH;
      gbc.weightx = 1;
      gbc.weighty = 1;

      fileTableModel = new FileTableModel(connection);
      fileTable = new JTable(fileTableModel);
      fileTable.setDefaultRenderer(Double.class,new
ProgressBarRenderer());
      fileTable.setRowSelectionAllowed(false);
      fileTable.setColumnSelectionAllowed(false);

      int tableWidths[] = {70,70,70,70,70,250};
      for(int i=0; i<tableWidths.length; i++) {

fileTable.getColumnModel().getColumn(i).setPreferredWidth(tableWidths[i
]);
      }
      this.add(new JScrollPane(fileTable),gbc);
    }

    public void updateAll() {
      fileTableModel.update();
      fileTableModel.fireTableDataChanged();
    }
}
```

VisualMonitor/FileTableModel.java

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;
import java.io.*;
```

```
public class FileTableModel extends AbstractTableModel {
    String colnames[] = {"ID","Owner","Creator","Exists","Depends
On","Server Filename"};
    FileInfo info[] = new FileInfo[0];
    ServerConnection connection;

    public FileTableModel(ServerConnection connection) {
      this.connection = connection;
      update();
    }

    public void update() {
      try {
          info = connection.getAllFileInfo();
      } catch (IOException e) {
          if (e.getMessage() == null) {
            System.out.println("Connection to the server lost.");
          } else {
            System.out.println("Connection to the server lost: " +
e.getMessage());
          }
          System.exit(1);
      } catch (ServerException e) {
          if (e.getMessage() == null) {
            System.out.println("Connection to the server lost.");
          } else {
            System.out.println("Connection to the server lost: " +
e.getMessage());
          }
          System.exit(1);
      }
    }

    public String getColumnName(int col) {
      return colnames[col];
    }

    public int getColumnCount() {
      return colnames.length;
    }

    public int getRowCount() {
      return info.length;
    }

    public boolean isCellEditable(int row, int col) {
      return false;
    }

    public Object getValueAt(int row, int col) {
      try {
          if (col == 0) {
            return info[row].getId();
          } else if (col == 1) {
```

```
            return info[row].getOwner();
          } else if (col == 2) {
            if (info[row].getCreator() == null) {
                return "";
            } else {
                return info[row].getCreator();
            }
          } else if (col == 3) {
            return new Boolean(info[row].getExists());
          } else if (col == 4) {
            if (info[row].getJobDependency() == null) {
                return "";
            } else {
                return info[row].getJobDependency();
            }
          } else if (col == 5) {
            return info[row].getServerFilePath();
          } else {
            return null;
          }
      } catch (ArrayIndexOutOfBoundsException e) {
          return "";
      }
    }
}
```

VisualMonitor/JobTableModel.java

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;
import java.io.*;

public class JobTableModel extends AbstractTableModel {
    String colnames[] =
{"ID","Command","Inputs","Outputs","Owner","Worker","Status","Error","P
rogress"};
    Job info[] = new Job[0];
    ServerConnection connection;

    public JobTableModel(ServerConnection connection) {
      this.connection = connection;
      update();
    }

    public void update() {
      try {
          info = connection.getAllJobs();
      } catch (IOException e) {
          if (e.getMessage() == null) {
            System.out.println("Connection to the server lost.");
          } else {
            System.out.println("Connection to the server lost: " +
e.getMessage());
```

```
            }
            System.exit(1);
        } catch (ServerException e) {
            if (e.getMessage() == null) {
              System.out.println("Connection to the server lost.");
            } else {
              System.out.println("Connection to the server lost: " +
e.getMessage());
            }
            System.exit(1);
        }
    }

    public Class getColumnClass(int col) {
      if (col == 8) return Double.class;
      return String.class;
    }

    public String getColumnName(int col) {
      return colnames[col];
    }

    public int getColumnCount() {
      return colnames.length;
    }

    public int getRowCount() {
      return info.length;
    }

    public boolean isCellEditable(int row, int col) {
      return false;
    }

    public Object getValueAt(int row, int col) {
      try {
          if (col == 0) {
            return info[row].getId().toString();
          } else if (col == 1) {
            return info[row].getCommand();
          } else if (col == 2) {
            FileId inputs[] = info[row].getInputs();
            String text = "";
            for(int i=0; i<inputs.length; i++) {
                if (i != 0) text += ",";
                text += inputs[i];
            }
            return text;
          } else if (col == 3) {
            FileId outputs[] = info[row].getOutputs();
            String text = "";
            for(int i=0; i<outputs.length; i++) {
                if (i != 0) text += ",";
                text += outputs[i];
            }
```

```
            return text;
          } else if (col == 4) {
            return info[row].getOwner().toString();
          } else if (col == 5) {
            if (info[row].getWorker() == null) {
                return "";
            } else {
                return info[row].getWorker().toString();
            }
          } else if (col == 6) {
            return info[row].getStatus().toString();
          } else if (col == 7) {
            return info[row].getErrorMessage();
          } else if (col == 8) {
            if ((info[row].getStatus() == JobStatus.COMPLETE) ||
(info[row].getStatus() == JobStatus.RUNNING)) {
                return new Double(info[row].getProgress());
            } else {
                return new Double(-1.0);
            }
          } else {
            return null;
          }
      } catch (ArrayIndexOutOfBoundsException e) {
          if (col == 8) {
            return new Double(0.0);
          } else {
            return "";
          }
      }
    }
}
```

VisualMonitor/UserPanel.java

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;
import java.io.*;

class UserPanel extends JPanel {
    UserTableModel userTableModel;
    JTable userTable;

    public UserPanel(ServerConnection connection) {
      this.setLayout(new GridBagLayout());

      GridBagConstraints gbc = new GridBagConstraints();
      gbc.insets = new Insets(5,5,5,5);
      gbc.fill = GridBagConstraints.BOTH;
      gbc.weightx = 1;
      gbc.weighty = 1;

      userTableModel = new UserTableModel(connection);
```

```
      userTable = new JTable(userTableModel);
      userTable.setDefaultRenderer(Double.class,new
ProgressBarRenderer());
      userTable.setRowSelectionAllowed(false);
      userTable.setColumnSelectionAllowed(false);

      int tableWidths[] = {70,100,100,70,100,70,200};
      for(int i=0; i<tableWidths.length; i++) {

userTable.getColumnModel().getColumn(i).setPreferredWidth(tableWidths[i
]);
      }
      this.add(new JScrollPane(userTable),gbc);
    }

    public void updateAll() {
      userTableModel.update();
      userTableModel.fireTableDataChanged();
    }
}
```

VisualMonitor/UserTableModel.java

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;
import java.io.*;

public class UserTableModel extends AbstractTableModel {
    String colnames[] =
{"ID","Host","Address","Type","Action","Job","Progress"};
    UserInfo info[] = new UserInfo[0];
    ServerConnection connection;

    public UserTableModel(ServerConnection connection) {
      this.connection = connection;
      update();
    }

    public void update() {
      try {
          info = connection.getAllUserInfo();
      } catch (IOException e) {
          if (e.getMessage() == null) {
            System.out.println("Connection to the server lost.");
          } else {
            System.out.println("Connection to the server lost: " +
e.getMessage());
          }
          System.exit(1);
      } catch (ServerException e) {
          if (e.getMessage() == null) {
            System.out.println("Connection to the server lost.");
          } else {
```

```
            System.out.println("Connection to the server lost: " +
e.getMessage());
          }
          System.exit(1);
      }
    }

    public Class getColumnClass(int col) {
      if (col == 6) return Double.class;
      return String.class;
    }

    public String getColumnName(int col) {
      return colnames[col];
    }

    public int getColumnCount() {
      return colnames.length;
    }

    public int getRowCount() {
      return info.length;
    }

    public boolean isCellEditable(int row, int col) {
      return false;
    }

    public Object getValueAt(int row, int col) {
      try {
          if (col == 0) {
            return info[row].getId().toString();
          } else if (col == 1) {
            return info[row].getHostName();
          } else if (col == 2) {
            return info[row].getHostAddress();
          } else if (col == 3) {
            return info[row].getType();
          } else if (col == 4) {
            return info[row].getAction();
          } else if (col == 5) {
            if (info[row].getJobId() == null) {
                return "";
            } else {
                return info[row].getJobId().toString();
            }
          } else if (col == 6) {
            if (info[row].getJobId() == null) {
                return new Double(-1.0);
            } else {
                return new Double(info[row].getProgress());
            }
          } else {
            return null;
          }
```

```
      } catch (ArrayIndexOutOfBoundsException e) {
          if (col == 6) {
            return new Double(0.0);
          } else {
            return "";
          }
      }
    }
}
```

Monitor/Monitor.java

```
import java.io.*;

class Monitor {
    public static void main(String args[]) {
      if (args.length != 2) {
          System.out.println("usage: java Monitor <host> <port>");
          return;
      }

      System.out.println();
      System.out.println("BioGrid Monitor v1.0");
      System.out.println("written by James M. McCollum
(jmccoll2@utk.edu)");
      System.out.println("University of Tennessee - Knoxville");
      System.out.println("http://biocomp.ece.utk.edu");
      System.out.println();

      try {
          ServerConnection connection = new
ServerConnection(args[0],Integer.parseInt(args[1]),"Monitor");
          System.out.println("Connected to server as " +
connection.getId());

          while (true) {
            FileInfo[] allFiles = connection.getAllFileInfo();
            UserInfo[] allUsers = connection.getAllUserInfo();
            Job[] allJobs = connection.getAllJobs();
            for(int i=0; i<allFiles.length; i++) {
                System.out.println(allFiles[i]);
            }
            System.out.println();
            for(int i=0; i<allUsers.length; i++) {
                System.out.println(allUsers[i]);
            }
            System.out.println();
            for(int i=0; i<allJobs.length; i++) {
                System.out.println(allJobs[i]);
            }
            System.out.println();
            System.out.println();
            try { Thread.sleep(1000); } catch (Exception e) { }
          }
      } catch (IOException e) {
```

```
            System.out.println("Connection to server lost.");
        } catch (ServerException e) {
            System.out.println(e.getMessage());
        } catch (NumberFormatException e) {
            System.out.println("usage: java Monitor <host> <port>");
            return;
        }
    }
}
```

ESSWorker/DataFile.java

```
import java.io.*;

class DataFile {
    public static void write(TimeSeriesData tsd, String filename)
throws IOException {
      DataOutputStream out = new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(filename)));
      out.writeInt(13579);
      out.writeInt(97531);
      int rows = tsd.data[0].length;
      int columns = tsd.header.length;
      out.writeInt(rows);
      out.writeInt(columns);
      for(int row=0; row < rows; row++) {
          for(int col=0; col<columns; col++) {
            out.writeDouble(tsd.data[col][row]);
          }
      }
      for(int i=0; i<tsd.header.length; i++) {
          String columnName = tsd.header[i];
          out.writeInt(columnName.length());
          out.writeChars(columnName);
      }
      out.close();
    }

    public static TimeSeriesData read(String filename) throws
IOException {
      DataInputStream in = new DataInputStream(new
BufferedInputStream(new FileInputStream(filename)));
      if (in.readInt() != 13579) throw new IOException("Invalid file
format.");
      if (in.readInt() != 97531) throw new IOException("Invalid file
format.");
      int rows = in.readInt();
      int columns = in.readInt();
      TimeSeriesData tsd = new TimeSeriesData();
      tsd.data = new double[columns][];
      for(int i=0; i<columns; i++) {
          tsd.data[i] = new double[rows];
      }
      for(int row=0; row<tsd.data[0].length; row++) {
          for(int col=0; col<tsd.data.length; col++) {
```

```
            tsd.data[col][row] = in.readDouble();
          }
        }
        tsd.header = new String[columns];
        for(int i=0; i<columns; i++) {
          tsd.header[i] = "";
          int length = in.readInt();
          for(int j=0; j<length; j++) {
            tsd.header[i] += in.readChar();
          }
        }
        return tsd;
      }

      public static void sample(String inputFilename, String
outputFilename, int rate) throws IOException {
        DataInputStream in = new DataInputStream(new
BufferedInputStream(new FileInputStream(inputFilename)));
        if (in.readInt() != 13579) throw new IOException("Invalid file
format.");
        if (in.readInt() != 97531) throw new IOException("Invalid file
format.");
        DataOutputStream out = new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(outputFilename)));

        int in_rows = in.readInt();
        int in_cols = in.readInt();

        int out_rows = in_rows / rate;

        out.writeInt(13579);
        out.writeInt(97531);
        out.writeInt(out_rows);
        out.writeInt(in_cols);

        for(int row=0; row<out_rows; row++) {
          for(int col=0; col<in_cols; col++) {
            out.writeDouble(in.readDouble());
          }
          in.skipBytes(8*in_cols*(rate-1));
        }
        in.skipBytes(8*in_cols*(in_rows % rate));

        int i;
        while((i = in.read()) != -1) {
          out.write(i);
        }
        in.close();
        out.close();
      }

      public static void tsd2tab(String inputFile, String outputFile)
throws IOException {
        TABFile.write(TSDFile.read(inputFile),outputFile);
      }
```

```
    public static void tsd2bsd(String inputFile, String outputFile)
throws IOException {
      write(TSDFile.read(inputFile),outputFile);
    }

    public static void bsd2tsd(String inputFile, String outputFile)
throws IOException {
      TSDFile.write(read(inputFile),outputFile);
    }

    public static void bsd2tab(String inputFile, String outputFile)
throws IOException {
      TABFile.write(read(inputFile),outputFile);
    }

    public static void tab2bsd(String inputFile, String outputFile)
throws IOException {
      write(TABFile.read(inputFile),outputFile);
    }

    public static void tab2tsd(String inputFile, String outputFile)
throws IOException {
      TSDFile.write(TABFile.read(inputFile),outputFile);
    }

    public static void merge(String inputFileA, String inputFileB,
                        String prefixA, String prefixB,
                        String outputFile, boolean zeroPad) throws
IOException {
      DataInputStream inA = new DataInputStream(new
BufferedInputStream(new FileInputStream(inputFileA)));
      DataInputStream inB = new DataInputStream(new
BufferedInputStream(new FileInputStream(inputFileB)));
      if (inA.readInt() != 13579) throw new IOException("Invalid file
format.");
      if (inA.readInt() != 97531) throw new IOException("Invalid file
format.");
      if (inB.readInt() != 13579) throw new IOException("Invalid file
format.");
      if (inB.readInt() != 97531) throw new IOException("Invalid file
format.");
      DataOutputStream out = new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(outputFile)));
      int inArows = inA.readInt();
      int inAcols = inA.readInt();
      int inBrows = inB.readInt();
      int inBcols = inB.readInt();
      int newrows = inArows;
      if (zeroPad) {
          if (inArows < inBrows) newrows = inBrows;
      } else {
          if (inArows > inBrows) newrows = inBrows;
      }
      int newcols = inAcols + inBcols;
```

```
      out.writeInt(13579);
      out.writeInt(97531);
      out.writeInt(newrows);
      out.writeInt(newcols);
      for(int row=0; row<newrows; row++) {
          for(int col=0; col<inAcols; col++) {
            if (row<inArows) {
                out.writeDouble(inA.readDouble());
            } else {
                out.writeDouble(0.0);
            }
          }
          for(int col=0; col<inBcols; col++) {
            if (row<inBrows) {
                out.writeDouble(inB.readDouble());
            } else {
                out.writeDouble(0.0);
            }
          }
      }

      if (!zeroPad) {
          if (newrows == inArows) {
            inB.skipBytes(8*inBcols*(inBrows-newrows));
          } else {
            inA.skipBytes(8*inAcols*(inArows-newrows));
          }
      }

      for(int i=0; i<inAcols; i++) {
          int strlen = inA.readInt();
          String header = "" + prefixA;
          for(int j=0; j<strlen; j++) {
            header += inA.readChar();
          }
          out.writeInt(header.length());
          out.writeChars(header);
      }
      for(int i=0; i<inBcols; i++) {
          int strlen = inB.readInt();
          String header = "" + prefixB;
          for(int j=0; j<strlen; j++) {
            header += inB.readChar();
          }
          out.writeInt(header.length());
          out.writeChars(header);
      }
      out.close();
      inA.close();
      inB.close();
    }
}
```

ESSWorker/ESSCommand.java

```
import java.io.*;

public class ESSCommand extends Command {
    public String getCommandName() {
      return "ess";
    }

    public int getInputCount() {
      return 2;
    }

    public int getOutputCount() {
      return 1;
    }

    public void run(Worker worker,
                File inputFiles[],
                File outputFiles[]) throws ServerException,IOException
{
      BioSpreadsheetData data = new BioSpreadsheetData();
      File tempFile;
      try {
          data.load(inputFiles[0].getAbsolutePath());
      } catch (IOException e) {
          throw new ServerException("Error reading sbml input file: " +
e.getMessage());
      }

      double printInterval;
      double endTime;
      int seed;
      try {
          DataInputStream in = new DataInputStream(new
FileInputStream(inputFiles[1]));
          printInterval = in.readDouble();
          endTime = in.readDouble();
          seed = in.readInt();
          in.close();
      } catch (IOException e) {
          throw new ServerException("Error reading simulation parameter
file: " + e.getMessage());
      }

      try {
          tempFile = File.createTempFile("essWorker.",".in");
          data.writeEssFile(tempFile.getAbsolutePath());
      } catch (IOException e) {
          throw new ServerException("Error writing ess file from sbml
file: " + e.getMessage());
      }

      DataOutputStream out;
      try {
          out = new DataOutputStream(new BufferedOutputStream(new
FileOutputStream(outputFiles[0])));
```

```
            out.writeInt(13579);
            out.writeInt(97531);
            out.writeInt((int)(endTime/printInterval + 1));
            out.writeInt(data.getOutputCount() + 1);
        } catch (IOException e) {
            throw new ServerException("Error creating output data file: "
+ e.getMessage());
        }

        worker.reportProgress(0.0);
        String cmdline = System.getProperty("ess.path");
        cmdline += " " + tempFile.getAbsolutePath() + " " + printInterval
+ " " + endTime + " " + seed;
        Process process;
        try {
            process = Runtime.getRuntime().exec(cmdline);
        } catch (IOException e) {
            throw new ServerException("Error starting simulator: " +
e.getMessage());
        }

        BufferedReader in = new BufferedReader(new
InputStreamReader(process.getInputStream()));
        double currentTime = 0.0;
        while (currentTime != endTime) {
            String s = "";
            try {
              s = in.readLine();
            } catch (IOException e) {
              process.destroy();
              throw new ServerException("Error reading data from the
simulator: " + e.getMessage());
            }
            String items[] = s.split(" ");
            for(int i=0; i<items.length; i++) {
              try {
                  out.writeDouble(Double.parseDouble(items[i]));
              } catch (IOException e) {
                  process.destroy();
                  throw new ServerException("Error writing data to the
output file: " + e.getMessage());
              }
            }
            currentTime = Double.parseDouble(items[0]);
            double temp = currentTime/endTime*100.0;
            if (temp != 100.0) {
              worker.reportProgress(temp);
            }
        }

        try {
            out.writeInt(("time").length());
            out.writeChars("time");
        } catch (IOException e) {
            process.destroy();
```

```
            throw new ServerException("Error writing data to the output
file: " + e.getMessage());
        }

        for(int i=0; i<data.getSpeciesCount(); i++) {
            if (data.getSpeciesOutput(i) == Boolean.TRUE) {
              try {
                  out.writeInt(data.getSpeciesName(i).length());
                  out.writeChars(data.getSpeciesName(i));
              } catch (IOException e) {
                  process.destroy();
                  throw new ServerException("Error writing data to the
output file: " + e.getMessage());
              }
            }
        }

        try { process.waitFor(); } catch (InterruptedException e) { }
        try { in.close(); } catch (IOException e) { }
        try { out.close(); } catch (IOException e) { }

        tempFile.delete();
        worker.reportProgress(100.0);
        out.close();
    }
}
```

ESSWorker/TSDFile.java

```
import java.io.*;
import java.util.*;

class TSDFile {
    public static TimeSeriesData read(String filename)
      throws IOException {

      TimeSeriesData tsd = new TimeSeriesData();

      int columnCount = getColumnCount(filename);
      int rowCount = getRowCount(filename);
      if ((columnCount < 0) || (rowCount < 0)) {
          throw new IOException("Bad time series data file format.");
      }

      tsd.header = new String[columnCount];
      tsd.data = new double[columnCount][];
      for(int i=0; i<columnCount; i++) {
          tsd.data[i] = new double[rowCount];
      }

      BufferedReader in = new BufferedReader(new FileReader(filename));

      in.read();
      if (columnCount > 1) {
          in.read();
```

```
      }

      for(int i=0; i<columnCount; i++) {
          tsd.header[i] = getNextString(in);
      }
      for(int i=0; i<rowCount; i++) {
          for(int j=0; j<columnCount; j++) {
            tsd.data[j][i] = getNextDouble(in);
          }
      }

      in.close();
      return tsd;
    }

    public static void write(TimeSeriesData tsd, String filename)
      throws IOException {
      PrintStream out = new PrintStream(new
FileOutputStream(filename));
      if (tsd.header.length > 1) {
          out.print("((");
          for(int i=0; i<tsd.header.length; i++) {
            if (i != 0) out.print(",");
            out.print("\"");
            out.print(tsd.header[i]);
            out.print("\"");
          }
          for(int i=0; i<tsd.data[0].length; i++) {
            out.print("),(");
            for(int j=0; j<tsd.data.length; j++) {
                if (j != 0) out.print(",");
                out.print(tsd.data[j][i]);
            }
          }
          out.print("))");
      } else {
          out.print("(\"");
          out.print(tsd.header[0]);
          out.print("\"");
          for(int i=0; i<tsd.data[0].length; i++) {
            out.print(",");
            out.print(tsd.data[0][i]);
          }
          out.print(")");
      }
      out.close();
    }

    private static int getColumnCount(String filename) throws
IOException {
      BufferedReader in = new BufferedReader(new FileReader(filename));
      int c = in.read();
      c = in.read();
      if (c == '(') {
          int columnCount = 1;
```

```
            while ((c != -1) && (c != ')')) {
              if (c == ',') columnCount++;
              c = in.read();
            }
            return columnCount;
        } else {
            return 1;
        }
    }

    private static int getRowCount(String filename) throws IOException
{
        BufferedReader in = new BufferedReader(new FileReader(filename));
        int c = in.read();
        c = in.read();
        if (c == '(') {
            int rowCount = -2;
            while (c != -1) {
              if (c == ')') {
                  rowCount++;
              }
              c = in.read();
            }
            return rowCount;
        } else {
            int rowCount = 0;
            while (c != -1) {
              if (c == ',') {
                  rowCount++;
              }
              c = in.read();
            }
            return rowCount;
        }
    }

    private static double getNextDouble(BufferedReader in) throws
IOException {
        String value = "";
        int c = in.read();
        if (c == -1) {
            throw new IOException("End of file.");
        }
        while ((c != ')') && (c != ',') && (c != -1)) {
            value += (char)c;
            c = in.read();
        }
        try {
            if (c == ')') {
              in.read();
              in.read();
            }
            return Double.parseDouble(value);
        } catch (NumberFormatException e) {
            throw new IOException("Invalid number: " + value);
```

```
      }
    }

    public static String getNextString(BufferedReader in) throws
IOException {
      String value = "";
      int c = in.read();
      if (c != '"') throw new IOException("String expected.");
      c = in.read();
      while((c != -1) && (c != '"')) {
          value += (char)c;
          c = in.read();
      }
      c = in.read();
      if (c == ')') {
          in.read();
          in.read();
      }
      return value;
    }
}
```

ESSWorker/ESSWorker.java

```
import java.io.*;

class ESSWorker {
    public static void printUsage() {
      System.out.println("usage: java ESSWorker <host> <port>");
      System.exit(1);
    }

    public static void main(String args[]) {
      if (args.length != 2) {
          printUsage();
      }
      String host = args[0];
      int port = 0;
      try {
          port = Integer.parseInt(args[1]);
      } catch (NumberFormatException e) {
          printUsage();
      }

      System.out.println();
      System.out.println("BioGrid ESS Worker v1.0");
      System.out.println("written by James M. McCollum
(jmccoll2@utk.edu)");
      System.out.println("University of Tennessee - Knoxville");
      System.out.println("http://biocomp.ece.utk.edu");
      System.out.println();

      if (System.getProperty("ess.path") == null) {
          System.out.println("ess.path variable not set.");
          System.exit(1);
```

```
      }

      Worker worker = new Worker(host,port,"ESSWorker");
      worker.addCommand(new ESSCommand());
      worker.run();
    }
}
```

ESSWorker/TimeSeriesData.java

```
class TimeSeriesData {
    String header[];
    double data[][];

    public void print() {
      for(int i=0; i<header.length; i++) {
          if (i != 0) System.out.print("\t");
          System.out.print(header[i]);
      }
      System.out.println();

      for(int i=0; i<data[0].length; i++) {
          for(int j=0; j<data.length; j++) {
            if (j != 0) System.out.print("\t");
            System.out.print(data[j][i]);
          }
          System.out.println();
      }
    }
}
```

ESSWorker/TABFile.java

```
import java.io.*;

class TABFile {
    public static TimeSeriesData read(String filename) throws
IOException {
      int columnCount = getColumnCount(filename);
      int rowCount = getRowCount(filename);

      String header[];
      double data[][] = new double[columnCount][];
      for(int i=0; i<columnCount; i++) {
          data[i] = new double[rowCount];
      }

      BufferedReader in = new BufferedReader(new FileReader(filename));
      header = in.readLine().split("\\t");

      for(int i=0; i<rowCount; i++) {
          String dataStr[] = in.readLine().split("\\t");

          if (dataStr.length != columnCount) {
            throw new IOException("Bad file format.");
```

```
            }

            for(int j=0; j<columnCount; j++) {
              try {
                  data[j][i] = Double.parseDouble(dataStr[j]);
              } catch(NumberFormatException e) {
                  throw new IOException("Bad file format.");
              }
            }
        }

        TimeSeriesData tsd = new TimeSeriesData();
        tsd.data = data;
        tsd.header = header;
        return tsd;
    }

    public static void write(TimeSeriesData data,String filename)
throws IOException {
        PrintStream out = new PrintStream(new
FileOutputStream(filename));
        for(int i=0; i<data.header.length; i++) {
            if (i != 0) out.print("\t");
            out.print(data.header[i]);
        }
        out.println();

        for(int i=0; i<data.data[0].length; i++) {
            for(int j=0; j<data.data.length; j++) {
              if (j != 0) out.print("\t");
              out.print(data.data[j][i]);
            }
            out.println();
        }
        out.close();
    }

    private static int getColumnCount(String filename) throws
IOException {
        BufferedReader in = new BufferedReader(new FileReader(filename));
        int c = in.read();
        int columnCount = 1;
        while ((c != '\n') && (c != -1)) {
            if (c == '\t') {
              columnCount++;
            }
            c = in.read();
        }
        in.close();
        return columnCount;
    }

    private static int getRowCount(String filename) throws IOException
{
        BufferedReader in = new BufferedReader(new FileReader(filename));
```

```
      int c = in.read();
      int rowCount = -1;
      while (c != -1) {
          if (c == '\n') {
            rowCount++;
          }
          c = in.read();
      }
      in.close();
      return rowCount;
    }
}
```

ESSWorker/Reaction.java

```
import java.util.*;

/**
 * Class used to represent a chemical reaction.
 */
public class Reaction {
    /**
     * Creates a new Reaction
     */
    public Reaction() {

    }

    /**
     * Adds a reactant to the reaction.
     * @param coefficient
     * the reactant coefficient (must be greater than zero).
     * @param species
     * the name of the species (see ReactionParser for desciption of
valid species names).
     * @exception ReactionException
     * thrown if the coefficient or species in invalid.
     * @see ReactionParser
     */
    public void addReactant(int coefficient, String species) throws
ReactionException {
      if
(!ReactionParser.getReactionParser().isSpeciesNameValid(species)) {
          throw new ReactionException("Invalid species name");
      }
      if (coefficient < 1) {
          throw new ReactionException("Reactant coefficient is less
than 1");
      }
      reactants.add(new SpeciesUpdate(coefficient,species));
    }

    /**
     * Adds a product to the reaction.
     * @param coefficient
```

```
     * the product coefficient (must be greater than zero).
     * @param species
     * the name of the species (see ReactionParser for desciption of
valid species names).
     * @exception ReactionException
     * thrown if the coefficient or species in invalid.
     * @see ReactionParser
     */
    public void addProduct(int coefficient, String species) throws
ReactionException {
      if
(!ReactionParser.getReactionParser().isSpeciesNameValid(species)) {
          throw new ReactionException("Invalid species name");
      }
      if (coefficient < 1) {
          throw new ReactionException("Reactant coefficient is less
than 1");
      }
      products.add(new SpeciesUpdate(coefficient,species));
    }

    /**
     * Retrieves the number of reactants.
     * @return
     * the number of reactants.
     */
    public int getReactantCount() {
      return reactants.size();
    }

    /**
     * Retrieves the number of products.
     * @return
     * the number of products.
     */
    public int getProductCount() {
      return products.size();
    }

    /**
     * Retrieves the coefficient of a specific reactant.
     * @param index
     * the index of the reactant.
     * @return
     * the coefficient of the reactant.
     */
    public int getReactantCoefficient(int index) {
      return ((SpeciesUpdate)(reactants.get(index))).coefficient;
    }

    /**
     * Retrieves the species name of a specific reactant.
     * @param index
     * the index of the reactant.
     * @return
```

```
 * the species name of the reactant.
 */
public String getReactantSpecies(int index) {
  return ((SpeciesUpdate)(reactants.get(index))).species;
}

/**
 * Retrieves the coefficient of a specific product.
 * @param index
 * the index of the product.
 * @return
 * the coefficient of the product.
 */
public int getProductCoefficient(int index) {
  return ((SpeciesUpdate)(products.get(index))).coefficient;
}

/**
 * Retrieves the species name of a specific product.
 * @param index
 * the index of the product.
 * @return
 * the species name of the product.
 */
public String getProductSpecies(int index) {
  return ((SpeciesUpdate)(products.get(index))).species;
}

/**
 * Retrieves a string representation for this reaction.
 * @return
 * a string representing the reaction.
 */
public String toString() {
  String returnval = "";
  for(int i=0; i<getReactantCount(); i++) {
      if (i != 0) {
        returnval += "+ ";
      }
      if (getReactantCoefficient(i) != 1) {
        returnval += getReactantCoefficient(i) + " ";
      }
      returnval += getReactantSpecies(i) + " ";
  }
  returnval += "=> ";
  if (getProductCount() == 0) {
      returnval += "*";
  } else {
      for(int i=0; i<getProductCount(); i++) {
        if (i != 0) {
            returnval += "+ ";
        }
        if (getProductCoefficient(i) != 1) {
            returnval += getProductCoefficient(i) + " ";
        }
```

```
            returnval += getProductSpecies(i) + " ";
          }
      }
      return returnval;
    }

    private class SpeciesUpdate {
      public String species;
      public int coefficient;

      public SpeciesUpdate(int coefficient, String species) {
          this.species = species;
          this.coefficient = coefficient;
      }
    }

    private Vector reactants = new Vector();
    private Vector products = new Vector();
}
```

ESSWorker/BioSpreadsheetData.java

```
import java.util.Vector;
import java.io.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Document;
import org.w3c.dom.DOMException;

class BioSpreadsheetData {
    Vector speciesNames = new Vector();
    Vector speciesPopulations = new Vector();
    Vector speciesComments = new Vector();
    Vector speciesOutputs = new Vector();
    Vector reactions = new Vector();
    Vector reactionRates = new Vector();
    Vector reactionComments = new Vector();
    String name;
    String title;
    String author;
    String location;
    String description;

    public void addSpecies(String name, String population, Boolean
output, String comment) {
        speciesNames.add(name);
      speciesPopulations.add(population);
      speciesOutputs.add(output);
```

```
      speciesComments.add(comment);
    }

    public void addReaction(String reaction, String rate, String
comment) {
      reactions.add(reaction);
      reactionRates.add(rate);
      reactionComments.add(comment);
    }

    public void setTitle(String title) {
      this.title = title;
    }

    public void setName(String name) {
      this.name = name;
    }

    public void setAuthor(String author) {
      this.author = author;
    }

    public void setLocation(String location) {
      this.location = location;
    }

    public void setDescription(String description) {
      this.description = description;
    }

    public int getSpeciesCount() {
      return speciesNames.size();
    }

    public int getReactionCount() {
      return reactions.size();
    }

    public String getSpeciesName(int index) {
      return (String)speciesNames.get(index);
    }

    public String getSpeciesPopulation(int index) {
      return (String)speciesPopulations.get(index);
    }

    public Boolean getSpeciesOutput(int index) {
      return (Boolean)speciesOutputs.get(index);
    }

    public String getSpeciesComment(int index) {
      return (String)speciesComments.get(index);
    }

    public String getReaction(int index) {
```

```
      return (String)reactions.get(index);
    }

    public String getReactionRate(int index) {
      return (String)reactionRates.get(index);
    }

    public String getReactionComment(int index) {
      return (String)reactionComments.get(index);
    }

    public String getTitle() {
      return title;
    }

    public String getName() {
      return name;
    }

    public String getAuthor() {
      return author;
    }

    public String getLocation() {
      return location;
    }

    public String getDescription() {
      return description;
    }

    public Vector check() {
      Vector errors = new Vector();
      ReactionParser reactionParser =
ReactionParser.getReactionParser();

      if (reactions.size() == 0) {
          errors.add("At least one reaction must exist.");
      }

      for(int i=0; i<speciesNames.size(); i++) {
          if
(!reactionParser.isSpeciesNameValid((String)speciesNames.get(i))) {
            errors.add("Invalid species name \"" + speciesNames.get(i)
+ "\".");
          }
          try {
            int population =
Integer.parseInt((String)speciesPopulations.get(i));
            if (population < 0) {
                errors.add("Invalid species population \"" +
speciesPopulations.get(i) + "\".  Species populations must be non-
negative integers.");
            }
          } catch (NumberFormatException e) {
```

```
            errors.add("Invalid species population \"" +
speciesPopulations.get(i) + "\".  Species populations must be non-
negative integers.");
          }
      }

      String speciesNameArray[] = new String[speciesNames.size()];
      for(int i=0; i<speciesNames.size(); i++) {
          speciesNameArray[i] = (String)speciesNames.get(i);
      }

      for(int i=0; i<reactions.size(); i++) {
          try {

      reactionParser.parse((String)reactions.get(i),speciesNameArray);
          } catch (ReactionParserException e) {
            String errorMessage = "Invalid Reaction: ";
            errorMessage += e.getReaction() + "\n";
            for(int j=0; j<e.getLocation(); j++) {
                errorMessage += " ";
            }
            errorMessage += "                  ^ " + e.getMessage();
            errors.add(errorMessage);
          }
          try {
            double rate =
Double.parseDouble((String)reactionRates.get(i));
            if (rate < 0) {
                errors.add("Invalid reaction rate \"" +
reactionRates.get(i) + "\".  Reaction rates must be non-negative real
numbers.");
            }
          } catch (NumberFormatException e) {
            errors.add("Invalid reaction rate \"" +
reactionRates.get(i) + "\".  Reaction rates must be non-negative real
numbers.");
          }
      }

      return errors;
    }

    public String fix(Object in) {
      return in.toString().replace('\"','\'');
    }

    public void save(String filename) throws IOException {
      PrintStream out = new PrintStream(new
FileOutputStream(filename));
      out.println("<?xml version=\"1.0\" encoding=\"UTF-8\" ?>");
      out.println("<sbml xmlns=\"http://www.sbml.org/sbml/level2\"
level=\"2\" version=\"1\">");
      out.println("<model id=\"" + fix(name) + "\">");
      out.println("<annotation
xmlns:biospreadsheet=\"http://biocomp.ece.utk.edu/ns\">");
```

```
      out.println("<biospreadsheet:model author=\"" + fix(author) + "\"
title=\"" + fix(title) + "\" " +
                  "location=\"" + fix(location) + "\" description=\"" +
fix(description) + "\"/>");
      out.println("</annotation>");
      out.println("<listOfCompartments>");
      out.println("<compartment id=\"cell\"/>");
      out.println("</listOfCompartments>");
      out.println("<listOfSpecies>");
      for(int i=0; i<speciesNames.size(); i++) {
          out.println("<species " +
                  "id=\"" + fix(speciesNames.get(i)) + "\" " +
                  "initialAmount=\"" + fix(speciesPopulations.get(i)) +
"\" " +
                  "compartment=\"cell\">");
          out.println("<annotation
xmlns:biospreadsheet=\"http://biocomp.ece.utk.edu/ns\">");
          out.println("<biospreadsheet:species " +
                  "output=\"" + fix(speciesOutputs.get(i)) + "\" " +
                  "comment=\"" + fix(speciesComments.get(i)) + "\"/>");
          out.println("</annotation>");
          out.println("</species>");
      }
      out.println("</listOfSpecies>");
      out.println("<listOfReactions>");
      for(int i=0; i<reactions.size(); i++) {
          out.println("<reaction id=\"reaction" + i + "\"
reversible=\"false\">");
          out.println("<annotation
xmlns:biospreadsheet=\"http://biocomp.ece.utk.edu/ns\">");
          out.println("<biospreadsheet:reaction " +
                  "reaction=\"" + fix(reactions.get(i)) + "\" " +
                  "rate=\"" + fix(reactionRates.get(i)) + "\" " +
                  "comment=\"" + fix(reactionComments.get(i)) +
"\"/>");
          out.println("</annotation>");
          try {
            Reaction r =
ReactionParser.getReactionParser().parse((String)reactions.get(i));
            out.println("<listOfReactants>");
            for(int j=0; j<r.getReactantCount(); j++) {
                out.println("<speciesReference " +
                        "species=\"" + fix(r.getReactantSpecies(j)) +
"\" " +
                        "stochiometry=\"" + r.getReactantCoefficient(j)
+ "\"/>");
            }
            out.println("</listOfReactants>");
            out.println("<listOfProducts>");
            for(int j=0; j<r.getProductCount(); j++) {
                out.println("<speciesReference " +
                        "species=\"" + fix(r.getProductSpecies(j)) +
"\" " +
                        "stochiometry=\"" + r.getProductCoefficient(j)
+ "\"/>");
```

```
            }
            out.println("</listOfProducts>");
            out.println("<kineticLaw>");
            out.println("<math
xmlns=\"http://www.w3.org/1998/Math/MathML\">");
            for(int j=0; j<r.getReactantCount()-1; j++) {
                out.println("<math:apply><math:add/>");
            }
            for(int j=0; j<r.getReactantCount(); j++) {
                int coefficient = r.getReactantCoefficient(j);
                String name = r.getReactantSpecies(j);
                out.println("<math:apply><math:divide/>");
                out.println("<math:apply><math:divide/>");
                out.println("<math:apply><math:factorial/>");
                out.println("<math:ci>" + name + "</math:ci>");
                out.println("</math:apply>");
                out.println("<math:apply><math:factorial/>");
                out.println("<math:cn>" + coefficient + "</math:cn>");
                out.println("</math:apply>");
                out.println("</math:apply>");
                out.println("<math:apply><math:factorial/>");
                out.println("<math:apply><math:minus/>");
                out.println("<math:ci>" + name + "</math:ci>");
                out.println("<math:cn>" + coefficient + "</math:cn>");
                out.println("</math:apply>");
                out.println("</math:apply>");
                out.println("</math:apply>");
            }
            for(int j=0; j<r.getReactantCount()-1; j++) {
                out.println("</math:apply>");
            }
            out.println("</math>");
            out.println("</kineticLaw>");
          } catch (ReactionParserException e) {
            out.println("<listOfReactants>");
            out.println("</listOfReactants>");
            out.println("<listOfProducts>");
            out.println("</listOfProducts>");
            out.println("<kineticLaw>");
            out.println("</kineticLaw>");
          }
          out.println("</reaction>");
      }
      out.println("</listOfReactions>");
      out.println("</model>");
      out.println("</sbml>");
      out.close();
    }

    public void load(String filename) throws IOException {
      speciesNames = new Vector();
      speciesPopulations = new Vector();
      speciesComments = new Vector();
      speciesOutputs = new Vector();
      reactions = new Vector();
```

```
      reactionRates = new Vector();
      reactionComments = new Vector();
      name = "";
      title = "";
      author = "";
      location = "";
      description = "";

      DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
      Document document;
        try {
          DocumentBuilder builder = factory.newDocumentBuilder();
          document = builder.parse(new File(filename));
        } catch (SAXParseException spe) {
          throw new IOException("Unable to read " + filename + "
because there was an error parsing the file on line " +
spe.getLineNumber() + ": " + spe.getMessage());
        } catch (SAXException sxe) {
          throw new IOException("Unable to read " + filename + "
because there was an error parsing the file: " + sxe.getMessage());
        } catch (ParserConfigurationException pce) {
          throw new IOException("Unable to read " + filename + "
because the XML parser could not be configured: " + pce.getMessage());
        } catch (IOException ioe) {
          throw new IOException("Unable to read " + filename + "
because: " + ioe.getMessage());
        }

      Vector modelVector = findNodes(document,"model");
      if (modelVector.size() != 1) return;
      Node modelNode = (Node)modelVector.elementAt(0);
      if (modelNode.hasAttributes()) {
          NamedNodeMap attributes = modelNode.getAttributes();
          name = getItemFromAttributes(attributes,"id");
      }

      Vector bssModelVector =
findNodes(document,"biospreadsheet:model");
      if (bssModelVector.size() == 1) {
          Node bssModelNode = (Node)bssModelVector.elementAt(0);
          if (bssModelNode.hasAttributes()) {
            NamedNodeMap attributes = bssModelNode.getAttributes();
            title = getItemFromAttributes(attributes,"title");
            author = getItemFromAttributes(attributes,"author");
            location = getItemFromAttributes(attributes,"location");
            description =
getItemFromAttributes(attributes,"description");
          }
      }

      Vector listOfSpeciesNodes = findNodes(modelNode,"listOfSpecies");
      if (listOfSpeciesNodes.size() == 1) {
          Node listOfSpeciesNode = (Node)listOfSpeciesNodes.get(0);
          Vector speciesNodes = findNodes(listOfSpeciesNode,"species");
```

```
          for(int i=0; i<speciesNodes.size(); i++) {
            String name = "";
            String population = "";
            Boolean output = Boolean.TRUE;
            String comment = "";
            Node speciesNode = (Node)speciesNodes.elementAt(i);
            if (speciesNode.hasAttributes()) {
                NamedNodeMap attributes = speciesNode.getAttributes();
                name = getItemFromAttributes(attributes,"id");
                population =
getItemFromAttributes(attributes,"initialAmount");
            }
            Vector listOfBssSpeciesNodes =
findNodes(speciesNode,"biospreadsheet:species");
            if (listOfBssSpeciesNodes.size() == 1) {
                NamedNodeMap attributes =
((Node)listOfBssSpeciesNodes.get(0)).getAttributes();
                if
(getItemFromAttributes(attributes,"output").equals("false")) {
                  output = Boolean.FALSE;
                }
                comment = getItemFromAttributes(attributes,"comment");
            }
            addSpecies(name,population,output,comment);
          }
      }

      Vector listOfReactionsNodes =
findNodes(modelNode,"listOfReactions");
      if (listOfReactionsNodes.size() == 1) {
          Node listOfReactionsNode = (Node)listOfReactionsNodes.get(0);
          Vector reactionNodes =
findNodes(listOfReactionsNode,"reaction");
          for(int i=0; i<reactionNodes.size(); i++) {
            String reaction = "";
            String rate = "";
            String comment = "";
            Node reactionNode = (Node)reactionNodes.elementAt(i);
            Vector listOfBssReactionNodes =
findNodes(reactionNode,"biospreadsheet:reaction");
            if (listOfBssReactionNodes.size() == 1) {
                NamedNodeMap attributes =
((Node)listOfBssReactionNodes.get(0)).getAttributes();
                reaction =
getItemFromAttributes(attributes,"reaction");
                rate = getItemFromAttributes(attributes,"rate");
                comment = getItemFromAttributes(attributes,"comment");

            } else {
                reaction = getReactionString(reactionNode);
            }
            addReaction(reaction,rate,comment);
          }
      }
    }
```

```
    private String getItemFromAttributes(NamedNodeMap map, String name)
{
      if (map.getNamedItem(name) != null) {
          return map.getNamedItem(name).getNodeValue();
      }
      return "";
    }

    private String getReactionString(Node n) {
      String s = "";
      Vector listOfReactantsNodes = findNodes(n,"listOfReactants");
      int reactantCount = 0;
      for(int i=0; i<listOfReactantsNodes.size(); i++) {
          Node listOfReactantsNode =
(Node)listOfReactantsNodes.elementAt(i);
          Vector speciesReferences =
findNodes(listOfReactantsNode,"speciesReference");
          for(int j=0; j<speciesReferences.size(); j++) {
            Node speciesReference =
(Node)speciesReferences.elementAt(j);
            if (speciesReference.hasAttributes()) {
                NamedNodeMap attributes =
speciesReference.getAttributes();
                String speciesName =
getItemFromAttributes(attributes,"species");
                String coefficient =
getItemFromAttributes(attributes,"stoichiometry");
                if (!speciesName.equals("")) {
                  if (reactantCount != 0) {
                      s += "+ ";
                  }
                  if (!(coefficient.equals("") ||
coefficient.equals("1"))) {
                      s += coefficient + " ";
                  }
                  s += speciesName + " ";
                  reactantCount++;
                }
            }
          }
      }
      s += "-> ";
      Vector listOfProductsNodes = findNodes(n,"listOfProducts");
      int productCount = 0;
      for(int i=0; i<listOfProductsNodes.size(); i++) {
          Node listOfProductsNode =
(Node)listOfProductsNodes.elementAt(i);
          Vector speciesReferences =
findNodes(listOfProductsNode,"speciesReference");
          for(int j=0; j<speciesReferences.size(); j++) {
            Node speciesReference =
(Node)speciesReferences.elementAt(j);
            if (speciesReference.hasAttributes()) {
```

```
                NamedNodeMap attributes =
speciesReference.getAttributes();
                String speciesName =
getItemFromAttributes(attributes,"species");
                if (speciesName.equals("")) {
                  speciesName =
getItemFromAttributes(attributes,"specie");
                }
                String coefficient =
getItemFromAttributes(attributes,"stoichiometry");
                if (!speciesName.equals("")) {
                  if (productCount != 0) {
                      s += "+ ";
                  }
                  if (!(coefficient.equals("") ||
coefficient.equals("1"))) {
                      s += coefficient + " ";
                  }
                  s += speciesName + " ";
                  productCount++;
                }
              }
            }
      }
      if (productCount == 0) {
          s += "*";
      }
      return s.trim();
    }

    private Vector findNodes(Node n, String name) {
      Vector v = new Vector();
      if (n.getNodeName().equals(name)) {
          v.add(n);
      } else if (n.getNodeName().equals(name)) {
          v.add(n);
      } else if (n.hasChildNodes()) {
          NodeList nl = n.getChildNodes();
          for(int i=0; i<nl.getLength(); i++) {
            Vector v2 = findNodes(nl.item(i),name);
            for(int j=0; j<v2.size(); j++) {
                v.add(v2.elementAt(j));
            }
          }
      }
      return v;
    }

    public int getOutputCount() {
      int outputCount = 0;
      for(int i=0; i<getSpeciesCount(); i++) {
          if (getSpeciesOutput(i) == Boolean.TRUE) outputCount++;
      }
      return outputCount;
    }
```

```
    public void writeEssFile(String filename) throws IOException {
      PrintStream out = new PrintStream(new
FileOutputStream(filename));
      out.println(getSpeciesCount());
      for(int i=0; i<getSpeciesCount(); i++) {
          if (i != 0) out.print(" ");
          out.print(getSpeciesPopulation(i));
      }
      out.println();
      out.println(getReactionCount());
      for(int i=0; i<getReactionCount(); i++) {
          String reactionString = getReaction(i);
          try {
            Reaction r =
ReactionParser.getReactionParser().parse(reactionString);
            out.print("" + r.getReactantCount() + " ");
            for(int j=0; j<r.getReactantCount(); j++) {
                out.print("" + r.getReactantCoefficient(j) + " ");
                out.print("" + getSpeciesIndex(r.getReactantSpecies(j))
+ " ");
            }
            out.print("" + r.getProductCount() + " ");
            for(int j=0; j<r.getProductCount(); j++) {
                out.print("" + r.getProductCoefficient(j) + " ");
                out.print("" + getSpeciesIndex(r.getProductSpecies(j))
+ " ");
            }
            out.print("" + getReactionRate(i));
            out.println();
          } catch (ReactionParserException e) {
            System.out.println("This should never happen
(writeEssFile).");
            System.exit(1);
          }
      }
      out.println(getOutputCount());
      for(int i=0; i<getSpeciesCount(); i++) {
          if (getSpeciesOutput(i) == Boolean.TRUE) out.print("" + i + "
");
      }
      out.println();
      out.close();
    }

    public int getSpeciesIndex(String name) {
      for(int i=0; i<getSpeciesCount(); i++) {
          if (getSpeciesName(i).equals(name)) {
            return i;
          }
      }
      return -1;
    }
}
```

ESSWorker/ReactionException.java

```
/**
 * Exception class for reactions.
 */
public class ReactionException extends Exception {
    /**
     * Creates a new ReactionException
     * @param message
     * an error message
     */
    public ReactionException(String message) {
      super(message);
    }
}
```

ESSWorker/ReactionParser.java

```
import java.util.*;
import java.io.*;

/**
 * Class used to parse strings.
 * This is a singleton class so the constructor is private.
 * To retrieve an instance of this class, call the getReactionParser()
method.
 *
 * The string is parsed using the following reaction grammar:
 *
 * <pre>[reaction]     := [species-list] '=>' [species-list]</pre>
 * <pre>               := [species-list] '=>' '*'</pre>
 *
 * <pre>[species-list] := [species] '+' [species-list]</pre>
 * <pre>               := [species]</pre>
 *
 * <pre>[species]      := [unsigned-integer] [species-name]</pre>
 * <pre>               := [species-name]</pre>
 *
 * Species names may contain letters, numbers, or underscores, but the
first character must not be a number.
 *
 * <pre>Valid Reaction Examples:</pre>
 * <pre>   2 H + O => H2O</pre>
 * <pre>   H2S + 2 _Catalyst_ => *</pre>
 * @see Reaction
 */
public class ReactionParser {
    private static ReactionParser parser = null;

    /**
     * Retrieves a ReactionParser
     *
     * @return the ReactionParser
     */
    public static ReactionParser getReactionParser() {
```

```
      if (parser == null) {
          parser = new ReactionParser();
      }
      return parser;
    }

    /**
     * Converts a string into a reaction.
     *
     * @param reaction
     * the reaction to be parsed.
     * @return
     * the parsed reaction.
     * @exception ReactionParserException
     * if the reaction is not of the right format, this exception will
be thrown.
     */
    public Reaction parse(String reaction) throws
ReactionParserException {
      Reaction parsedReaction = new Reaction();
      Vector tokens = scan(reaction);
      int currentTokenIndex = 0;
      currentTokenIndex =
parseSpecies(tokens,currentTokenIndex,parsedReaction,true,reaction);
      ReactionToken token =
(ReactionToken)tokens.get(currentTokenIndex++);
      while (token.type == ReactionToken.PLUS) {
          currentTokenIndex =
parseSpecies(tokens,currentTokenIndex,parsedReaction,true,reaction);
          token = (ReactionToken)tokens.get(currentTokenIndex++);
      }
      if (token.type != ReactionToken.PRODUCES) {
          throw new ReactionParserException("'+' or '=>'
expected.",reaction,token.location);
      }
      token = (ReactionToken)tokens.get(currentTokenIndex++);
      if (token.type == ReactionToken.STAR) {
          token = (ReactionToken)tokens.get(currentTokenIndex++);
      } else if ((token.type == ReactionToken.INT) || (token.type ==
ReactionToken.SPECIES)) {
          currentTokenIndex--;
          currentTokenIndex =
parseSpecies(tokens,currentTokenIndex,parsedReaction,false,reaction);
          token = (ReactionToken)tokens.get(currentTokenIndex++);
          while (token.type == ReactionToken.PLUS) {
            currentTokenIndex =
parseSpecies(tokens,currentTokenIndex,parsedReaction,false,reaction);
            token = (ReactionToken)tokens.get(currentTokenIndex++);
          }
      } else {
          throw new ReactionParserException("'*' or species
expected.",reaction,token.location);
      }
      if (token.type != ReactionToken.END_OF_STRING) {
```

```
          throw new ReactionParserException("End of string
expected.",reaction,token.location);
      }
      return parsedReaction;
    }

    /**
     * Converts a string into a reaction.
     *
     * @param reaction
     * the reaction to be parsed.
     * @param validSpeciesNames
     * an array of valid species names
     * @return
     * the parsed reaction.
     * @exception ReactionParserException
     * if the reaction is not of the right format or contains an
invalid species name, this exception will be thrown.
     */
    public Reaction parse(String reaction, String[] validSpeciesNames)
throws ReactionParserException {
      Vector tokens = scan(reaction);
      for(int i=0; i<tokens.size(); i++) {
          ReactionToken token = (ReactionToken)tokens.get(i);
          if (token.type == ReactionToken.SPECIES) {
            boolean found = false;
            for(int j=0; j<validSpeciesNames.length; j++) {
                if (token.str.equals(validSpeciesNames[j])) {
                  found = true;
                  break;
                }
            }
            if (found == false) {
                throw new ReactionParserException("Undeclared species:
\"" + token.str + "\".",reaction,token.location);
            }
          }
      }
      return parse(reaction);
    }

    /**
     * Determines if a string is a valid species name..
     *
     * @param speciesName
     * the species name to check.
     * @return
     * true if the species name is valid.  false otherwise.
     */
    public boolean isSpeciesNameValid(String speciesName) {
      if (speciesName.length() == 0) {
          return false;
      } else {
          char c = speciesName.charAt(0);
          if ((Character.isLetter(c) == false) && (c != '_')) {
```

```
            return false;
          }
          for(int i=1; i<speciesName.length(); i++) {
            c = speciesName.charAt(i);
            if ((Character.isLetterOrDigit(c) == false) && (c != '_'))
{
                return false;
            }
          }
      }
      return true;
    }

    private int parseSpecies(Vector tokens,
                        int currentTokenIndex,
                        Reaction parsedReaction,
                        boolean isReactant,
                        String reaction) throws ReactionParserException
{
      ReactionToken token =
(ReactionToken)tokens.get(currentTokenIndex++);
      String name = "";
      int coefficient = 1;
      if (token.type == ReactionToken.INT) {
          try {
            coefficient = Integer.parseInt(token.str);
            if (coefficient == 0) {
                throw new ReactionParserException("Zero
Coefficient",reaction,token.location);
            }
          } catch (NumberFormatException e) {
            throw new ReactionParserException("Invalid integer '" +
token.str + "'.",reaction,token.location);
          }
          token = (ReactionToken)tokens.get(currentTokenIndex++);
      }

      if (token.type == ReactionToken.SPECIES) {
          name = token.str;
      } else {
          throw new ReactionParserException("Species
expected.",reaction,token.location);
      }

      try {
          if (isReactant) {
            parsedReaction.addReactant(coefficient,name);
          } else {
            parsedReaction.addProduct(coefficient,name);
          }
      } catch (ReactionException e) {
          // This should never happen.
      }
      return currentTokenIndex;
    }
```

```
    private Vector scan(String rxn) throws ReactionParserException {
      Vector tokens = new Vector();
      int i = 0;
      while (i<rxn.length()) {
          if (Character.isWhitespace(rxn.charAt(i))) {
            i++;
          } else if ((Character.isLetter(rxn.charAt(i))) ||
(rxn.charAt(i) == '_')) {
            int location = i;
            String token = "" + rxn.charAt(i);
            i++;
            while (i<rxn.length()) {
                if ((Character.isLetterOrDigit(rxn.charAt(i)) ||
(rxn.charAt(i) == '_'))) {
                  token += rxn.charAt(i);
                  i++;
                } else {
                  break;
                }
            }
            tokens.add(new
ReactionToken(token,location,ReactionToken.SPECIES));
          } else if (Character.isDigit(rxn.charAt(i))) {
            int location = i;
            String token = "" + rxn.charAt(i);
            i++;
            while (i<rxn.length()) {
                if (Character.isDigit(rxn.charAt(i))) {
                  token += rxn.charAt(i);
                  i++;
                } else {
                  break;
                }
            }
            tokens.add(new
ReactionToken(token,location,ReactionToken.INT));
          } else if ((rxn.charAt(i) == '-') || (rxn.charAt(i) == '='))
{
            if ((i+1<rxn.length()) && (rxn.charAt(i+1) == '>')) {
                tokens.add(new
ReactionToken("=>",i,ReactionToken.PRODUCES));
                i += 2;
            } else {
                throw new ReactionParserException("'"+rxn.charAt(i) +
">' expected.",rxn,i);
            }
          } else if (rxn.charAt(i) == '*') {
            tokens.add(new ReactionToken("*",i,ReactionToken.STAR));
            i++;
          } else if (rxn.charAt(i) == '+') {
            tokens.add(new ReactionToken("+",i,ReactionToken.PLUS));
            i++;
          } else {
```

```
            throw new ReactionParserException("Illegal character '" +
rxn.charAt(i) + "'.",rxn,i);
          }
      }
      tokens.add(new ReactionToken("END OF
STRING",i,ReactionToken.END_OF_STRING));
      return tokens;
    }

    private class ReactionToken {
      public String str;
      public int location;
      public int type;

      ReactionToken(String str,int location,int type) {
          this.str = str;
          this.location = location;
          this.type = type;
      }

      public static final int SPECIES = 1;
      public static final int INT = 2;
      public static final int PLUS = 3;
      public static final int PRODUCES = 4;
      public static final int STAR = 5;
      public static final int END_OF_STRING = 6;
    }

    private ReactionParser() {

    }
}
```

ESSWorker/ReactionParserException.java

```
/**
 * The ReactionParser's exception class.
 *
 * This exception class stores a message regarding why the parse error
occured, the reaction that caused the exception, and the location in
reaction where the parse error occurred.
 * @see ReactionParser
 */
public class ReactionParserException extends Exception {
    private String reaction;
    private int location;

    /**
     * Creates a new reaction parser exception.
     * @param message
     * the error message.
     * @param reaction
     * the reaction string that caused the parsing exception.
     * @param location
```

```
     * the index of the character in the reaction string where the
parse error occurred.
     */
    public ReactionParserException(String message,String reaction,int
location) {
      super(message);
      this.reaction = reaction;
      this.location = location;
    }

    /**
     * Retrieves the reaction that caused the parse error.
     * @return
     * the reaction string.
     */
    public String getReaction() {
      return reaction;
    }

    /**
     * Retrieves the location in the reaction string where the parse
error occurred.
     * @return
     * the index of the character in the reaction string where the
parse error occurred.
     */
    public int getLocation() {
      return location;
    }

    /**
     * Retrieves a message regarding why the error occurred.
     * @return
     * the error message.
     */
    public String getMessage() {
      return super.getMessage();
    }
}
```

Client/DataFile.java

```
import java.io.*;

class DataFile {
    public static void write(TimeSeriesData tsd, String filename)
throws IOException {
      DataOutputStream out = new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(filename)));
      out.writeInt(13579);
      out.writeInt(97531);
      int rows = tsd.data[0].length;
      int columns = tsd.header.length;
      out.writeInt(rows);
      out.writeInt(columns);
```

```
      for(int row=0; row < rows; row++) {
          for(int col=0; col<columns; col++) {
            out.writeDouble(tsd.data[col][row]);
          }
      }
      for(int i=0; i<tsd.header.length; i++) {
          String columnName = tsd.header[i];
          out.writeInt(columnName.length());
          out.writeChars(columnName);
      }
      out.close();
    }

    public static TimeSeriesData read(String filename) throws
IOException {
      DataInputStream in = new DataInputStream(new
BufferedInputStream(new FileInputStream(filename)));
      if (in.readInt() != 13579) throw new IOException("Invalid file
format.");
      if (in.readInt() != 97531) throw new IOException("Invalid file
format.");
      int rows = in.readInt();
      int columns = in.readInt();
      TimeSeriesData tsd = new TimeSeriesData();
      tsd.data = new double[columns][];
      for(int i=0; i<columns; i++) {
          tsd.data[i] = new double[rows];
      }
      for(int row=0; row<tsd.data[0].length; row++) {
          for(int col=0; col<tsd.data.length; col++) {
            tsd.data[col][row] = in.readDouble();
          }
      }
      tsd.header = new String[columns];
      for(int i=0; i<columns; i++) {
          tsd.header[i] = "";
          int length = in.readInt();
          for(int j=0; j<length; j++) {
            tsd.header[i] += in.readChar();
          }
      }
      return tsd;
    }

    public static void sample(String inputFilename, String
outputFilename, int rate) throws IOException {
      DataInputStream in = new DataInputStream(new
BufferedInputStream(new FileInputStream(inputFilename)));
      if (in.readInt() != 13579) throw new IOException("Invalid file
format.");
      if (in.readInt() != 97531) throw new IOException("Invalid file
format.");
      DataOutputStream out = new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(outputFilename)));
```

```
      int in_rows = in.readInt();
      int in_cols = in.readInt();

      int out_rows = in_rows / rate;

      out.writeInt(13579);
      out.writeInt(97531);
      out.writeInt(out_rows);
      out.writeInt(in_cols);

      for(int row=0; row<out_rows; row++) {
          for(int col=0; col<in_cols; col++) {
            out.writeDouble(in.readDouble());
          }
          in.skipBytes(8*in_cols*(rate-1));
      }
      in.skipBytes(8*in_cols*(in_rows % rate));

      int i;
      while((i = in.read()) != -1) {
          out.write(i);
      }
      in.close();
      out.close();
    }

    public static void tsd2tab(String inputFile, String outputFile)
throws IOException {
      TABFile.write(TSDFile.read(inputFile),outputFile);
    }

    public static void tsd2bsd(String inputFile, String outputFile)
throws IOException {
      write(TSDFile.read(inputFile),outputFile);
    }

    public static void bsd2tsd(String inputFile, String outputFile)
throws IOException {
      TSDFile.write(read(inputFile),outputFile);
    }

    public static void bsd2tab(String inputFile, String outputFile)
throws IOException {
      TABFile.write(read(inputFile),outputFile);
    }

    public static void tab2bsd(String inputFile, String outputFile)
throws IOException {
      write(TABFile.read(inputFile),outputFile);
    }

    public static void tab2tsd(String inputFile, String outputFile)
throws IOException {
      TSDFile.write(TABFile.read(inputFile),outputFile);
    }
```

```
    public static void merge(String inputFileA, String inputFileB,
                        String prefixA, String prefixB,
                        String outputFile, boolean zeroPad) throws
IOException {
      DataInputStream inA = new DataInputStream(new
BufferedInputStream(new FileInputStream(inputFileA)));
      DataInputStream inB = new DataInputStream(new
BufferedInputStream(new FileInputStream(inputFileB)));
      if (inA.readInt() != 13579) throw new IOException("Invalid file
format.");
      if (inA.readInt() != 97531) throw new IOException("Invalid file
format.");
      if (inB.readInt() != 13579) throw new IOException("Invalid file
format.");
      if (inB.readInt() != 97531) throw new IOException("Invalid file
format.");
      DataOutputStream out = new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(outputFile)));
      int inArows = inA.readInt();
      int inAcols = inA.readInt();
      int inBrows = inB.readInt();
      int inBcols = inB.readInt();
      int newrows = inArows;
      if (zeroPad) {
          if (inArows < inBrows) newrows = inBrows;
      } else {
          if (inArows > inBrows) newrows = inBrows;
      }
      int newcols = inAcols + inBcols;
      out.writeInt(13579);
      out.writeInt(97531);
      out.writeInt(newrows);
      out.writeInt(newcols);
      for(int row=0; row<newrows; row++) {
          for(int col=0; col<inAcols; col++) {
            if (row<inArows) {
                out.writeDouble(inA.readDouble());
            } else {
                out.writeDouble(0.0);
            }
          }
          for(int col=0; col<inBcols; col++) {
            if (row<inBrows) {
                out.writeDouble(inB.readDouble());
            } else {
                out.writeDouble(0.0);
            }
          }
      }

      if (!zeroPad) {
          if (newrows == inArows) {
            inB.skipBytes(8*inBcols*(inBrows-newrows));
          } else {
```

```
            inA.skipBytes(8*inAcols*(inArows-newrows));
          }
        }

        for(int i=0; i<inAcols; i++) {
            int strlen = inA.readInt();
            String header = "" + prefixA;
            for(int j=0; j<strlen; j++) {
              header += inA.readChar();
            }
            out.writeInt(header.length());
            out.writeChars(header);
        }
        for(int i=0; i<inBcols; i++) {
            int strlen = inB.readInt();
            String header = "" + prefixB;
            for(int j=0; j<strlen; j++) {
              header += inB.readChar();
            }
            out.writeInt(header.length());
            out.writeChars(header);
        }
        out.close();
        inA.close();
        inB.close();
      }
}
```

Client/Client.java

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;

public class Client extends JFrame {
    JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.BOTTOM);
    ModelPanel modelPanel = new ModelPanel(this);
    ParameterPanel parameterPanel = new ParameterPanel(this);
    PostProcessingPanel postProcessingPanel = new
PostProcessingPanel(this);
    JFileChooser fileChooser = new JFileChooser();
    ErrorWindow errorWindow = new ErrorWindow();
    String host;
    int port;

    public Client(String host,int port) {
      super("BioGrid Client v1.0");
      this.host = host;
      this.port = port;
      setSize(500,500);
      setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
      addWindowListener(new WindowAdapter() {
```

```
            public void windowClosing(WindowEvent e) { onFileExit();
}});
      initPanels();
      initMenu();
      show();
    }

    public JFileChooser getFileChooser() {
      return fileChooser;
    }

    public void onFileRun() {
      modelPanel.stopEditing();
      parameterPanel.stopEditing();
      postProcessingPanel.stopEditing();

      ClientData clientData = new ClientData(this);
      modelPanel.setData(clientData);
      parameterPanel.setData(clientData);
      postProcessingPanel.setData(clientData);

      Vector errors = clientData.check();
      if (errors.size() != 0) {
          String errorText = "";
          for(int i=0; i<errors.size(); i++) {
            errorText += errors.get(i).toString();
            errorText += "\n";
          }
          errorWindow.setText(errorText);
          errorWindow.show();
          return;
      }

      try {
          clientData.run(host,port);
      } catch (IOException e) {
          JOptionPane.showMessageDialog(this,
                                e.getMessage(),
                                "BioGrid Client v1.0",
                                JOptionPane.ERROR_MESSAGE);
      } catch (ServerException e) {
          JOptionPane.showMessageDialog(this,
                                e.getMessage(),
                                "BioGrid Client v1.0",
                                JOptionPane.ERROR_MESSAGE);
      }
    }

    public void onFileExit() {
      System.exit(0);
    }

    private void initPanels() {
      tabbedPane.add(modelPanel,"Model");
      tabbedPane.add(parameterPanel,"Parameter");
```

```
      tabbedPane.add(postProcessingPanel,"Post Processing");
      getContentPane().add(tabbedPane);
    }

    private void initMenu() {
      JMenu fileMenu = new JMenu("File");
      JMenuItem fileRunMenuItem = new JMenuItem("Run");
      JMenuItem fileExitMenuItem = new JMenuItem("Exit");
      fileRunMenuItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) { onFileRun();
}});
      fileExitMenuItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) { onFileExit();
}});
      fileMenu.add(fileRunMenuItem);
      fileMenu.addSeparator();
      fileMenu.add(fileExitMenuItem);

      JMenuBar menuBar = new JMenuBar();
      menuBar.add(fileMenu);
      setJMenuBar(menuBar);
    }

    public static void main(String args[]) {
      System.out.println();
      System.out.println("BioGrid Client v1.0");
      System.out.println("written by James M. McCollum
(jmccoll2@utk.edu)");
      System.out.println("University of Tennessee - Knoxville");
      System.out.println("http://biocomp.ece.utk.edu");
      System.out.println();
      if (args.length != 2) {
          System.out.println("usage: java Client <host> <port>");
          return;
      }
      try {
          Client c = new Client(args[0],Integer.parseInt(args[1]));
      } catch (NumberFormatException e) {
          System.out.println("usage: java Client <host> <port>");
      }
    }
}
```

Client/ClientData.java

```
import java.util.*;
import java.io.*;
import java.awt.*;
import javax.swing.*;

class ClientData {
    String printInterval;
    String endTime;
    String outputDirectory;
```

```
    String modelFileName;
    String reactionParameters[];
    String speciesParameters[];
    String parameterNames[];
    String parameterValues[];
    String functionNames[];
    String functionExpressions[];
    String parameters[][];
    Frame parent;

    JobId essJobs[];
    JobId octaveJobs[][];
    RunWindow runWindow;
    ServerConnection connection;
    TimerTask runTask;

    public ClientData(Frame parent) {
      this.parent = parent;
    }

    public void run(String host, int port) throws
IOException,ServerException {
      int rows = 1;
      for(int i=0; i<parameterValues.length; i++) {
          rows *= parameterValues[i].split(",").length;
      }

      int indicies[][] = new int[parameterNames.length][];
      for(int i=0; i<parameterNames.length; i++) {
          indicies[i] = new int[rows];
      }

      for(int i=0; i<rows; i++) {
          indicies[0][i] = i % (parameterValues[0].split(",").length);
      }

      for(int i=1; i<parameterNames.length; i++) {
          int nValue = 0;
          int mValue = 0;
          for(int j=0; j<rows; j++) {
            if ((nValue != indicies[i-1][j]) && (indicies[i-1][j] ==
0)) {
                mValue = (mValue + 1) %
parameterValues[i].split(",").length;
            }
            indicies[i][j] = mValue;
            nValue = indicies[i-1][j];
          }
      }

      parameters = new String[parameterNames.length][];
      for(int i=0; i<parameterNames.length; i++) {
          parameters[i] = new String[rows];
      }
```

```
      for(int j=0; j<parameterNames.length; j++) {
          String val[] = parameterValues[j].split(",");
          for(int i=0; i<rows; i++) {
            parameters[j][i] = val[indicies[j][i]];
          }
      }

      BioSpreadsheetData data = new BioSpreadsheetData();
      data.load(modelFileName);
      new File(outputDirectory).mkdirs();
      for(int row=0; row<rows; row++) {
          String sbmlFilename = outputDirectory + File.separator +
(row+1) + ".sbml";
          for(int i=0; i<data.getSpeciesCount(); i++) {
            String speciesParameter = speciesParameters[i];
            if (!speciesParameter.equals("")) {
                for(int j=0; j<parameterNames.length; j++) {
                  if (parameterNames[j].equals(speciesParameter)) {
                      data.setSpeciesPopulation(i,parameters[j][row]);
                  }
                }
            }
          }
          for(int i=0; i<data.getReactionCount(); i++) {
            String reactionParameter = reactionParameters[i];
            if (!reactionParameter.equals("")) {
                for(int j=0; j<parameterNames.length; j++) {
                  if (parameterNames[j].equals(reactionParameter)) {
                      data.setReactionRate(i,parameters[j][row]);
                  }
                }
            }
          }
          data.save(sbmlFilename);

          String essParameterFile = outputDirectory + File.separator +
(row+1) + ".ess.parameters";
          DataOutputStream out = new DataOutputStream(new
FileOutputStream(essParameterFile));
          out.writeDouble(Double.parseDouble(printInterval));
          out.writeDouble(Double.parseDouble(endTime));
          out.writeInt(Integer.parseInt(parameters[0][row]));
          out.close();
      }

      connection = new ServerConnection(host,port,"Client");
      FileId sbmlInputFiles[] = new FileId[rows];
      FileId essParameterFiles[] = new FileId[rows];
      essJobs = new JobId[rows];
      octaveJobs = new JobId[rows][];

      File octaveFiles[] = new File[functionNames.length];
      FileId octaveFileId[] = new FileId[functionNames.length];
      for(int i=0; i<functionNames.length; i++) {
```

```
          octaveFiles[i] = new File(outputDirectory + File.separator +
(i+1) + ".octave");
          PrintStream out = new PrintStream(new
FileOutputStream(octaveFiles[i]));
          out.println(functionNames[i] + " = " +
functionExpressions[i]);
          out.println(functionNames[i]);
          out.close();
          octaveFileId[i] = connection.addFile(octaveFiles[i]);
      }

      for(int i=0; i<rows; i++) {
          sbmlInputFiles[i] = connection.addFile(new
File(outputDirectory + File.separator + (i+1) + ".sbml"));
          essParameterFiles[i] = connection.addFile(new
File(outputDirectory + File.separator + (i+1) + ".ess.parameters"));
          FileId temp[] = {sbmlInputFiles[i],essParameterFiles[i]};
          essJobs[i] = connection.addJob("ess",temp,1);
          octaveJobs[i] = new JobId[functionNames.length];
          for(int j=0; j<functionNames.length; j++) {
            FileId temp2[] =
{(connection.getJob(essJobs[i]).getOutputs())[0],octaveFileId[j]};
            octaveJobs[i][j] = connection.addJob("octave",temp2,1);
          }
      }

      runWindow = new
RunWindow(parent,rows,parameters,parameterNames,functionNames);
      runTask = new TimerTask() { public void run() { updateAll(); } };
      java.util.Timer timer = new java.util.Timer();
      timer.schedule(runTask,0,500);
      runWindow.show();
    }

    public void updateAll() {
      try {
          boolean done = true;
          Job essjobs[] = connection.getJob(essJobs);
          for(int i=0; i<essjobs.length; i++) {
            if ((essjobs[i].getStatus() != JobStatus.COMPLETE) &&
                (essjobs[i].getStatus() != JobStatus.ERROR)) {
                done = false;
            }

      runWindow.setOutputStatus(i,essjobs[i].getStatus().toString());
          }
          for(int i=0; i<essJobs.length; i++) {
            Job octavejobs[] = connection.getJob(octaveJobs[i]);
            for(int j=0; j<octavejobs.length; j++) {
                if ((octavejobs[j].getStatus() != JobStatus.COMPLETE)
&&
                  (octavejobs[j].getStatus() != JobStatus.ERROR)) {
                  done = false;
                }
```

```
runWindow.setFunctionStatus(i,j,octavejobs[j].getStatus().toString());
            }
          }

          if (done) {
            runTask.cancel();
            runWindow.hide();

            PrintStream out = new PrintStream(new
FileOutputStream(outputDirectory + File.separator + "index.html"));
            BioSpreadsheetData data = new BioSpreadsheetData();
            data.load(modelFileName);

            out.println("<html>");

            out.println("<p><h3>Input SBML File: " + modelFileName +
"</h3></p>");

            out.println("<p><h3>Species:</h3></p>");
            out.println("<table border=1 cellpadding=5
cellspacing=1>");

      out.println("<tr><td>Species</td><td>Population</td><td>Parameter
</td></tr>");
            for(int i=0; i<data.getSpeciesCount(); i++) {
                out.println("<tr>");
                out.println("<td>" + data.getSpeciesName(i) + "</td>");
                out.println("<td>" + data.getSpeciesPopulation(i) +
"</td>");
                out.println("<td>" + speciesParameters[i] + "</td>");
                out.println("</tr>");
            }
            out.println("</table>");

            out.println("<p><h3>Reactions:</h3></p>");
            out.println("<table border=1 cellpadding=5
cellspacing=1>");

      out.println("<tr><td>Reaction</td><td>Rate</td><td>Parameter</td>
</tr>");
            for(int i=0; i<data.getReactionCount(); i++) {
                out.println("<tr>");
                out.println("<td>" + data.getReaction(i) + "</td>");
                out.println("<td>" + data.getReactionRate(i) +
"</td>");
                out.println("<td>" + reactionParameters[i] + "</td>");
                out.println("</tr>");
            }
            out.println("</table>");

            out.println("<p><h3>Print Interval: " + printInterval +
"</h3></p>");

            out.println("<p><h3>End Time: " + endTime + "</h3></p>");
```

```
            out.println("<p><h3>Parameters:</h3></p>");
            out.println("<table border=1 cellpadding=5
cellspacing=1>");
            out.println("<tr><td>Parameter</td><td>Value</td></tr>");
            for(int i=0; i<parameterNames.length; i++) {
                out.println("<tr>");
                out.println("<td>" + parameterNames[i] + "</td>");
                out.println("<td>" + parameterValues[i] + "</td>");
                out.println("</tr>");
            }
            out.println("</table>");

            out.println("<p><h3>Output Data:</h3></p>");
            out.println("<table border=1 cellpadding=5
cellspacing=1>");
            out.println("<tr>");
            out.println("<td>Model File</td>");
            for(int i=0; i<parameterNames.length; i++) {
                out.println("<td>" + parameterNames[i] + "</td>");
            }
            out.println("<td>Output File</td>");
            for(int i=0; i<functionNames.length; i++) {
                out.println("<td>" + functionNames[i] + "</td>");
            }
            out.println("</tr>");


            for(int i=0; i<essjobs.length; i++) {
                out.println("<tr>");
                out.println("<td>" + (i+1) + ".sbml</td>");
                for(int j=0; j<parameterNames.length; j++) {
                  out.println("<td>" + parameters[j][i] + "</td>");
                }
                if (essjobs[i].getStatus() == JobStatus.COMPLETE) {
                  FileId fileIds[] = essjobs[i].getOutputs();
                  connection.getFile(fileIds[0],new
File(outputDirectory +
                                                    File.separator +
                                                    (i+1) + ".ess.out"));
                  out.println("<td>" + (i+1) + ".ess.out</td>");
                } else {
                  out.println("<td>Error</td>");
                }
                Job octavejobs[] = connection.getJob(octaveJobs[i]);
                for(int j=0; j<octavejobs.length; j++) {
                  if (octavejobs[j].getStatus() == JobStatus.COMPLETE)
{
                      FileId fileIds[] = octavejobs[j].getOutputs();
                      connection.getFile(fileIds[0],new
File(outputDirectory +
                                                      File.separator +
                                                      (i+1) + ".fun" +
(j+1) + ".out"));
                      try {
```

```
                        TimeSeriesData tsd =
DataFile.read(outputDirectory +
                                                    File.separator +
                                                    (i+1) + ".fun" +
(j+1) + ".out");
                        if ((tsd.header.length == 1) &&
(tsd.data[0].length == 1)) {
                            out.println("<td>" + tsd.data[0][0] +
"</td>");
                        } else {
                            out.println("<td>" + (i+1) + ".fun" + (j+1)
+ ".out</td>");
                        }
                      } catch (IOException e) {
                        out.println("<td>" + (i+1) + ".fun" + (j+1) +
".out</td>");
                      }
                    } else {
                        out.println("<td>Error</td>");
                    }
                  }
                  out.println("</tr>");
              }
              out.println("</html>");
              out.close();

              connection.close();

              JOptionPane.showMessageDialog(null,
                                        "Run completed successfully.  Open
index.html for output information.",
                                        "Run Window",
                                        JOptionPane.INFORMATION_MESSAGE);
            }
        } catch (IOException e) {
            runTask.cancel();
            runWindow.hide();
            JOptionPane.showMessageDialog(null,
                                    "Error reading data: " + e.getMessage()
+ ", run cancelled.",
                                    "Run Window",
                                    JOptionPane.ERROR_MESSAGE);
        } catch (ServerException e) {
            runTask.cancel();
            runWindow.hide();
            JOptionPane.showMessageDialog(null,
                                    "Error reading data: " + e.getMessage()
+ ", run cancelled.",
                                    "Run Window",
                                    JOptionPane.ERROR_MESSAGE);
        }
      }

      public Vector check() {
        Vector errors = new Vector();
```

```
      try {
          Double.parseDouble(printInterval);
      } catch (NumberFormatException e) {
          errors.add("Invalid print interval value: " + printInterval);
      }

      try {
          Double.parseDouble(endTime);
      } catch (NumberFormatException e) {
          errors.add("Invalid end time value: " + endTime);
      }

      if (modelFileName.equals("")) {
          errors.add("No model filename entered.");
      }
      for(int i=0; i<parameterNames.length; i++) {
          if (parameterNames[i].equals("")) {
            errors.add("Invalid parameter name: \"\".");
          }
      }

      for(int i=0; i<speciesParameters.length; i++) {
          if (!speciesParameters[i].equals("")) {
            boolean valid = false;
            for(int j=0; j<parameterNames.length; j++) {
                if (parameterNames[j].equals(speciesParameters[i])) {
                  valid = true;
                }
            }
            if (!valid) errors.add("Undeclared parameter: \"" +
speciesParameters[i] + "\".");
          }
      }

      for(int i=0; i<reactionParameters.length; i++) {
          if (!reactionParameters[i].equals("")) {
            boolean valid = false;
            for(int j=0; j<parameterNames.length; j++) {
                if (parameterNames[j].equals(reactionParameters[i])) {
                  valid = true;
                }
            }
            if (!valid) errors.add("Undeclared parameter: \"" +
reactionParameters[i] + "\".");
          }
      }

      ReactionParser rp = ReactionParser.getReactionParser();
      for(int i=0; i<functionNames.length; i++) {
          if (!rp.isSpeciesNameValid(functionNames[i])) {
            errors.add("Invalid output name: \"" + functionNames[i] +
"\".");
          }
      }
```

```
      for(int i=0; i<parameterValues.length; i++) {
          String values[] = parameterValues[i].split(",");
          for(int j=0; j<values.length; j++) {
            try {
                Double.parseDouble(values[j]);
            } catch (NumberFormatException e) {
                errors.add("Invalid parameter value: \"" + values[j] +
"\" for parameter \"" + parameterNames[i] + "\".");
            }
          }
      }

      return errors;
    }
}
```

Client/ParameterPanel.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import java.util.*;
import java.lang.*;

class ParameterPanel extends JPanel {
    ParameterPanelTableModel tableModel = new
ParameterPanelTableModel();
    JTable table;
    Client client;

    public class Parameter {
      String name = "";
      String values = "";
    }

    public class ParameterPanelTableModel extends AbstractTableModel {
      String columnNames[] = {"Parameter","Values"};
      public Vector data = new Vector();

      ParameterPanelTableModel() {
          removeAll();
      }

      public void removeAll() {
          data = new Vector();
          addRow("Print Interval","");
          addRow("End Time","");
          addRow("Output Directory","");
          addRow("Seed","");
          fireTableDataChanged();
      }

      public void addRow(String name, String values) {
```

```
        Parameter p = new Parameter();
        p.name = name;
        p.values = values;
        data.add(p);
        fireTableDataChanged();
    }

    public void addRow() {
        data.add(new Parameter());
        fireTableDataChanged();
    }

    public void removeRow(int row) {
        if (row > 3) {
          data.remove(row);
          fireTableDataChanged();
        }
    }

    public Vector getData() {
        return data;
    }

    public String getColumnName(int col) { return columnNames[col]; }
    public int getColumnCount() { return columnNames.length; }
    public int getRowCount() { return data.size(); }
    public boolean isCellEditable(int row, int col) {
        if ((row <= 3) && (col == 0))
          return false;
        else
          return true;
    }

    public Class getColumnClass(int col) {
        return String.class;
    }

    public Object getValueAt(int row, int col) {
        Parameter p = (Parameter)(data.get(row));
        if (col == 0) return p.name;
        else return p.values;
    }

    public void setValueAt(Object value, int row, int col) {
        Parameter p = (Parameter)(data.get(row));
        if (col == 0) p.name = (String)value;
        else p.values = (String)value;
    }
}

public ParameterPanel(Client client) {
    super();
    this.client = client;
    setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
```

```
      int columnWidths[] = {200,600};
      table = new JTable(tableModel);
      for (int i = 0; i < columnWidths.length; i++) {

table.getColumnModel().getColumn(i).setPreferredWidth(columnWidths[i]);
      }

      JButton addRowButton = new JButton("Add Parameter");
      JButton removeRowButton = new JButton("Remove Parameter");
      addRowButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) { onAddRow();
}});
      removeRowButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) { onRemoveRow();
}});

      gbc.anchor = GridBagConstraints.NORTHEAST;
      gbc.fill = GridBagConstraints.BOTH;
      gbc.insets = new Insets(5,5,5,5);
      gbc.weightx = 1;
      gbc.weighty = 1;
      gbc.gridwidth = 3;
      addComponent(new JScrollPane(table),0,0,gbc);
      gbc.gridwidth = 1;
      gbc.weighty = 0;
      addComponent(new JPanel(),0,1,gbc);
      gbc.anchor = GridBagConstraints.CENTER;
      gbc.fill = GridBagConstraints.NONE;
      gbc.weightx = 0;
      addComponent(addRowButton,1,1,gbc);
      addComponent(removeRowButton,2,1,gbc);

      table.setDefaultRenderer(String.class, new
MonospacedCellRenderer());
      JTextField textField = new JTextField();
      textField.setFont(new Font("Monospaced",Font.PLAIN,12));
      table.getColumnModel().getColumn(0).setCellEditor(new
DefaultCellEditor(textField));
      table.getColumnModel().getColumn(1).setCellEditor(new
DefaultCellEditor(textField));
    }

    private void onAddRow() {
      tableModel.addRow();
    }

    private void onRemoveRow() {
      int row = table.getSelectedRow();
      if (row != -1) tableModel.removeRow(row);
    }

    private void addComponent(Component c, int x, int y,
GridBagConstraints gbc) {
      gbc.gridx = x;
```

```
      gbc.gridy = y;
      add(c,gbc);
    }

    public void stopEditing() {
      table.editCellAt(-1,-1);
    }

    public void clearAll() {
      tableModel.removeAll();
    }

    public void setData(ClientData data) {
      data.printInterval = tableModel.getValueAt(0,1).toString();
      data.endTime = tableModel.getValueAt(1,1).toString();
      data.outputDirectory = tableModel.getValueAt(2,1).toString();
      data.parameterNames = new String[tableModel.getRowCount()-3];
      data.parameterValues = new String[tableModel.getRowCount()-3];

      data.parameterNames[0] = "Seed";
      data.parameterValues[0] = tableModel.getValueAt(3,1).toString();
      for(int i=0; i<tableModel.getRowCount()-4; i++) {
          data.parameterNames[i+1] =
tableModel.getValueAt(i+4,0).toString().trim();
          data.parameterValues[i+1] =
tableModel.getValueAt(i+4,1).toString().trim();
      }
    }
}
```

Client/MonospacedCellRenderer.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import java.util.*;
import java.lang.*;

public class MonospacedCellRenderer extends DefaultTableCellRenderer {
    Font font = new Font("Monospaced",Font.PLAIN,12);

    public Component getTableCellRendererComponent(JTable table, Object
value, boolean isSelected, boolean hasFocus, int row, int column) {
      JLabel label =
(JLabel)super.getTableCellRendererComponent(table,value,isSelected,hasF
ocus,row,column);
      label.setFont(font);
      return label;
    }
}
```

Client/TSDFile.java

```
import java.io.*;
```

```
import java.util.*;

class TSDFile {
    public static TimeSeriesData read(String filename)
      throws IOException {

      TimeSeriesData tsd = new TimeSeriesData();

      int columnCount = getColumnCount(filename);
      int rowCount = getRowCount(filename);
      if ((columnCount < 0) || (rowCount < 0)) {
          throw new IOException("Bad time series data file format.");
      }

      tsd.header = new String[columnCount];
      tsd.data = new double[columnCount][];
      for(int i=0; i<columnCount; i++) {
          tsd.data[i] = new double[rowCount];
      }

      BufferedReader in = new BufferedReader(new FileReader(filename));

      in.read();
      if (columnCount > 1) {
          in.read();
      }

      for(int i=0; i<columnCount; i++) {
          tsd.header[i] = getNextString(in);
      }
      for(int i=0; i<rowCount; i++) {
          for(int j=0; j<columnCount; j++) {
            tsd.data[j][i] = getNextDouble(in);
          }
      }

      in.close();
      return tsd;
    }

    public static void write(TimeSeriesData tsd, String filename)
      throws IOException {
      PrintStream out = new PrintStream(new
FileOutputStream(filename));
      if (tsd.header.length > 1) {
          out.print("((");
          for(int i=0; i<tsd.header.length; i++) {
            if (i != 0) out.print(",");
            out.print("\"");
            out.print(tsd.header[i]);
            out.print("\"");
          }
          for(int i=0; i<tsd.data[0].length; i++) {
            out.print("),(");
            for(int j=0; j<tsd.data.length; j++) {
```

```
                if (j != 0) out.print(",");
                out.print(tsd.data[j][i]);
              }
            }
            out.print("))");
        } else {
            out.print("(\"");
            out.print(tsd.header[0]);
            out.print("\"");
            for(int i=0; i<tsd.data[0].length; i++) {
              out.print(",");
              out.print(tsd.data[0][i]);
            }
            out.print(")");
        }
        out.close();
    }

    private static int getColumnCount(String filename) throws
IOException {
        BufferedReader in = new BufferedReader(new FileReader(filename));
        int c = in.read();
        c = in.read();
        if (c == '(') {
            int columnCount = 1;
            while ((c != -1) && (c != ')')) {
              if (c == ',') columnCount++;
              c = in.read();
            }
            return columnCount;
        } else {
            return 1;
        }
    }

    private static int getRowCount(String filename) throws IOException
{
        BufferedReader in = new BufferedReader(new FileReader(filename));
        int c = in.read();
        c = in.read();
        if (c == '(') {
            int rowCount = -2;
            while (c != -1) {
              if (c == ')') {
                  rowCount++;
              }
              c = in.read();
            }
            return rowCount;
        } else {
            int rowCount = 0;
            while (c != -1) {
              if (c == ',') {
                  rowCount++;
              }
```

```
            c = in.read();
          }
          return rowCount;
      }
    }

    private static double getNextDouble(BufferedReader in) throws
IOException {
      String value = "";
      int c = in.read();
      if (c == -1) {
          throw new IOException("End of file.");
      }
      while ((c != ')') && (c != ',') && (c != -1)) {
          value += (char)c;
          c = in.read();
      }
      try {
          if (c == ')') {
            in.read();
            in.read();
          }
          return Double.parseDouble(value);
      } catch (NumberFormatException e) {
          throw new IOException("Invalid number: " + value);
      }
    }

    public static String getNextString(BufferedReader in) throws
IOException {
      String value = "";
      int c = in.read();
      if (c != '"') throw new IOException("String expected.");
      c = in.read();
      while((c != -1) && (c != '"')) {
          value += (char)c;
          c = in.read();
      }
      c = in.read();
      if (c == ')') {
          in.read();
          in.read();
      }
      return value;
    }
}
```

Client/ReactionParser.java

```
import java.util.*;
import java.io.*;

/**
 * Class used to parse strings.
 * This is a singleton class so the constructor is private.
```

```
 * To retrieve an instance of this class, call the getReactionParser()
method.
 *
 * The string is parsed using the following reaction grammar:
 *
 * <pre>[reaction]     := [species-list] '=>' [species-list]</pre>
 * <pre>               := [species-list] '=>' '*'</pre>
 *
 * <pre>[species-list] := [species] '+' [species-list]</pre>
 * <pre>               := [species]</pre>
 *
 * <pre>[species]      := [unsigned-integer] [species-name]</pre>
 * <pre>               := [species-name]</pre>
 *
 * Species names may contain letters, numbers, or underscores, but the
first character must not be a number.
 *
 * <pre>Valid Reaction Examples:</pre>
 * <pre>   2 H + O => H2O</pre>
 * <pre>   H2S + 2 _Catalyst_ => *</pre>
 * @see Reaction
 */
public class ReactionParser {
    private static ReactionParser parser = null;

    /**
     * Retrieves a ReactionParser
     *
     * @return the ReactionParser
     */
    public static ReactionParser getReactionParser() {
      if (parser == null) {
          parser = new ReactionParser();
      }
      return parser;
    }

    /**
     * Converts a string into a reaction.
     *
     * @param reaction
     * the reaction to be parsed.
     * @return
     * the parsed reaction.
     * @exception ReactionParserException
     * if the reaction is not of the right format, this exception will
be thrown.
     */
    public Reaction parse(String reaction) throws
ReactionParserException {
      Reaction parsedReaction = new Reaction();
      Vector tokens = scan(reaction);
      int currentTokenIndex = 0;
      currentTokenIndex =
parseSpecies(tokens,currentTokenIndex,parsedReaction,true,reaction);
```

```
      ReactionToken token =
(ReactionToken)tokens.get(currentTokenIndex++);
      while (token.type == ReactionToken.PLUS) {
          currentTokenIndex =
parseSpecies(tokens,currentTokenIndex,parsedReaction,true,reaction);
          token = (ReactionToken)tokens.get(currentTokenIndex++);
      }
      if (token.type != ReactionToken.PRODUCES) {
          throw new ReactionParserException("'+' or '=>'
expected.",reaction,token.location);
      }
      token = (ReactionToken)tokens.get(currentTokenIndex++);
      if (token.type == ReactionToken.STAR) {
          token = (ReactionToken)tokens.get(currentTokenIndex++);
      } else if ((token.type == ReactionToken.INT) || (token.type ==
ReactionToken.SPECIES)) {
          currentTokenIndex--;
          currentTokenIndex =
parseSpecies(tokens,currentTokenIndex,parsedReaction,false,reaction);
          token = (ReactionToken)tokens.get(currentTokenIndex++);
          while (token.type == ReactionToken.PLUS) {
            currentTokenIndex =
parseSpecies(tokens,currentTokenIndex,parsedReaction,false,reaction);
            token = (ReactionToken)tokens.get(currentTokenIndex++);
          }
      } else {
          throw new ReactionParserException("'*' or species
expected.",reaction,token.location);
      }
      if (token.type != ReactionToken.END_OF_STRING) {
          throw new ReactionParserException("End of string
expected.",reaction,token.location);
      }
      return parsedReaction;
    }

    /**
     * Converts a string into a reaction.
     *
     * @param reaction
     * the reaction to be parsed.
     * @param validSpeciesNames
     * an array of valid species names
     * @return
     * the parsed reaction.
     * @exception ReactionParserException
     * if the reaction is not of the right format or contains an
invalid species name, this exception will be thrown.
     */
    public Reaction parse(String reaction, String[] validSpeciesNames)
throws ReactionParserException {
      Vector tokens = scan(reaction);
      for(int i=0; i<tokens.size(); i++) {
          ReactionToken token = (ReactionToken)tokens.get(i);
          if (token.type == ReactionToken.SPECIES) {
```

```
            boolean found = false;
            for(int j=0; j<validSpeciesNames.length; j++) {
                if (token.str.equals(validSpeciesNames[j])) {
                  found = true;
                  break;
                }
            }
            if (found == false) {
                throw new ReactionParserException("Undeclared species:
\"" + token.str + "\".",reaction,token.location);
            }
          }
      }
      return parse(reaction);
    }

    /**
     * Determines if a string is a valid species name..
     *
     * @param speciesName
     * the species name to check.
     * @return
     * true if the species name is valid.  false otherwise.
     */
    public boolean isSpeciesNameValid(String speciesName) {
      if (speciesName.length() == 0) {
          return false;
      } else {
          char c = speciesName.charAt(0);
          if ((Character.isLetter(c) == false) && (c != '_')) {
            return false;
          }
          for(int i=1; i<speciesName.length(); i++) {
            c = speciesName.charAt(i);
            if ((Character.isLetterOrDigit(c) == false) && (c != '_'))
{
                return false;
            }
          }
      }
      return true;
    }

    private int parseSpecies(Vector tokens,
                       int currentTokenIndex,
                       Reaction parsedReaction,
                       boolean isReactant,
                       String reaction) throws ReactionParserException
{
      ReactionToken token =
(ReactionToken)tokens.get(currentTokenIndex++);
      String name = "";
      int coefficient = 1;
      if (token.type == ReactionToken.INT) {
          try {
```

```
            coefficient = Integer.parseInt(token.str);
            if (coefficient == 0) {
                throw new ReactionParserException("Zero
Coefficient",reaction,token.location);
            }
          } catch (NumberFormatException e) {
            throw new ReactionParserException("Invalid integer '" +
token.str + "'.",reaction,token.location);
          }
          token = (ReactionToken)tokens.get(currentTokenIndex++);
      }

      if (token.type == ReactionToken.SPECIES) {
          name = token.str;
      } else {
          throw new ReactionParserException("Species
expected.",reaction,token.location);
      }

      try {
          if (isReactant) {
            parsedReaction.addReactant(coefficient,name);
          } else {
            parsedReaction.addProduct(coefficient,name);
          }
      } catch (ReactionException e) {
          // This should never happen.
      }
      return currentTokenIndex;
    }

    private Vector scan(String rxn) throws ReactionParserException {
      Vector tokens = new Vector();
      int i = 0;
      while (i<rxn.length()) {
          if (Character.isWhitespace(rxn.charAt(i))) {
            i++;
          } else if ((Character.isLetter(rxn.charAt(i))) ||
(rxn.charAt(i) == '_')) {
            int location = i;
            String token = "" + rxn.charAt(i);
            i++;
            while (i<rxn.length()) {
                if ((Character.isLetterOrDigit(rxn.charAt(i)) ||
(rxn.charAt(i) == '_'))) {
                  token += rxn.charAt(i);
                  i++;
                } else {
                  break;
                }
            }
            tokens.add(new
ReactionToken(token,location,ReactionToken.SPECIES));
          } else if (Character.isDigit(rxn.charAt(i))) {
            int location = i;
```

```
            String token = "" + rxn.charAt(i);
            i++;
            while (i<rxn.length()) {
                if (Character.isDigit(rxn.charAt(i))) {
                  token += rxn.charAt(i);
                  i++;
                } else {
                  break;
                }
            }
            tokens.add(new
ReactionToken(token,location,ReactionToken.INT));
          } else if ((rxn.charAt(i) == '-') || (rxn.charAt(i) == '='))
{
            if ((i+1<rxn.length()) && (rxn.charAt(i+1) == '>')) {
                tokens.add(new
ReactionToken("=>",i,ReactionToken.PRODUCES));
                i += 2;
            } else {
                throw new ReactionParserException("'"+rxn.charAt(i) +
">' expected.",rxn,i);
            }
          } else if (rxn.charAt(i) == '*') {
            tokens.add(new ReactionToken("*",i,ReactionToken.STAR));
            i++;
          } else if (rxn.charAt(i) == '+') {
            tokens.add(new ReactionToken("+",i,ReactionToken.PLUS));
            i++;
          } else {
            throw new ReactionParserException("Illegal character '" +
rxn.charAt(i) + "'.",rxn,i);
          }
      }
      tokens.add(new ReactionToken("END OF
STRING",i,ReactionToken.END_OF_STRING));
      return tokens;
    }

    private class ReactionToken {
      public String str;
      public int location;
      public int type;

      ReactionToken(String str,int location,int type) {
          this.str = str;
          this.location = location;
          this.type = type;
      }

      public static final int SPECIES = 1;
      public static final int INT = 2;
      public static final int PLUS = 3;
      public static final int PRODUCES = 4;
      public static final int STAR = 5;
      public static final int END_OF_STRING = 6;
```

```
    }

    private ReactionParser() {

    }
}
```

Client/ReactionParserException.java

```
/**
 * The ReactionParser's exception class.
 *
 * This exception class stores a message regarding why the parse error
occured, the reaction that caused the exception, and the location in
reaction where the parse error occurred.
 * @see ReactionParser
 */
public class ReactionParserException extends Exception {
    private String reaction;
    private int location;

    /**
     * Creates a new reaction parser exception.
     * @param message
     * the error message.
     * @param reaction
     * the reaction string that caused the parsing exception.
     * @param location
     * the index of the character in the reaction string where the
parse error occurred.
     */
    public ReactionParserException(String message,String reaction,int
location) {
      super(message);
      this.reaction = reaction;
      this.location = location;
    }

    /**
     * Retrieves the reaction that caused the parse error.
     * @return
     * the reaction string.
     */
    public String getReaction() {
      return reaction;
    }

    /**
     * Retrieves the location in the reaction string where the parse
error occurred.
     * @return
     * the index of the character in the reaction string where the
parse error occurred.
     */
    public int getLocation() {
```

```
      return location;
    }

    /**
     * Retrieves a message regarding why the error occurred.
     * @return
     * the error message.
     */
    public String getMessage() {
      return super.getMessage();
    }
}
```

Client/TABFile.java

```
import java.io.*;

class TABFile {
    public static TimeSeriesData read(String filename) throws
IOException {
      int columnCount = getColumnCount(filename);
      int rowCount = getRowCount(filename);

      String header[];
      double data[][] = new double[columnCount][];
      for(int i=0; i<columnCount; i++) {
          data[i] = new double[rowCount];
      }

      BufferedReader in = new BufferedReader(new FileReader(filename));
      header = in.readLine().split("\\t");

      for(int i=0; i<rowCount; i++) {
          String dataStr[] = in.readLine().split("\\t");

          if (dataStr.length != columnCount) {
            throw new IOException("Bad file format.");
          }

          for(int j=0; j<columnCount; j++) {
            try {
                data[j][i] = Double.parseDouble(dataStr[j]);
            } catch(NumberFormatException e) {
                throw new IOException("Bad file format.");
            }
          }
      }

      TimeSeriesData tsd = new TimeSeriesData();
      tsd.data = data;
      tsd.header = header;
      return tsd;
    }
```

```
    public static void write(TimeSeriesData data,String filename)
throws IOException {
      PrintStream out = new PrintStream(new
FileOutputStream(filename));
      for(int i=0; i<data.header.length; i++) {
          if (i != 0) out.print("\t");
          out.print(data.header[i]);
      }
      out.println();

      for(int i=0; i<data.data[0].length; i++) {
          for(int j=0; j<data.data.length; j++) {
            if (j != 0) out.print("\t");
            out.print(data.data[j][i]);
          }
          out.println();
      }
      out.close();
    }

    private static int getColumnCount(String filename) throws
IOException {
      BufferedReader in = new BufferedReader(new FileReader(filename));
      int c = in.read();
      int columnCount = 1;
      while ((c != '\n') && (c != -1)) {
          if (c == '\t') {
            columnCount++;
          }
          c = in.read();
      }
      in.close();
      return columnCount;
    }

    private static int getRowCount(String filename) throws IOException
{
      BufferedReader in = new BufferedReader(new FileReader(filename));
      int c = in.read();
      int rowCount = -1;
      while (c != -1) {
          if (c == '\n') {
            rowCount++;
          }
          c = in.read();
      }
      in.close();
      return rowCount;
    }
}
```

Client/TimeSeriesData.java

```
class TimeSeriesData {
    String header[];
```

```
    double data[][];

    public void print() {
      for(int i=0; i<header.length; i++) {
          if (i != 0) System.out.print("\t");
          System.out.print(header[i]);
      }
      System.out.println();

      for(int i=0; i<data[0].length; i++) {
          for(int j=0; j<data.length; j++) {
            if (j != 0) System.out.print("\t");
            System.out.print(data[j][i]);
          }
          System.out.println();
      }
    }
}
```

Client/ReactionException.java

```
/**
 * Exception class for reactions.
 */
public class ReactionException extends Exception {
    /**
     * Creates a new ReactionException
     * @param message
     * an error message
     */
    public ReactionException(String message) {
      super(message);
    }
}
```

Client/Reaction.java

```
import java.util.*;

/**
 * Class used to represent a chemical reaction.
 */
public class Reaction {
    /**
     * Creates a new Reaction
     */
    public Reaction() {

    }

    /**
     * Adds a reactant to the reaction.
     * @param coefficient
     * the reactant coefficient (must be greater than zero).
     * @param species
```

```
     * the name of the species (see ReactionParser for desciption of
valid species names).
     * @exception ReactionException
     * thrown if the coefficient or species in invalid.
     * @see ReactionParser
     */
    public void addReactant(int coefficient, String species) throws
ReactionException {
      if
(!ReactionParser.getReactionParser().isSpeciesNameValid(species)) {
          throw new ReactionException("Invalid species name");
      }
      if (coefficient < 1) {
          throw new ReactionException("Reactant coefficient is less
than 1");
      }
      reactants.add(new SpeciesUpdate(coefficient,species));
    }

    /**
     * Adds a product to the reaction.
     * @param coefficient
     * the product coefficient (must be greater than zero).
     * @param species
     * the name of the species (see ReactionParser for desciption of
valid species names).
     * @exception ReactionException
     * thrown if the coefficient or species in invalid.
     * @see ReactionParser
     */
    public void addProduct(int coefficient, String species) throws
ReactionException {
      if
(!ReactionParser.getReactionParser().isSpeciesNameValid(species)) {
          throw new ReactionException("Invalid species name");
      }
      if (coefficient < 1) {
          throw new ReactionException("Reactant coefficient is less
than 1");
      }
      products.add(new SpeciesUpdate(coefficient,species));
    }

    /**
     * Retrieves the number of reactants.
     * @return
     * the number of reactants.
     */
    public int getReactantCount() {
      return reactants.size();
    }

    /**
     * Retrieves the number of products.
     * @return
```

```
 * the number of products.
 */
public int getProductCount() {
  return products.size();
}

/**
 * Retrieves the coefficient of a specific reactant.
 * @param index
 * the index of the reactant.
 * @return
 * the coefficient of the reactant.
 */
public int getReactantCoefficient(int index) {
  return ((SpeciesUpdate)(reactants.get(index))).coefficient;
}

/**
 * Retrieves the species name of a specific reactant.
 * @param index
 * the index of the reactant.
 * @return
 * the species name of the reactant.
 */
public String getReactantSpecies(int index) {
  return ((SpeciesUpdate)(reactants.get(index))).species;
}

/**
 * Retrieves the coefficient of a specific product.
 * @param index
 * the index of the product.
 * @return
 * the coefficient of the product.
 */
public int getProductCoefficient(int index) {
  return ((SpeciesUpdate)(products.get(index))).coefficient;
}

/**
 * Retrieves the species name of a specific product.
 * @param index
 * the index of the product.
 * @return
 * the species name of the product.
 */
public String getProductSpecies(int index) {
  return ((SpeciesUpdate)(products.get(index))).species;
}

/**
 * Retrieves a string representation for this reaction.
 * @return
 * a string representing the reaction.
 */
```

```
    public String toString() {
      String returnval = "";
      for(int i=0; i<getReactantCount(); i++) {
          if (i != 0) {
            returnval += "+ ";
          }
          if (getReactantCoefficient(i) != 1) {
            returnval += getReactantCoefficient(i) + " ";
          }
          returnval += getReactantSpecies(i) + " ";
      }
      returnval += "=> ";
      if (getProductCount() == 0) {
          returnval += "*";
      } else {
          for(int i=0; i<getProductCount(); i++) {
            if (i != 0) {
                returnval += "+ ";
            }
            if (getProductCoefficient(i) != 1) {
                returnval += getProductCoefficient(i) + " ";
            }
            returnval += getProductSpecies(i) + " ";
          }
      }
      return returnval;
    }

    private class SpeciesUpdate {
      public String species;
      public int coefficient;

      public SpeciesUpdate(int coefficient, String species) {
          this.species = species;
          this.coefficient = coefficient;
      }
    }

    private Vector reactants = new Vector();
    private Vector products = new Vector();
}
```

Client/ModelPanel.java

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import java.util.*;
import java.lang.*;

class ModelPanel extends JPanel {
    JTextField model = new JTextField();
```

```
    SpeciesPanelTableModel speciesTableModel = new
SpeciesPanelTableModel();
    ReactionsPanelTableModel reactionsTableModel = new
ReactionsPanelTableModel();
    JTable speciesTable,reactionsTable;
    Client client;

    public class Species {
      String species = "";
      String population = "";
      Boolean output = Boolean.TRUE;
      String comment = "";
      String parameter = "";
    }

    public class Reaction {
      String reaction = "";
      String rate = "";
      String comment = "";
      String parameter = "";
    }

    public class SpeciesPanelTableModel extends AbstractTableModel {
      String columnNames[] = {"Species","Population","Parameter"};
      public Vector data = new Vector();

      public void removeAll() {
          data = new Vector();
          fireTableDataChanged();
      }

      public void addRow(String species, String population, Boolean
output, String comment, String parameter) {
          Species s = new Species();
          s.species = species;
          s.population = population;
          s.output = output;
          s.comment = comment;
          s.parameter = parameter;
          data.add(s);
          fireTableDataChanged();
      }

      public void addRow() {
          data.add(new Species());
          fireTableDataChanged();
      }

      public void removeRow(int row) {
          data.remove(row);
          fireTableDataChanged();
      }

      public Vector getData() {
          return data;
```

```
      }

      public String getColumnName(int col) { return columnNames[col]; }
      public int getColumnCount() { return columnNames.length; }
      public int getRowCount() { return data.size(); }
      public boolean isCellEditable(int row, int col) {
          if (col == 2)
            return true;
          else
            return false;
      }

      public Class getColumnClass(int col) {
          return String.class;
      }

      public Object getValueAt(int row, int col) {
          Species s = (Species)(data.get(row));
          if (col == 0) return s.species;
          else if (col == 1) return s.population;
          else return s.parameter;
      }

      public void setValueAt(Object value, int row, int col) {
          Species s = (Species)(data.get(row));
          if (col == 0) s.species = (String)value;
          else if (col == 1) s.population = (String)value;
          else s.parameter = (String)value;
      }
    }

    public class ReactionsPanelTableModel extends AbstractTableModel {
      String columnNames[] = {"Reaction","Rate","Parameter"};
      public Vector data = new Vector();

      public void removeAll() {
          data = new Vector();
          fireTableDataChanged();
      }

      public void addRow(String reaction, String rate, String comment,
String parameter) {
          Reaction r = new Reaction();
          r.reaction = reaction;
          r.rate = rate;
          r.comment = comment;
          r.parameter = parameter;
          data.add(r);
          fireTableDataChanged();
      }

      public Vector getData() {
          return data;
      }
```

```
      public void addRow() {
          data.add(new Reaction());
          fireTableDataChanged();
      }

      public void removeRow(int row) {
          data.remove(row);
          fireTableDataChanged();
      }

      public String getColumnName(int col) { return columnNames[col]; }
      public int getColumnCount() { return columnNames.length; }
      public int getRowCount() { return data.size(); }
      public boolean isCellEditable(int row, int col) {
          if (col == 2)
            return true;
          else
            return false;
      }

      public Class getColumnClass(int col) {
          return String.class;
      }

      public Object getValueAt(int row, int col) {
          Reaction r = (Reaction)(data.get(row));
          if (col == 0) return r.reaction;
          else if (col == 1) return r.rate;
          else return r.parameter;
      }

      public void setValueAt(Object value, int row, int col) {
          Reaction r = (Reaction)(data.get(row));
          if (col == 0) r.reaction = (String)value;
          else if (col == 1) r.rate = (String)value;
          else r.parameter = (String)value;
      }
    }

    public ModelPanel(Client client) {
      super();
      this.client = client;
      setLayout(new GridBagLayout());
      GridBagConstraints gbc = new GridBagConstraints();


      int speciesColumnWidths[] = {300,150,300};
      speciesTable = new JTable(speciesTableModel);
      for (int i = 0; i < speciesColumnWidths.length; i++) {

speciesTable.getColumnModel().getColumn(i).setPreferredWidth(speciesCol
umnWidths[i]);
      }

      int reactionsColumnWidths[] = {300,150,300};
```

```
      reactionsTable = new JTable(reactionsTableModel);
      for (int i = 0; i < reactionsColumnWidths.length; i++) {

reactionsTable.getColumnModel().getColumn(i).setPreferredWidth(reaction
sColumnWidths[i]);
      }

      gbc.anchor = GridBagConstraints.CENTER;
      gbc.fill = GridBagConstraints.BOTH;
      gbc.insets = new Insets(5,5,5,5);
      gbc.weightx = 0;
      gbc.weighty = 0;
      addComponent(new JLabel("Input Model:"),0,0,gbc);

      gbc.weightx = 1;
      gbc.fill = GridBagConstraints.HORIZONTAL;
      model.setEditable(false);
      addComponent(model,1,0,gbc);

      gbc.weightx = 0;
      gbc.fill = GridBagConstraints.BOTH;
      JButton browseButton = new JButton("Browse");
      browseButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) { onBrowse();
}});
      addComponent(browseButton,2,0,gbc);

      gbc.weighty = 1;
      gbc.gridwidth = 3;
      addComponent(new JScrollPane(speciesTable),0,1,gbc);
      addComponent(new JScrollPane(reactionsTable),0,2,gbc);

      speciesTable.setDefaultRenderer(String.class, new
MonospacedCellRenderer());
      reactionsTable.setDefaultRenderer(String.class, new
MonospacedCellRenderer());
      JTextField textField = new JTextField();
      textField.setFont(new Font("Monospaced",Font.PLAIN,12));
      reactionsTable.getColumnModel().getColumn(0).setCellEditor(new
DefaultCellEditor(textField));
      reactionsTable.getColumnModel().getColumn(1).setCellEditor(new
DefaultCellEditor(textField));
      reactionsTable.getColumnModel().getColumn(2).setCellEditor(new
DefaultCellEditor(textField));
      JTextField textField2 = new JTextField();
      textField2.setFont(new Font("Monospaced",Font.PLAIN,12));
      speciesTable.getColumnModel().getColumn(0).setCellEditor(new
DefaultCellEditor(textField2));
      speciesTable.getColumnModel().getColumn(1).setCellEditor(new
DefaultCellEditor(textField2));
      speciesTable.getColumnModel().getColumn(2).setCellEditor(new
DefaultCellEditor(textField2));
    }

    public void onBrowse() {
```

```
      JFileChooser fileChooser = client.getFileChooser();
      int returnval = fileChooser.showOpenDialog(this);
      if (returnval == JFileChooser.APPROVE_OPTION) {
          try {
            BioSpreadsheetData data = new BioSpreadsheetData();
            data.load(fileChooser.getSelectedFile().getAbsolutePath());
            if (data.check().size() != 0) {
                throw new IOException("Model contains errors.  Please
open this model in BioSpreadsheet and fix these errors.");
            }

      model.setText(fileChooser.getSelectedFile().getAbsolutePath());
            speciesTableModel.removeAll();
            reactionsTableModel.removeAll();
            for(int i=0; i<data.getSpeciesCount(); i++) {
                speciesTableModel.addRow(data.getSpeciesName(i),
                                         data.getSpeciesPopulation(i),
                                         data.getSpeciesOutput(i),
                                         data.getSpeciesComment(i),
                                         "");
            }
            for(int i=0; i<data.getReactionCount(); i++) {
                reactionsTableModel.addRow(data.getReaction(i),
                                           data.getReactionRate(i),
                                           data.getReactionComment(i),
                                           "");
            }
          } catch (IOException e) {
            JOptionPane.showMessageDialog(this,
                                          "Error reading file: " +
e.getMessage(),
                                          "BioGrid Client
v1.0",JOptionPane.ERROR_MESSAGE);
          }
      }
    }

    private void addComponent(Component c, int x, int y,
GridBagConstraints gbc) {
      gbc.gridx = x;
      gbc.gridy = y;
      add(c,gbc);
    }

    public void stopEditing() {
      speciesTable.editCellAt(-1,-1);
      reactionsTable.editCellAt(-1,-1);
    }

    public void clearAll() {
      speciesTableModel.removeAll();
      reactionsTableModel.removeAll();
    }

    public void setData(ClientData data) {
```

```
      data.modelFileName = model.getText();
      data.speciesParameters = new
String[speciesTableModel.getRowCount()];
      for(int i=0; i<data.speciesParameters.length; i++) {
          data.speciesParameters[i] =
speciesTableModel.getValueAt(i,2).toString().trim();
      }
      data.reactionParameters = new
String[reactionsTableModel.getRowCount()];
      for(int i=0; i<data.reactionParameters.length; i++) {
          data.reactionParameters[i] =
reactionsTableModel.getValueAt(i,2).toString().trim();
      }
    }
}
```

Client/BioSpreadsheetData.java

```
import java.util.Vector;
import java.io.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Document;
import org.w3c.dom.DOMException;

class BioSpreadsheetData {
    Vector speciesNames = new Vector();
    Vector speciesPopulations = new Vector();
    Vector speciesComments = new Vector();
    Vector speciesOutputs = new Vector();
    Vector reactions = new Vector();
    Vector reactionRates = new Vector();
    Vector reactionComments = new Vector();
    String name;
    String title;
    String author;
    String location;
    String description;

    public void addSpecies(String name, String population, Boolean
output, String comment) {
        speciesNames.add(name);
      speciesPopulations.add(population);
      speciesOutputs.add(output);
      speciesComments.add(comment);
    }
```

```
    public void addReaction(String reaction, String rate, String
comment) {
      reactions.add(reaction);
      reactionRates.add(rate);
      reactionComments.add(comment);
    }

    public void setTitle(String title) {
      this.title = title;
    }

    public void setName(String name) {
      this.name = name;
    }

    public void setAuthor(String author) {
      this.author = author;
    }

    public void setLocation(String location) {
      this.location = location;
    }

    public void setDescription(String description) {
      this.description = description;
    }

    public int getSpeciesCount() {
      return speciesNames.size();
    }

    public int getReactionCount() {
      return reactions.size();
    }

    public void setSpeciesPopulation(int index, String value) {
      speciesPopulations.set(index,value);
    }

    public void setReactionRate(int index, String value) {
      reactionRates.set(index,value);
    }

    public String getSpeciesName(int index) {
      return (String)speciesNames.get(index);
    }

    public String getSpeciesPopulation(int index) {
      return (String)speciesPopulations.get(index);
    }

    public Boolean getSpeciesOutput(int index) {
      return (Boolean)speciesOutputs.get(index);
    }
```

```
    public String getSpeciesComment(int index) {
      return (String)speciesComments.get(index);
    }

    public String getReaction(int index) {
      return (String)reactions.get(index);
    }

    public String getReactionRate(int index) {
      return (String)reactionRates.get(index);
    }

    public String getReactionComment(int index) {
      return (String)reactionComments.get(index);
    }

    public String getTitle() {
      return title;
    }

    public String getName() {
      return name;
    }

    public String getAuthor() {
      return author;
    }

    public String getLocation() {
      return location;
    }

    public String getDescription() {
      return description;
    }

    public Vector check() {
      Vector errors = new Vector();
      ReactionParser reactionParser =
ReactionParser.getReactionParser();

      if (reactions.size() == 0) {
          errors.add("At least one reaction must exist.");
      }

      for(int i=0; i<speciesNames.size(); i++) {
          if
(!reactionParser.isSpeciesNameValid((String)speciesNames.get(i))) {
            errors.add("Invalid species name \"" + speciesNames.get(i)
+ "\".");
          }
          try {
            int population =
Integer.parseInt((String)speciesPopulations.get(i));
            if (population < 0) {
```

```
                errors.add("Invalid species population \"" +
speciesPopulations.get(i) + "\".  Species populations must be non-
negative integers.");
              }
            } catch (NumberFormatException e) {
              errors.add("Invalid species population \"" +
speciesPopulations.get(i) + "\".  Species populations must be non-
negative integers.");
            }
        }

        String speciesNameArray[] = new String[speciesNames.size()];
        for(int i=0; i<speciesNames.size(); i++) {
            speciesNameArray[i] = (String)speciesNames.get(i);
        }

        for(int i=0; i<reactions.size(); i++) {
            try {

        reactionParser.parse((String)reactions.get(i),speciesNameArray);
            } catch (ReactionParserException e) {
              String errorMessage = "Invalid Reaction: ";
              errorMessage += e.getReaction() + "\n";
              for(int j=0; j<e.getLocation(); j++) {
                  errorMessage += " ";
              }
              errorMessage += "                   ^ " + e.getMessage();
              errors.add(errorMessage);
            }
            try {
              double rate =
Double.parseDouble((String)reactionRates.get(i));
              if (rate < 0) {
                  errors.add("Invalid reaction rate \"" +
reactionRates.get(i) + "\".  Reaction rates must be non-negative real
numbers.");
              }
            } catch (NumberFormatException e) {
              errors.add("Invalid reaction rate \"" +
reactionRates.get(i) + "\".  Reaction rates must be non-negative real
numbers.");
            }
        }

        return errors;
    }

    public String fix(Object in) {
      return in.toString().replace('\"','\'');
    }

    public void save(String filename) throws IOException {
      PrintStream out = new PrintStream(new
FileOutputStream(filename));
      out.println("<?xml version=\"1.0\" encoding=\"UTF-8\" ?>");
```

```
      out.println("<sbml xmlns=\"http://www.sbml.org/sbml/level2\"
level=\"2\" version=\"1\">");
      out.println("<model id=\"" + fix(name) + "\">");
      out.println("<annotation
xmlns:biospreadsheet=\"http://biocomp.ece.utk.edu/ns\">");
      out.println("<biospreadsheet:model author=\"" + fix(author) + "\"
title=\"" + fix(title) + "\" " +
                "location=\"" + fix(location) + "\" description=\"" +
fix(description) + "\"/>");
      out.println("</annotation>");
      out.println("<listOfCompartments>");
      out.println("<compartment id=\"cell\"/>");
      out.println("</listOfCompartments>");
      out.println("<listOfSpecies>");
      for(int i=0; i<speciesNames.size(); i++) {
          out.println("<species " +
                  "id=\"" + fix(speciesNames.get(i)) + "\" " +
                  "initialAmount=\"" + fix(speciesPopulations.get(i)) +
"\" " +
                  "compartment=\"cell\">");
          out.println("<annotation
xmlns:biospreadsheet=\"http://biocomp.ece.utk.edu/ns\">");
          out.println("<biospreadsheet:species " +
                  "output=\"" + fix(speciesOutputs.get(i)) + "\" " +
                  "comment=\"" + fix(speciesComments.get(i)) + "\"/>");
          out.println("</annotation>");
          out.println("</species>");
      }
      out.println("</listOfSpecies>");
      out.println("<listOfReactions>");
      for(int i=0; i<reactions.size(); i++) {
          out.println("<reaction id=\"reaction" + i + "\"
reversible=\"false\">");
          out.println("<annotation
xmlns:biospreadsheet=\"http://biocomp.ece.utk.edu/ns\">");
          out.println("<biospreadsheet:reaction " +
                  "reaction=\"" + fix(reactions.get(i)) + "\" " +
                  "rate=\"" + fix(reactionRates.get(i)) + "\" " +
                  "comment=\"" + fix(reactionComments.get(i)) +
"\"/>");
          out.println("</annotation>");
          try {
            Reaction r =
ReactionParser.getReactionParser().parse((String)reactions.get(i));
            out.println("<listOfReactants>");
            for(int j=0; j<r.getReactantCount(); j++) {
                out.println("<speciesReference " +
                        "species=\"" + fix(r.getReactantSpecies(j)) +
"\" " +
                        "stochiometry=\"" + r.getReactantCoefficient(j)
+ "\"/>");
            }
            out.println("</listOfReactants>");
            out.println("<listOfProducts>");
            for(int j=0; j<r.getProductCount(); j++) {
```

```
                out.println("<speciesReference " +
                        "species=\"" + fix(r.getProductSpecies(j)) +
"\" " +
                        "stochiometry=\"" + r.getProductCoefficient(j)
+ "\"/>");
            }
            out.println("</listOfProducts>");
            out.println("<kineticLaw>");
            out.println("<math
xmlns=\"http://www.w3.org/1998/Math/MathML\">");
            for(int j=0; j<r.getReactantCount()-1; j++) {
                out.println("<math:apply><math:add/>");
            }
            for(int j=0; j<r.getReactantCount(); j++) {
                int coefficient = r.getReactantCoefficient(j);
                String name = r.getReactantSpecies(j);
                out.println("<math:apply><math:divide/>");
                out.println("<math:apply><math:divide/>");
                out.println("<math:apply><math:factorial/>");
                out.println("<math:ci>" + name + "</math:ci>");
                out.println("</math:apply>");
                out.println("<math:apply><math:factorial/>");
                out.println("<math:cn>" + coefficient + "</math:cn>");
                out.println("</math:apply>");
                out.println("</math:apply>");
                out.println("<math:apply><math:factorial/>");
                out.println("<math:apply><math:minus/>");
                out.println("<math:ci>" + name + "</math:ci>");
                out.println("<math:cn>" + coefficient + "</math:cn>");
                out.println("</math:apply>");
                out.println("</math:apply>");
                out.println("</math:apply>");
            }
            for(int j=0; j<r.getReactantCount()-1; j++) {
                out.println("</math:apply>");
            }
            out.println("</math>");
            out.println("</kineticLaw>");
          } catch (ReactionParserException e) {
            out.println("<listOfReactants>");
            out.println("</listOfReactants>");
            out.println("<listOfProducts>");
            out.println("</listOfProducts>");
            out.println("<kineticLaw>");
            out.println("</kineticLaw>");
          }
          out.println("</reaction>");
      }
      out.println("</listOfReactions>");
      out.println("</model>");
      out.println("</sbml>");
      out.close();
    }

    public void load(String filename) throws IOException {
```

```
      speciesNames = new Vector();
      speciesPopulations = new Vector();
      speciesComments = new Vector();
      speciesOutputs = new Vector();
      reactions = new Vector();
      reactionRates = new Vector();
      reactionComments = new Vector();
      name = "";
      title = "";
      author = "";
      location = "";
      description = "";

      DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
      Document document;
        try {
          DocumentBuilder builder = factory.newDocumentBuilder();
          document = builder.parse(new File(filename));
        } catch (SAXParseException spe) {
          throw new IOException("Unable to read " + filename + "
because there was an error parsing the file on line " +
spe.getLineNumber() + ": " + spe.getMessage());
        } catch (SAXException sxe) {
          throw new IOException("Unable to read " + filename + "
because there was an error parsing the file: " + sxe.getMessage());
        } catch (ParserConfigurationException pce) {
          throw new IOException("Unable to read " + filename + "
because the XML parser could not be configured: " + pce.getMessage());
        } catch (IOException ioe) {
          throw new IOException("Unable to read " + filename + "
because: " + ioe.getMessage());
        }

      Vector modelVector = findNodes(document,"model");
      if (modelVector.size() != 1) return;
      Node modelNode = (Node)modelVector.elementAt(0);
      if (modelNode.hasAttributes()) {
          NamedNodeMap attributes = modelNode.getAttributes();
          name = getItemFromAttributes(attributes,"id");
      }

      Vector bssModelVector =
findNodes(document,"biospreadsheet:model");
      if (bssModelVector.size() == 1) {
          Node bssModelNode = (Node)bssModelVector.elementAt(0);
          if (bssModelNode.hasAttributes()) {
            NamedNodeMap attributes = bssModelNode.getAttributes();
            title = getItemFromAttributes(attributes,"title");
            author = getItemFromAttributes(attributes,"author");
            location = getItemFromAttributes(attributes,"location");
            description =
getItemFromAttributes(attributes,"description");
          }
      }
```

```
      Vector listOfSpeciesNodes = findNodes(modelNode,"listOfSpecies");
      if (listOfSpeciesNodes.size() == 1) {
          Node listOfSpeciesNode = (Node)listOfSpeciesNodes.get(0);
          Vector speciesNodes = findNodes(listOfSpeciesNode,"species");
          for(int i=0; i<speciesNodes.size(); i++) {
            String name = "";
            String population = "";
            Boolean output = Boolean.TRUE;
            String comment = "";
            Node speciesNode = (Node)speciesNodes.elementAt(i);
            if (speciesNode.hasAttributes()) {
                NamedNodeMap attributes = speciesNode.getAttributes();
                name = getItemFromAttributes(attributes,"id");
                population =
getItemFromAttributes(attributes,"initialAmount");
            }
            Vector listOfBssSpeciesNodes =
findNodes(speciesNode,"biospreadsheet:species");
            if (listOfBssSpeciesNodes.size() == 1) {
                NamedNodeMap attributes =
((Node)listOfBssSpeciesNodes.get(0)).getAttributes();
                if
(getItemFromAttributes(attributes,"output").equals("false")) {
                  output = Boolean.FALSE;
                }
                comment = getItemFromAttributes(attributes,"comment");
            }
            addSpecies(name,population,output,comment);
          }
      }

      Vector listOfReactionsNodes =
findNodes(modelNode,"listOfReactions");
      if (listOfReactionsNodes.size() == 1) {
          Node listOfReactionsNode = (Node)listOfReactionsNodes.get(0);
          Vector reactionNodes =
findNodes(listOfReactionsNode,"reaction");
          for(int i=0; i<reactionNodes.size(); i++) {
            String reaction = "";
            String rate = "";
            String comment = "";
            Node reactionNode = (Node)reactionNodes.elementAt(i);
            Vector listOfBssReactionNodes =
findNodes(reactionNode,"biospreadsheet:reaction");
            if (listOfBssReactionNodes.size() == 1) {
                NamedNodeMap attributes =
((Node)listOfBssReactionNodes.get(0)).getAttributes();
                reaction =
getItemFromAttributes(attributes,"reaction");
                rate = getItemFromAttributes(attributes,"rate");
                comment = getItemFromAttributes(attributes,"comment");

            } else {
                reaction = getReactionString(reactionNode);
```

```
            }
            addReaction(reaction,rate,comment);
          }
      }
    }

    private String getItemFromAttributes(NamedNodeMap map, String name)
{
      if (map.getNamedItem(name) != null) {
          return map.getNamedItem(name).getNodeValue();
      }
      return "";
    }

    private String getReactionString(Node n) {
      String s = "";
      Vector listOfReactantsNodes = findNodes(n,"listOfReactants");
      int reactantCount = 0;
      for(int i=0; i<listOfReactantsNodes.size(); i++) {
          Node listOfReactantsNode =
(Node)listOfReactantsNodes.elementAt(i);
          Vector speciesReferences =
findNodes(listOfReactantsNode,"speciesReference");
          for(int j=0; j<speciesReferences.size(); j++) {
            Node speciesReference =
(Node)speciesReferences.elementAt(j);
            if (speciesReference.hasAttributes()) {
                NamedNodeMap attributes =
speciesReference.getAttributes();
                String speciesName =
getItemFromAttributes(attributes,"species");
                String coefficient =
getItemFromAttributes(attributes,"stoichiometry");
                if (!speciesName.equals("")) {
                  if (reactantCount != 0) {
                      s += "+ ";
                  }
                  if (!(coefficient.equals("") ||
coefficient.equals("1"))) {
                      s += coefficient + " ";
                  }
                  s += speciesName + " ";
                  reactantCount++;
                }
            }
          }
      }
      s += "-> ";
      Vector listOfProductsNodes = findNodes(n,"listOfProducts");
      int productCount = 0;
      for(int i=0; i<listOfProductsNodes.size(); i++) {
          Node listOfProductsNode =
(Node)listOfProductsNodes.elementAt(i);
          Vector speciesReferences =
findNodes(listOfProductsNode,"speciesReference");
```

```
          for(int j=0; j<speciesReferences.size(); j++) {
            Node speciesReference =
(Node)speciesReferences.elementAt(j);
            if (speciesReference.hasAttributes()) {
                NamedNodeMap attributes =
speciesReference.getAttributes();
                String speciesName =
getItemFromAttributes(attributes,"species");
                if (speciesName.equals("")) {
                  speciesName =
getItemFromAttributes(attributes,"specie");
                }
                String coefficient =
getItemFromAttributes(attributes,"stoichiometry");
                if (!speciesName.equals("")) {
                  if (productCount != 0) {
                      s += "+ ";
                  }
                  if (!(coefficient.equals("") ||
coefficient.equals("1"))) {
                      s += coefficient + " ";
                  }
                  s += speciesName + " ";
                  productCount++;
                }
            }
          }
      }
      if (productCount == 0) {
          s += "*";
      }
      return s.trim();
    }

    private Vector findNodes(Node n, String name) {
      Vector v = new Vector();
      if (n.getNodeName().equals(name)) {
          v.add(n);
      } else if (n.getNodeName().equals(name)) {
          v.add(n);
      } else if (n.hasChildNodes()) {
          NodeList nl = n.getChildNodes();
          for(int i=0; i<nl.getLength(); i++) {
            Vector v2 = findNodes(nl.item(i),name);
            for(int j=0; j<v2.size(); j++) {
                v.add(v2.elementAt(j));
            }
          }
      }
      return v;
    }

    public int getOutputCount() {
      int outputCount = 0;
      for(int i=0; i<getSpeciesCount(); i++) {
```

```
            if (getSpeciesOutput(i) == Boolean.TRUE) outputCount++;
        }
        return outputCount;
    }

    public void writeEssFile(String filename) throws IOException {
        PrintStream out = new PrintStream(new
FileOutputStream(filename));
        out.println(getSpeciesCount());
        for(int i=0; i<getSpeciesCount(); i++) {
            if (i != 0) out.print(" ");
            out.print(getSpeciesPopulation(i));
        }
        out.println();
        out.println(getReactionCount());
        for(int i=0; i<getReactionCount(); i++) {
            String reactionString = getReaction(i);
            try {
              Reaction r =
ReactionParser.getReactionParser().parse(reactionString);
              out.print("" + r.getReactantCount() + " ");
              for(int j=0; j<r.getReactantCount(); j++) {
                  out.print("" + r.getReactantCoefficient(j) + " ");
                  out.print("" + getSpeciesIndex(r.getReactantSpecies(j))
+ " ");
              }
              out.print("" + r.getProductCount() + " ");
              for(int j=0; j<r.getProductCount(); j++) {
                  out.print("" + r.getProductCoefficient(j) + " ");
                  out.print("" + getSpeciesIndex(r.getProductSpecies(j))
+ " ");
              }
              out.print("" + getReactionRate(i));
              out.println();
            } catch (ReactionParserException e) {
              System.out.println("This should never happen
(writeEssFile).");
              System.exit(1);
            }
        }
        out.println(getOutputCount());
        for(int i=0; i<getSpeciesCount(); i++) {
            if (getSpeciesOutput(i) == Boolean.TRUE) out.print("" + i + "
");
        }
        out.println();
        out.close();
    }

    public int getSpeciesIndex(String name) {
        for(int i=0; i<getSpeciesCount(); i++) {
            if (getSpeciesName(i).equals(name)) {
              return i;
            }
        }
```

```
      return -1;
    }
}
```

Client/PostProcessingPanel.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import java.util.*;
import java.lang.*;

class PostProcessingPanel extends JPanel {
    PostProcessingPanelTableModel tableModel = new
PostProcessingPanelTableModel();
    JTable table;
    Client client;

    public class Function {
      String name = "";
      String expression = "";
    }

    public class PostProcessingPanelTableModel extends
AbstractTableModel {
      String columnNames[] = {"Output Name","Octave Expression"};
      public Vector data = new Vector();

      PostProcessingPanelTableModel() {
          removeAll();
      }

      public void removeAll() {
          data = new Vector();
          fireTableDataChanged();
      }

      public void addRow(String name, String expression) {
          Function f = new Function();
          f.name = name;
          f.expression = expression;
          data.add(f);
          fireTableDataChanged();
      }

      public void addRow() {
          data.add(new Function());
          fireTableDataChanged();
      }

      public void removeRow(int row) {
          data.remove(row);
          fireTableDataChanged();
      }
```

```
      public Vector getData() {
          return data;
      }

      public String getColumnName(int col) { return columnNames[col]; }
      public int getColumnCount() { return columnNames.length; }
      public int getRowCount() { return data.size(); }
      public boolean isCellEditable(int row, int col) {
          return true;
      }

      public Class getColumnClass(int col) {
          return String.class;
      }

      public Object getValueAt(int row, int col) {
          Function f = (Function)(data.get(row));
          if (col == 0) return f.name;
          else return f.expression;
      }

      public void setValueAt(Object value, int row, int col) {
          Function f = (Function)(data.get(row));
          if (col == 0) f.name = (String)value;
          else f.expression = (String)value;
      }
    }

    public PostProcessingPanel(Client client) {
      super();
      this.client = client;
      setLayout(new GridBagLayout());
      GridBagConstraints gbc = new GridBagConstraints();

      int columnWidths[] = {200,600};
      table = new JTable(tableModel);
      for (int i = 0; i < columnWidths.length; i++) {

table.getColumnModel().getColumn(i).setPreferredWidth(columnWidths[i]);
      }

      JButton addRowButton = new JButton("Add Function");
      JButton removeRowButton = new JButton("Remove Function");
      addRowButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) { onAddRow();
}});
      removeRowButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) { onRemoveRow();
}});

      gbc.anchor = GridBagConstraints.NORTHEAST;
      gbc.fill = GridBagConstraints.BOTH;
      gbc.insets = new Insets(5,5,5,5);
      gbc.weightx = 1;
```

```
      gbc.weighty = 1;
      gbc.gridwidth = 3;
      addComponent(new JScrollPane(table),0,0,gbc);
      gbc.gridwidth = 1;
      gbc.weighty = 0;
      addComponent(new JPanel(),0,1,gbc);
      gbc.anchor = GridBagConstraints.CENTER;
      gbc.fill = GridBagConstraints.NONE;
      gbc.weightx = 0;
      addComponent(addRowButton,1,1,gbc);
      addComponent(removeRowButton,2,1,gbc);

      table.setDefaultRenderer(String.class, new
MonospacedCellRenderer());
      JTextField textField = new JTextField();
      textField.setFont(new Font("Monospaced",Font.PLAIN,12));
      table.getColumnModel().getColumn(0).setCellEditor(new
DefaultCellEditor(textField));
      table.getColumnModel().getColumn(1).setCellEditor(new
DefaultCellEditor(textField));
    }

    private void onAddRow() {
      tableModel.addRow();
    }

    private void onRemoveRow() {
      int row = table.getSelectedRow();
      if (row != -1) tableModel.removeRow(row);
    }

    private void addComponent(Component c, int x, int y,
GridBagConstraints gbc) {
      gbc.gridx = x;
      gbc.gridy = y;
      add(c,gbc);
    }

    public void stopEditing() {
      table.editCellAt(-1,-1);
    }

    public void clearAll() {
      tableModel.removeAll();
    }

    public void setData(ClientData data) {
      data.functionNames = new String[tableModel.getRowCount()];
      data.functionExpressions = new String[tableModel.getRowCount()];
      for(int i=0; i<tableModel.getRowCount(); i++) {
          data.functionNames[i] =
tableModel.getValueAt(i,0).toString().trim();
          data.functionExpressions[i] =
tableModel.getValueAt(i,1).toString().trim();
      }
```

```
    }
}
```

Client/ErrorWindow.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class ErrorWindow extends JFrame {
    JTextArea textArea = new JTextArea();

    public ErrorWindow() {
      super("Error Browser");
      textArea.setEditable(false);
      textArea.setLineWrap(true);
      textArea.setWrapStyleWord(true);
      textArea.setFont(new Font("Monospaced",Font.PLAIN,12));
      getContentPane().add(new JScrollPane(textArea));
      setSize(400,400);
    }

    public void setText(String text) {
      textArea.setText(text);
    }
}
```

Client/RunWindow.java

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import java.util.*;
import java.lang.*;

class RunWindow extends JDialog {
    RunWindowTableModel tableModel;

    public class RunWindowTableModel extends AbstractTableModel {
      String parameters[][];
      String parameterNames[];
      String functionNames[];
      String outputFiles[];
      String sbmlFiles[];
      String functionFiles[][];

      public RunWindowTableModel(int rows,
                          String parameters[][],
                          String parameterNames[],
                          String functionNames[]) {
          this.parameters = parameters;
          this.parameterNames = parameterNames;
          this.functionNames = functionNames;
```

```
    outputFiles = new String[rows];
    sbmlFiles = new String[rows];
    functionFiles = new String[functionNames.length][];
    for(int j=0; j<functionNames.length; j++) {
      functionFiles[j] = new String[rows];
    }
    for(int i=0; i<rows; i++) {
      outputFiles[i] = "Pending";
      sbmlFiles[i] = "" + (i+1) + ".sbml";
      for(int j=0; j<functionNames.length; j++) {
          functionFiles[j][i] = "Pending";
      }
    }
}

public void setOutputStatus(int output, String value) {
    outputFiles[output] = value;
    fireTableDataChanged();
}

public void setFunctionStatus(int row, int col, String value) {
    functionFiles[col][row] = value;
    fireTableDataChanged();
}

public String getColumnName(int col) {
    if (col == 0) {
      return "Model File";
    }
    col -= 1;
    if (col < parameterNames.length) {
      return parameterNames[col];
    }
    col -= parameterNames.length;
    if (col == 0) {
      return "Output File";
    }
    col -= 1;
    if (col < functionNames.length) {
      return functionNames[col];
    }
    return "";
}

public int getColumnCount() {
    return parameterNames.length+2+functionNames.length;
}

public int getRowCount() {
    return sbmlFiles.length;
}

public boolean isCellEditable(int row, int col) {
    return false;
```

```
      }

      public Class getColumnClass(int col) {
          return String.class;
      }

      public Object getValueAt(int row, int col) {
          if (col == 0) {
            return sbmlFiles[row];
          }
          col -= 1;
          if (col < parameterNames.length) {
            return parameters[col][row];
          }
          col -= parameterNames.length;
          if (col == 0) {
            return outputFiles[row];
          }
          col -= 1;
          return functionFiles[col][row];
      }

      public void setValueAt(Object value, int row, int col) {

      }
    }

    public void setOutputStatus(int output, String value) {
      tableModel.setOutputStatus(output,value);
    }

    public void setFunctionStatus(int row, int col, String value) {
      tableModel.setFunctionStatus(row,col,value);
    }

    public RunWindow(Frame parent,
                 int rows,
                 String parameters[][],
                 String parameterNames[],
                 String functionNames[]) {
      super(parent,"Run Window",true);
      tableModel = new RunWindowTableModel(rows,
                                   parameters,
                                   parameterNames,
                                   functionNames);
      JTable table = new JTable(tableModel);
      setSize(600,400);
      getContentPane().add(new JScrollPane(table));
      setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
    }
}
```

OctaveWorker/DataFile.java

```
import java.io.*;
```

```
class DataFile {
    public static void write(TimeSeriesData tsd, String filename)
throws IOException {
      DataOutputStream out = new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(filename)));
      out.writeInt(13579);
      out.writeInt(97531);
      int rows = tsd.data[0].length;
      int columns = tsd.header.length;
      out.writeInt(rows);
      out.writeInt(columns);
      for(int row=0; row < rows; row++) {
          for(int col=0; col<columns; col++) {
            out.writeDouble(tsd.data[col][row]);
          }
      }
      for(int i=0; i<tsd.header.length; i++) {
          String columnName = tsd.header[i];
          out.writeInt(columnName.length());
          out.writeChars(columnName);
      }
      out.close();
    }

    public static TimeSeriesData read(String filename) throws
IOException {
      DataInputStream in = new DataInputStream(new
BufferedInputStream(new FileInputStream(filename)));
      if (in.readInt() != 13579) throw new IOException("Invalid file
format.");
      if (in.readInt() != 97531) throw new IOException("Invalid file
format.");
      int rows = in.readInt();
      int columns = in.readInt();
      TimeSeriesData tsd = new TimeSeriesData();
      tsd.data = new double[columns][];
      for(int i=0; i<columns; i++) {
          tsd.data[i] = new double[rows];
      }
      for(int row=0; row<tsd.data[0].length; row++) {
          for(int col=0; col<tsd.data.length; col++) {
            tsd.data[col][row] = in.readDouble();
          }
      }
      tsd.header = new String[columns];
      for(int i=0; i<columns; i++) {
          tsd.header[i] = "";
          int length = in.readInt();
          for(int j=0; j<length; j++) {
            tsd.header[i] += in.readChar();
          }
      }
      return tsd;
    }
```

```
    public static void sample(String inputFilename, String
outputFilename, int rate) throws IOException {
      DataInputStream in = new DataInputStream(new
BufferedInputStream(new FileInputStream(inputFilename)));
      if (in.readInt() != 13579) throw new IOException("Invalid file
format.");
      if (in.readInt() != 97531) throw new IOException("Invalid file
format.");
      DataOutputStream out = new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(outputFilename)));

      int in_rows = in.readInt();
      int in_cols = in.readInt();

      int out_rows = in_rows / rate;

      out.writeInt(13579);
      out.writeInt(97531);
      out.writeInt(out_rows);
      out.writeInt(in_cols);

      for(int row=0; row<out_rows; row++) {
          for(int col=0; col<in_cols; col++) {
            out.writeDouble(in.readDouble());
          }
          in.skipBytes(8*in_cols*(rate-1));
      }
      in.skipBytes(8*in_cols*(in_rows % rate));

      int i;
      while((i = in.read()) != -1) {
          out.write(i);
      }
      in.close();
      out.close();
    }

    public static void tsd2tab(String inputFile, String outputFile)
throws IOException {
      TABFile.write(TSDFile.read(inputFile),outputFile);
    }

    public static void tsd2bsd(String inputFile, String outputFile)
throws IOException {
      write(TSDFile.read(inputFile),outputFile);
    }

    public static void bsd2tsd(String inputFile, String outputFile)
throws IOException {
      TSDFile.write(read(inputFile),outputFile);
    }

    public static void bsd2tab(String inputFile, String outputFile)
throws IOException {
```

```
      TABFile.write(read(inputFile),outputFile);
    }

    public static void tab2bsd(String inputFile, String outputFile)
throws IOException {
      write(TABFile.read(inputFile),outputFile);
    }

    public static void tab2tsd(String inputFile, String outputFile)
throws IOException {
      TSDFile.write(TABFile.read(inputFile),outputFile);
    }

    public static void merge(String inputFileA, String inputFileB,
                        String prefixA, String prefixB,
                        String outputFile, boolean zeroPad) throws
IOException {
      DataInputStream inA = new DataInputStream(new
BufferedInputStream(new FileInputStream(inputFileA)));
      DataInputStream inB = new DataInputStream(new
BufferedInputStream(new FileInputStream(inputFileB)));
      if (inA.readInt() != 13579) throw new IOException("Invalid file
format.");
      if (inA.readInt() != 97531) throw new IOException("Invalid file
format.");
      if (inB.readInt() != 13579) throw new IOException("Invalid file
format.");
      if (inB.readInt() != 97531) throw new IOException("Invalid file
format.");
      DataOutputStream out = new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(outputFile)));
      int inArows = inA.readInt();
      int inAcols = inA.readInt();
      int inBrows = inB.readInt();
      int inBcols = inB.readInt();
      int newrows = inArows;
      if (zeroPad) {
          if (inArows < inBrows) newrows = inBrows;
      } else {
          if (inArows > inBrows) newrows = inBrows;
      }
      int newcols = inAcols + inBcols;
      out.writeInt(13579);
      out.writeInt(97531);
      out.writeInt(newrows);
      out.writeInt(newcols);
      for(int row=0; row<newrows; row++) {
          for(int col=0; col<inAcols; col++) {
            if (row<inArows) {
                out.writeDouble(inA.readDouble());
            } else {
                out.writeDouble(0.0);
            }
          }
          for(int col=0; col<inBcols; col++) {
```

```
            if (row<inBrows) {
                out.writeDouble(inB.readDouble());
            } else {
                out.writeDouble(0.0);
            }
          }
      }

      if (!zeroPad) {
          if (newrows == inArows) {
            inB.skipBytes(8*inBcols*(inBrows-newrows));
          } else {
            inA.skipBytes(8*inAcols*(inArows-newrows));
          }
      }

      for(int i=0; i<inAcols; i++) {
          int strlen = inA.readInt();
          String header = "" + prefixA;
          for(int j=0; j<strlen; j++) {
            header += inA.readChar();
          }
          out.writeInt(header.length());
          out.writeChars(header);
      }
      for(int i=0; i<inBcols; i++) {
          int strlen = inB.readInt();
          String header = "" + prefixB;
          for(int j=0; j<strlen; j++) {
            header += inB.readChar();
          }
          out.writeInt(header.length());
          out.writeChars(header);
      }
      out.close();
      inA.close();
      inB.close();
    }
}
```

OctaveWorker/TSDFile.java

```
import java.io.*;
import java.util.*;

class TSDFile {
    public static TimeSeriesData read(String filename)
      throws IOException {

      TimeSeriesData tsd = new TimeSeriesData();

      int columnCount = getColumnCount(filename);
      int rowCount = getRowCount(filename);
      if ((columnCount < 0) || (rowCount < 0)) {
          throw new IOException("Bad time series data file format.");
```

```
      }

      tsd.header = new String[columnCount];
      tsd.data = new double[columnCount][];
      for(int i=0; i<columnCount; i++) {
          tsd.data[i] = new double[rowCount];
      }

      BufferedReader in = new BufferedReader(new FileReader(filename));

      in.read();
      if (columnCount > 1) {
          in.read();
      }

      for(int i=0; i<columnCount; i++) {
          tsd.header[i] = getNextString(in);
      }
      for(int i=0; i<rowCount; i++) {
          for(int j=0; j<columnCount; j++) {
            tsd.data[j][i] = getNextDouble(in);
          }
      }

      in.close();
      return tsd;
    }

    public static void write(TimeSeriesData tsd, String filename)
      throws IOException {
      PrintStream out = new PrintStream(new
FileOutputStream(filename));
      if (tsd.header.length > 1) {
          out.print("((");
          for(int i=0; i<tsd.header.length; i++) {
            if (i != 0) out.print(",");
            out.print("\"");
            out.print(tsd.header[i]);
            out.print("\"");
          }
          for(int i=0; i<tsd.data[0].length; i++) {
            out.print("),(");
            for(int j=0; j<tsd.data.length; j++) {
                if (j != 0) out.print(",");
                out.print(tsd.data[j][i]);
            }
          }
          out.print("))");
      } else {
          out.print("(\"");
          out.print(tsd.header[0]);
          out.print("\"");
          for(int i=0; i<tsd.data[0].length; i++) {
            out.print(",");
            out.print(tsd.data[0][i]);
```

```
            }
            out.print(")");
        }
        out.close();
    }

    private static int getColumnCount(String filename) throws
IOException {
        BufferedReader in = new BufferedReader(new FileReader(filename));
        int c = in.read();
        c = in.read();
        if (c == '(') {
            int columnCount = 1;
            while ((c != -1) && (c != ')')) {
              if (c == ',') columnCount++;
              c = in.read();
            }
            return columnCount;
        } else {
            return 1;
        }
    }

    private static int getRowCount(String filename) throws IOException
{
        BufferedReader in = new BufferedReader(new FileReader(filename));
        int c = in.read();
        c = in.read();
        if (c == '(') {
            int rowCount = -2;
            while (c != -1) {
              if (c == ')') {
                  rowCount++;
              }
              c = in.read();
            }
            return rowCount;
        } else {
            int rowCount = 0;
            while (c != -1) {
              if (c == ',') {
                  rowCount++;
              }
              c = in.read();
            }
            return rowCount;
        }
    }

    private static double getNextDouble(BufferedReader in) throws
IOException {
        String value = "";
        int c = in.read();
        if (c == -1) {
            throw new IOException("End of file.");
```

```
      }
      while ((c != ')') && (c != ',') && (c != -1)) {
          value += (char)c;
          c = in.read();
      }
      try {
          if (c == ')') {
            in.read();
            in.read();
          }
          return Double.parseDouble(value);
      } catch (NumberFormatException e) {
          throw new IOException("Invalid number: " + value);
      }
    }

    public static String getNextString(BufferedReader in) throws
IOException {
      String value = "";
      int c = in.read();
      if (c != '"') throw new IOException("String expected.");
      c = in.read();
      while((c != -1) && (c != '"')) {
          value += (char)c;
          c = in.read();
      }
      c = in.read();
      if (c == ')') {
          in.read();
          in.read();
      }
      return value;
    }
}
```

OctaveWorker/OctaveWorker.java

```
import java.io.*;

class OctaveWorker {
    public static void printUsage() {
      System.out.println("usage: java OctaveWorker <host> <port>");
      System.exit(1);
    }

    public static void main(String args[]) {
      if (args.length != 2) {
          printUsage();
      }
      String host = args[0];
      int port = 0;
      try {
          port = Integer.parseInt(args[1]);
      } catch (NumberFormatException e) {
          printUsage();
```

```
      }

      System.out.println();
      System.out.println("BioGrid Octave Worker v1.0");
      System.out.println("written by James M. McCollum
(jmccoll2@utk.edu)");
      System.out.println("University of Tennessee - Knoxville");
      System.out.println("http://biocomp.ece.utk.edu");
      System.out.println();

      if (System.getProperty("octave.path") == null) {
          System.out.println("octave.path variable not set.");
          System.exit(1);
      }

      Worker worker = new Worker(host,port,"OctaveWorker");
      worker.addCommand(new OctaveCommand());
      worker.run();
    }
}
```

OctaveWorker/TABFile.java

```
import java.io.*;

class TABFile {
    public static TimeSeriesData read(String filename) throws
IOException {
      int columnCount = getColumnCount(filename);
      int rowCount = getRowCount(filename);

      String header[];
      double data[][] = new double[columnCount][];
      for(int i=0; i<columnCount; i++) {
          data[i] = new double[rowCount];
      }

      BufferedReader in = new BufferedReader(new FileReader(filename));
      header = in.readLine().split("\\t");

      for(int i=0; i<rowCount; i++) {
          String dataStr[] = in.readLine().split("\\t");

          if (dataStr.length != columnCount) {
            throw new IOException("Bad file format.");
          }

          for(int j=0; j<columnCount; j++) {
            try {
                data[j][i] = Double.parseDouble(dataStr[j]);
            } catch(NumberFormatException e) {
                throw new IOException("Bad file format.");
            }
          }
      }
```

```
      TimeSeriesData tsd = new TimeSeriesData();
      tsd.data = data;
      tsd.header = header;
      return tsd;
    }

    public static void write(TimeSeriesData data,String filename)
throws IOException {
      PrintStream out = new PrintStream(new
FileOutputStream(filename));
      for(int i=0; i<data.header.length; i++) {
          if (i != 0) out.print("\t");
          out.print(data.header[i]);
      }
      out.println();

      for(int i=0; i<data.data[0].length; i++) {
          for(int j=0; j<data.data.length; j++) {
            if (j != 0) out.print("\t");
            out.print(data.data[j][i]);
          }
          out.println();
      }
      out.close();
    }

    private static int getColumnCount(String filename) throws
IOException {
      BufferedReader in = new BufferedReader(new FileReader(filename));
      int c = in.read();
      int columnCount = 1;
      while ((c != '\n') && (c != -1)) {
          if (c == '\t') {
            columnCount++;
          }
          c = in.read();
      }
      in.close();
      return columnCount;
    }

    private static int getRowCount(String filename) throws IOException
{
      BufferedReader in = new BufferedReader(new FileReader(filename));
      int c = in.read();
      int rowCount = -1;
      while (c != -1) {
          if (c == '\n') {
            rowCount++;
          }
          c = in.read();
      }
      in.close();
      return rowCount;
```

```
    }
}
```

OctaveWorker/TimeSeriesData.java

```
class TimeSeriesData {
    String header[];
    double data[][];

    public void print() {
      for(int i=0; i<header.length; i++) {
          if (i != 0) System.out.print("\t");
          System.out.print(header[i]);
      }
      System.out.println();

      for(int i=0; i<data[0].length; i++) {
          for(int j=0; j<data.length; j++) {
            if (j != 0) System.out.print("\t");
            System.out.print(data[j][i]);
          }
          System.out.println();
      }
    }
}
OctaveWorker/OctaveFile.java


import java.io.*;

class OctaveFile {
    public static void write(String filename, TimeSeriesData tsd)
throws IOException {
      PrintStream out = new PrintStream(new
FileOutputStream(filename));
      for(int i=0; i<tsd.header.length; i++) {
          out.print("# name: " + tsd.header[i] + "\n");
          out.print("# type: matrix\n");
          out.print("# rows: 1\n");
          out.print("# columns: " + tsd.data[i].length + "\n");
          for(int j=0; j<tsd.data[i].length; j++) {
            out.print(" " + tsd.data[i][j]);
          }
          out.print("\n");
      }
      out.close();
    }

    public static TimeSeriesData read(String filename) throws
IOException {
      BufferedReader in = new BufferedReader(new FileReader(filename));
      TimeSeriesData tsd = new TimeSeriesData();
      in.readLine(); // read Octave version information line
      tsd.header = new String[1];
      try {
```

```
          tsd.header[0] = (in.readLine().split(" "))[2];
          tsd.data = new double[1][];
          String type = (in.readLine().split(" "))[2];
          if (type.equals("scalar")) {
            tsd.data[0] = new double[1];
            tsd.data[0][0] = Integer.parseInt(in.readLine());
          } else if (type.equals("matrix")) {
            int rows = Integer.parseInt((in.readLine().split(" "))[2]);
            int columns = Integer.parseInt((in.readLine().split("
"))[2]);
            if (rows != 1) {
                throw new IOException("Unsupported data type.");
            }
            tsd.data[0] = new double[columns];
            int c = in.read();
            for(int i=0; i<columns; i++) {
                c = in.read();
                String str = "";
                while ((c != -1) && (c != ' ')) {
                  str += (char)c;
                  c = in.read();
                }
                try {
                  tsd.data[0][i] = Double.parseDouble(str);
                } catch (NumberFormatException e) {
                  throw new IOException("Error reading file, number
expected but found \"" + str + "\".");
                }
            }
          } else {
            throw new IOException("Unsupported data type.");
          }
      } catch (NullPointerException e) {
          throw new IOException("Bad file format.");
      }
      in.close();
      return tsd;
    }
}
```

OctaveWorker/OctaveInterface.java

```
import java.io.*;

class OctaveInterface {
    public static TimeSeriesData run(TimeSeriesData tsd,
                                     String command,
                                     String outputName) throws IOException {
      // Write the input data to a file
      File inputDataFile = File.createTempFile("octave.",".in");
      OctaveFile.write(inputDataFile.getAbsolutePath(),tsd);

      // Write the octave commands to a file
      File sourceFile = File.createTempFile("octave.",".src");
```

```
      PrintStream out = new PrintStream(new
FileOutputStream(sourceFile));
      out.print("load -ascii -force " + inputDataFile + "\n");
      out.print(command + ";\n");
      File octaveOutputFile = File.createTempFile("octave.",".out");
      out.print("save -ascii " + octaveOutputFile + " " + outputName +
"\n");
      out.close();

      // Run octave
      Process p = null;
      try {
          String commandLine = "";
          if (System.getProperty("octave.path") == null) {
            throw new IOException ("\"octave.path\" system variable not
set.");
          } else {
            commandLine += System.getProperty("octave.path") + " ";
          }
          commandLine += "-q " + sourceFile;
          p = Runtime.getRuntime().exec(commandLine,null,new
File("."));
      } catch (IOException e) {
          throw new IOException("Octave not found.");
      }

      // Read the output data from octave
      String errorMessage = "";
      BufferedReader in = new BufferedReader(new
InputStreamReader(p.getErrorStream()));
      try {
          p.waitFor();
      } catch (InterruptedException e) {
          throw new IOException("Octave Interrupted.");
      }
      int c;
      while((c = in.read()) != -1) {
          errorMessage += (char)c;
      }

      // Delete the input and source files
      inputDataFile.delete();
      sourceFile.delete();

      // Read the output time series data
      try {
          TimeSeriesData tsdout =
OctaveFile.read(octaveOutputFile.getAbsolutePath());
          octaveOutputFile.delete();
          return tsdout;
      } catch (IOException e) {
          octaveOutputFile.delete();
          throw new IOException("Error reading octave output: " +
errorMessage);
      }
```

```
    }
}
```

OctaveWorker/OctaveCommand.java

```
import java.io.*;

public class OctaveCommand extends Command {
    public String getCommandName() {
      return "octave";
    }

    public int getInputCount() {
      return 2;
    }

    public int getOutputCount() {
      return 1;
    }

    public void run(Worker worker,
                File inputFiles[],
                File outputFiles[]) throws ServerException,IOException
{
      TimeSeriesData data;
      try {
          data = DataFile.read(inputFiles[0].getAbsolutePath());
      } catch (IOException e) {
          throw new ServerException("Unable to read input data file: "
+ e.getMessage());
      }

      String command;
      String outputName;
      try {
          BufferedReader in = new BufferedReader(new
FileReader(inputFiles[1]));
          command = in.readLine();
          outputName = in.readLine();
      } catch (IOException e) {
          throw new ServerException("Unable to read input command file:
" + e.getMessage());
      }

      TimeSeriesData out;
      try {
          out = OctaveInterface.run(data,command,outputName);
      } catch (IOException e) {
          throw new ServerException("Error running octave: " +
e.getMessage());
      }

      out.print();

      try {
```

```
            DataFile.write(out,outputFiles[0].getAbsolutePath());
        } catch (IOException e) {
            throw new ServerException("Error writing output file: " +
e.getMessage());
        }

        worker.reportProgress(100.0);
    }
}
```

Partioning Sorting Direct Method

The following source code is used to implement the Partitioning Sorting Direct Method. This code depends on the source code found in Appendix C and Appendix D.

ess.cpp

```
#include <sys/time.h>
#include <iostream>
#include <math.h>
#include "../common/Model.h"
#include "../common/Exception.h"
#include "../common/Infinity.h"
#include "Random.h"
#include "SortingDirectMethod.h"

using namespace std;

void printState(double time,
            Model *model,
            Vector<long> *species,
            FILE **speciesFiles) {
  for(unsigned long i=0; i<model->getOutputs()->size(); i++) {
    int speciesIndex = model->getOutputs()->get(i);
    if (speciesFiles[speciesIndex] != NULL) {
      fprintf(speciesFiles[speciesIndex],"%i\n",species-
>get(speciesIndex));
    }
  }
}

double getDoubleFromString(char *arg) {
  char *endPtr = NULL;
  double returnValue = strtod(arg,&endPtr);
  return returnValue;
}

void getStatus(int *speciesStatus, int *reactionStatus,
             char *oldModelFile, char *modelFile) {
  Model oldModel(oldModelFile);
  Model model(modelFile);

  for(int i=0; i<model.getSpecies()->size(); i++) {
    if (model.getSpecies()->get(i) != oldModel.getSpecies()->get(i)) {
      speciesStatus[i] = 1;
    }
  }

  for(int i=0; i<model.getReactions()->size(); i++) {
    if (model.getReactions()->getPtr(i)->getRateConstant() !=
      oldModel.getReactions()->getPtr(i)->getRateConstant()) {
      reactionStatus[i] = 1;
    }
```

```
  }

  int changes = 1;
  while (changes != 0) {
    changes = 0;

    for(int i=0; i<model.getSpecies()->size(); i++) {
      if (speciesStatus[i] == 1) {
      for(int j=0; j<model.getReactions()->size(); j++) {
        if (reactionStatus[j] == 0) {
          Reaction *rxn = model.getReactions()->getPtr(j);
          for(int k=0; k<rxn->getReactants()->size(); k++) {
            if (rxn->getReactants()->getPtr(k)->getSpeciesIndex() == i)
{
            reactionStatus[j] = 1;
            changes++;
            }
          }
        }
      }
      }
    }

    for(int i=0; i<model.getReactions()->size(); i++) {
      if (reactionStatus[i] == 1) {
      Reaction *rxn = model.getReactions()->getPtr(i);
      for(int j=0; j<model.getSpecies()->size(); j++) {
        if (speciesStatus[j] == 0) {
          if (rxn->affectsSpecies(j)) {
            speciesStatus[j] = 1;
            changes++;
          }
        }
      }
      }
    }
  }

  changes = 1;
  while (changes != 0) {
    changes = 0;

    for(int i=0; i<model.getSpecies()->size(); i++) {
      if (speciesStatus[i] == 0) {
      for(int j=0; j<model.getReactions()->size(); j++) {
        if (reactionStatus[j] == 1) {
          Reaction *rxn = model.getReactions()->getPtr(j);
          for(int k=0; k<rxn->getReactants()->size(); k++) {
            if (rxn->getReactants()->getPtr(k)->getSpeciesIndex() == i)
{
            speciesStatus[i] = 2;
            changes++;
            }
          }
        }
```

```
      }
      }
    }

    for(int i=0; i<model.getReactions()->size(); i++) {
      if (reactionStatus[i] == 0) {
      Reaction *rxn = model.getReactions()->getPtr(i);
      for(int j=0; j<model.getSpecies()->size(); j++) {
        if (speciesStatus[j] != 0) {
          if (rxn->affectsSpecies(j)) {
            reactionStatus[i] = 2;
            changes++;
          }
        }
      }
      }
    }
  }
}

int main(int argc, char **argv) {
  try {
    timeval before;
    timeval after;
    gettimeofday(&before,NULL);

    char *oldModelFile = NULL;
    char *modelFile = NULL;
    double printInterval;
    double endTime;

    if (argc == 4) {
      modelFile = argv[1];
      printInterval = getDoubleFromString(argv[2]);
      endTime = getDoubleFromString(argv[3]);
    } else if (argc == 5) {
      oldModelFile = argv[1];
      modelFile = argv[2];
      printInterval = getDoubleFromString(argv[3]);
      endTime = getDoubleFromString(argv[4]);
    } else {
      cout << "usage: ess [<old-model>] <model> <storage>"
         << " <print-interval> <end-time>" << endl;
      return -1;
    }

    Model model(modelFile);

    int *speciesStatus = new int[model.getSpecies()->size()];
    int *reactionStatus = new int[model.getReactions()->size()];

    if (oldModelFile == NULL) {
      for(int i=0; i<model.getSpecies()->size(); i++) speciesStatus[i]
= 1;
```

```
      for(int i=0; i<model.getReactions()->size(); i++)
reactionStatus[i] = 1;
    } else {
      for(int i=0; i<model.getSpecies()->size(); i++) speciesStatus[i]
= 0;
      for(int i=0; i<model.getReactions()->size(); i++)
reactionStatus[i] = 0;
      getStatus(speciesStatus,reactionStatus,oldModelFile,modelFile);
    }

    FILE **speciesFiles = new (FILE*)[model.getSpecies()->size()];
    for(int i=0; i<model.getSpecies()->size(); i++) {
      if ((speciesStatus[i] == 1) && (model.getOutputs()->find(i) != -
1)) {
      char filename[80];
      sprintf(filename,"species.%i.out",i);
      speciesFiles[i] = fopen(filename,"w");
      } else {
      speciesFiles[i] = NULL;
      }
    }

    Vector<long> species;
    species.resize(model.getSpecies()->size());
    for(unsigned int i=0; i<model.getSpecies()->size(); i++) {
      species.set(i,model.getSpecies()->get(i));
    }

    double currentTime = 0.0;
    unsigned long reactionCount = 0;
    double nextPrintTime = printInterval;

    SortingDirectMethod algorithm;
    double finalPrintTime = endTime + printInterval*0.01;

    Random random(55);


algorithm.init(&model,&random,&currentTime,&species,reactionStatus);

    printState(currentTime,&model,&species,speciesFiles);
    while (true) {
      algorithm.step();
      while ((nextPrintTime < finalPrintTime) && (currentTime >
nextPrintTime)) {
      printState(nextPrintTime,&model,&species,speciesFiles);
      nextPrintTime += printInterval;
      }
      if (currentTime > endTime) break;
      algorithm.execute();
      reactionCount++;
    }

    gettimeofday(&after,NULL);
    double totalTime = after.tv_sec - before.tv_sec;
```

```
    totalTime += ((double)after.tv_usec)/1000000.0;
    totalTime -= ((double)before.tv_usec)/1000000.0;
    ofstream out("run.profile");
    out << reactionCount << " " << totalTime << endl;

    return 1;
  } catch (Exception e) {
    cout << e.getMessage() << endl;
    return -1;
  }
}
```

SortingDirectMethod.h

```
#ifndef SORTINGDIRECTMETHOD_H
#define SORTINGDIRECTMETHOD_H

#include "../common/Model.h"
#include "RandomBuffer.h"
#include "MinHeap.h"
#include <iostream>

class SortingDirectMethod {
 private:
  Model *model;
  double *currentTime;
  unsigned long reactionIndex;
  Vector<double> propensities;
  Vector<long>* species;
  double totalPropensity;
  RandomBuffer *uniformRandomBuffer;
  RandomBuffer *exponentialRandomBuffer;
  Vector<unsigned long> reactionList;
  Vector<unsigned long> reactionCounts;
  int *reactionStatus;
  FILE **infiles;
  FILE **outfiles;
  MinHeap *minHeap;
  double nextLocalReactionTime;

  double getNextReactionTime(int reactionIndex);

 public:
  SortingDirectMethod();
  ~SortingDirectMethod();
  void init(Model *model, Random *random, double *currentTime,
          Vector<long>* species, int *reactionStatus);
  void reinit();
  void step();
  void execute();
  Vector<unsigned long>* getReactionCounts() { return &reactionCounts;
};
};
```

```
#endif
```

SortingDirectMethod.cpp

```
#include "SortingDirectMethod.h"
#include "../common/Exception.h"
#include <stdlib.h>
#include <iostream>
#include "MinHeap.h"

using namespace std;

SortingDirectMethod::SortingDirectMethod() {
  uniformRandomBuffer = NULL;
  exponentialRandomBuffer = NULL;
}

SortingDirectMethod::~SortingDirectMethod() {
  if (uniformRandomBuffer != NULL)
    delete uniformRandomBuffer;
  if (exponentialRandomBuffer != NULL)
    delete exponentialRandomBuffer;
}

void SortingDirectMethod::init(Model *model,
                         Random *random,
                         double *currentTime,
                         Vector<long>* species,
                         int *reactionStatus) {
  this->reactionStatus = reactionStatus;
  this->model = model;
  this->currentTime = currentTime;
  this->species = species;
  totalPropensity = 0.0;

  uniformRandomBuffer = new
RandomBuffer(random,100000,RANDOM_BUFFER_UNIFORM);
  if (uniformRandomBuffer == NULL) {
    throw Exception("SortingDirectMethod","init","Out of memory.");
  }

  exponentialRandomBuffer = new
RandomBuffer(random,100000,RANDOM_BUFFER_EXPONENTIAL);
  if (exponentialRandomBuffer == NULL) {
    throw Exception("SortingDirectMethod","init","Out of memory.");
  }

  propensities.resize(model->getReactions()->size());

  reactionList.resize(model->getReactions()->size());
  for(unsigned long i=0; i<model->getReactions()->size(); i++) {
    reactionList.set(i,i);
  }
```

```
  minHeap = new MinHeap(model->getReactions()->size());

  reactionCounts.resize(model->getReactions()->size());
  for(int i=0; i<reactionCounts.size(); i++) {
    reactionCounts.set(i,0);
  }
  reinit();
}

void SortingDirectMethod::reinit() {
  infiles = new (FILE*)[model->getReactions()->size()];
  outfiles = new (FILE*)[model->getReactions()->size()];

  totalPropensity = 0.0;
  for(unsigned long i=0; i<model->getReactions()->size(); i++) {
    char filename[80];
    sprintf(filename,"reaction.%i.out",i);
    if (reactionStatus[i] == 1) {
      propensities.set(i,model->getReactions()->getPtr(i)-
>getPropensity(species));
      totalPropensity += propensities.get(i);
      infiles[i] = NULL;
      outfiles[i] = fopen(filename,"wb");
      minHeap->update(i,INFINITY);
    } else if (reactionStatus[i] == 2) {
      propensities.set(i,0);
      infiles[i] = fopen(filename,"rb");
      outfiles[i] = NULL;
      minHeap->update(i,getNextReactionTime(i));
    } else {
      propensities.set(i,0);
      infiles[i] = NULL;
      outfiles[i] = NULL;
      minHeap->update(i,INFINITY);
    }
  }

  nextLocalReactionTime = exponentialRandomBuffer->getNumber() /
totalPropensity;
}

double SortingDirectMethod::getNextReactionTime(int reaction) {
  double time;
  int i = fread(&time,sizeof(double),1,infiles[reaction]);
  if (i == 0) {
    return INFINITY;
  }
  return time;
}

void SortingDirectMethod::step() {
  if (nextLocalReactionTime < minHeap->getMinValue()) {
    *currentTime = nextLocalReactionTime;
  } else {
```

```
    *currentTime = minHeap->getMinValue();
  }
}

void SortingDirectMethod::execute() {
  if (*currentTime == nextLocalReactionTime) {
    double scaledRand = uniformRandomBuffer->getNumber() *
totalPropensity;
    reactionIndex = 0;
    unsigned long i;
    for(i=0; i<model->getReactions()->size(); i++) {
      reactionIndex = reactionList.get(i);
      if (propensities.get(reactionIndex) != 0.0) {
      scaledRand -= propensities.get(reactionIndex);
      if (scaledRand <= 0.0) break;
      }
    }
    if (i != 0) {
      reactionList.swap(i,i-1);
    }
    fwrite(currentTime,sizeof(double),1,outfiles[reactionIndex]);
  } else {
    reactionIndex = minHeap->getMin();
  }

  model->getReactions()->getPtr(reactionIndex)->execute(species);

  double oldPropensity = totalPropensity;

  Vector<unsigned long>* dependencies = model-
>getReactionDependencies()->getPtr(reactionIndex);
  for(unsigned long i=0; i<dependencies->size(); i++) {
    unsigned long index = dependencies->get(i);
    if (reactionStatus[index] == 1) {
      totalPropensity -= propensities.get(index);
      propensities.set(index,model->getReactions()->getPtr(index)-
>getPropensity(species));
      totalPropensity += propensities.get(index);
    }
  }
  if (totalPropensity < 0) totalPropensity = 0.0;

reactionCounts.set(reactionIndex,reactionCounts.get(reactionIndex)+1);

  if (*currentTime == nextLocalReactionTime) {
    nextLocalReactionTime = *currentTime + exponentialRandomBuffer-
>getNumber() / totalPropensity;
  } else {
    if (oldPropensity == 0.0) {
      nextLocalReactionTime = *currentTime + exponentialRandomBuffer-
>getNumber() / totalPropensity;
    } else {
      nextLocalReactionTime -= *currentTime;
      nextLocalReactionTime *= oldPropensity/totalPropensity;
      nextLocalReactionTime += *currentTime;
```

```
    }
    minHeap->update(reactionIndex,getNextReactionTime(reactionIndex));
  }
}
```

Hardware Source Code

The following java program and include files are used to generate VHDL source code for for the hardware accelerated version of ESS.

hwess.java

```
import java.io.*;

class hwess {
    public static void printWithIndent(String str) {
      System.out.println("          " + str);
    }

    public static void printFile(String filename) throws IOException {
      BufferedReader in = new BufferedReader(new FileReader(filename));
      String s = in.readLine();
      while (s != null) {
          System.out.println(s);
          s = in.readLine();
      }
      System.out.println();
    }

    public static void printConstants(int propBits,
                              int rxnIdxBits) {
      System.out.println("constant PROP_BITS : integer " +
                    " := " + propBits + ";");
      System.out.println("constant RXN_COUNT : integer " +
                    ":= " + (int)(Math.pow(2,rxnIdxBits)) + ";");
      System.out.println("constant RXN_IDX_BITS : integer " +
                    ":= " + rxnIdxBits + ";");
      System.out.println();
    }

    public static void printSumTree(int rxnIdxBits) {
      int reactionCount = (int)Math.pow(2,rxnIdxBits);
      System.out.println();
      printWithIndent("when 1 =>");
      System.out.println();
      for(int i=0; i<reactionCount/2; i++) {
          printWithIndent("  prop_tree(" + i + ") := "
                        + "prop(" + (2*i) + ") + prop("
                        + (2*i+1) + ");");
      }
      System.out.println();
      printWithIndent("  if (reaction_counter = \"0000000000000000\")
then");
      printWithIndent("    out_done <= '1';");
      printWithIndent("    stage := 0;");
      printWithIndent("  else");
      printWithIndent("    stage := 2;");
```

```
      printWithIndent("  end if;");

      int laststart = 0;
      int start = reactionCount/2;
      int incr = reactionCount/4;

      for(int i=2; i<=rxnIdxBits; i++) {
          System.out.println();
          System.out.println();
          printWithIndent("when " + i + " =>");
          System.out.println();
          for(int j=0; j<incr; j++) {
            printWithIndent("  prop_tree(" + (j+start) + ") := " +
                        "prop_tree(" + (2*j+laststart) + ") + " +
                        "prop_tree(" + (2*j+1+laststart) + ");");
          }
          System.out.println();
          if (i == 2) {
            printWithIndent("  reaction_counter := reaction_counter -
1;");
          }
          printWithIndent("  stage := " + (i+1) + ";");

          laststart = start;
          start += incr;
          incr /= 2;
      }
    }

    public static void printMult(int rxnIdxBits) {
      int reactionCount = (int)Math.pow(2,rxnIdxBits);
      System.out.println();
      System.out.println();
      printWithIndent("when " + (rxnIdxBits+1) + " =>");
      System.out.println();
      printWithIndent("  mult_result := " +
                  "rand(PROP_BITS-1 downto 0) * " +
                  "prop_tree(" + (reactionCount-2) + ");");
      printWithIndent("  selector := " +
                  "mult_result(2*PROP_BITS-1 downto PROP_BITS);");
      printWithIndent("  stage := " + (rxnIdxBits+2) + ";");
    }

    public static String intToBitString(int value, int bits) {
      String s = "";
      for(int i=bits-1; i>=0; i--) {
          if ((value >> i) % 2 == 1) {
            s += "1";
          } else {
            s += "0";
          }
      }
      return s;
    }
```

```
    public static int printSearch(int rxnIdxBits) {
      int reactionCount = (int)Math.pow(2,rxnIdxBits);

      String tree[] = new String[reactionCount*2-1];
      int tree2[] = new int[reactionCount*2-1];
      for(int i=0; i<=reactionCount-2; i++) {
          tree[i] = "prop_tree(" + (reactionCount-2-i) + ")";
      }
      for(int i=0; i<reactionCount; i++) {
          tree[i + reactionCount-1] = "prop(" + (reactionCount-1-i) +
")";
          tree2[i + reactionCount-1] = reactionCount-1-i;
      }

      int endStage = rxnIdxBits+1+reactionCount;

      for(int i=0; i<reactionCount-1; i++) {
          int currentStage = i+rxnIdxBits+2;
          int rightIndex = i*2+2;
          int leftIndex = i*2+1;
          int rightStage = rightIndex+rxnIdxBits+2;
          int leftStage = leftIndex+rxnIdxBits+2;

          if (i < reactionCount / 2 - 1) {
            System.out.println();
            System.out.println();
            printWithIndent("when " + currentStage + " =>");
            printWithIndent("  if (selector < " + tree[rightIndex] + ")
then");
            printWithIndent("    stage := " + rightStage + ";");
            printWithIndent("  else");
            printWithIndent("    selector := selector - " +
tree[rightIndex] + ";");
            printWithIndent("    stage := " + leftStage + ";");
            printWithIndent("  end if;");
          } else {
            System.out.println();
            System.out.println();
            printWithIndent("when " + currentStage + " =>");
            printWithIndent("  if (selector < " + tree[rightIndex] + ")
then");
            printWithIndent("    selrxn := \"" +
                            intToBitString(tree2[rightIndex],rxnIdxBits)
+ "\";");
            printWithIndent("  else");
            printWithIndent("    selrxn := \"" +
                            intToBitString(tree2[leftIndex],rxnIdxBits)
+ "\";");
            printWithIndent("  end if;");
            printWithIndent("  stage := " + endStage + ";");
          }
      }

      return endStage;
    }
```

```
    public static int printMemStages(int stage) {
      System.out.println();
      System.out.println();
      printWithIndent("when " + stage + " =>");
      printWithIndent("  update_stage := \"000\";");
      printWithIndent("  mem_addr(RXN_IDX_BITS+2 downto 3) <=
selrxn;");
      printWithIndent("  mem_addr(2 downto 0) <= update_stage;");
      printWithIndent("  stage := " + (stage+1) + ";");

      System.out.println();
      System.out.println();
      printWithIndent("when " + (stage+1) + " =>");
      printWithIndent("  update_stage := update_stage + 1;");
      printWithIndent("  mem_addr(RXN_IDX_BITS+2 downto 3) <=
selrxn;");
      printWithIndent("  mem_addr(2 downto 0) <= update_stage;");
      printWithIndent("  stage := " + (stage+2) + ";");

      return stage+2;
    }


    public static void printFooter(int stage) throws IOException{
      System.out.println();
      System.out.println();
      printWithIndent("when " + stage + " =>");
      printFile("inc-footer.vxx");
    }

    public static void main(String args[]) throws IOException {
      int propBits = Integer.parseInt(args[0]);
      int rxnIdxBits = Integer.parseInt(args[1]);

      printFile("inc-header.vxx");
      printConstants(propBits,rxnIdxBits);
      printFile("inc-types.vxx");
      printFile("inc-process.vxx");
      printSumTree(rxnIdxBits);
      printMult(rxnIdxBits);
      int stage;
      stage = printSearch(rxnIdxBits);
      stage = printMemStages(stage);
      printFooter(stage);
    }
}
```

inc-footer.vxx

```
          for i in 0 to RXN_COUNT-1 loop
            if (mem_dout(RXN_IDX_BITS-1 downto 0) = a(i)) then
            if (mem_dout(6) = '0') then
```

```
              prop(i) := prop(i) + kB(i);
              kA(i) := kA(i) + k(i);
            else
              prop(i) := prop(i) - kB(i);
              kA(i) := kA(i) - k(i);
            end if;
            elsif (mem_dout(RXN_IDX_BITS-1 downto 0) = b(i)) then
            if (mem_dout(6) = '0') then
              prop(i) := prop(i) + kA(i);
              kB(i) := kB(i) + k(i);
            else
              prop(i) := prop(i) - kA(i);
              kB(i) := kB(i) - k(i);
            end if;
            end if;
          end loop;

          if (species_index = mem_dout(5 downto 0)) then
            if (mem_dout(6) = '0') then
              species := species + 1;
              else
                species := species - 1;
              end if;
            end if;

          if (mem_dout(7) = '0') then
            stage := 1;
          else
            update_stage := update_stage + 1;
            mem_addr(RXN_IDX_BITS+2 downto 3) <= selrxn;
            mem_addr(2 downto 0) <= update_stage;
          end if;

        when others =>
          null;

      end case;

      out_species <= species;
      end if;
    end if;
  end process;
end architecture a;
```

inc-header.vxx

```
library ieee;
library work;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity ess is
  port (in_clk : in std_logic;
```

```
      in_we : in std_logic;
      in_cmd : in std_logic_vector(3 downto 0);
      in_addr : in std_logic_vector(7 downto 0);
      in_data : in std_logic_vector(31 downto 0);
      out_done : out std_logic;
      out_species : out std_logic_vector(31 downto 0));
end entity ess;

architecture a of ess is

component lfsr32 is
  port   ( in_clock          : in std_logic;
         out_random_number : out std_logic_vector(31 downto 0));
end component lfsr32;

component ram8x2048
      port (
      addr: IN std_logic_VECTOR(10 downto 0);
      clk: IN std_logic;
      din: IN std_logic_VECTOR(7 downto 0);
      dout: OUT std_logic_VECTOR(7 downto 0);
      we: IN std_logic);
end component;

signal mem_we : std_logic;
signal mem_addr : std_logic_vector(10 downto 0);
signal mem_din : std_logic_vector(7 downto 0);
signal mem_dout : std_logic_vector(7 downto 0);
signal rand : std_logic_vector(31 downto 0);
```

inc-process.vxx

```
begin

  RAM1 : ram8x2048 port map (mem_addr,in_clk,mem_din,mem_dout,mem_we);

  LFSR1 : lfsr32 port map (in_clk,rand);

  process(in_clk)

    variable prop : propensity_vector(RXN_COUNT-1 downto 0);
    variable k : propensity_vector(RXN_COUNT-1 downto 0);
    variable kA : propensity_vector(RXN_COUNT-1 downto 0);
    variable kB : propensity_vector(RXN_COUNT-1 downto 0);
    variable A : species_index_vector(RXN_COUNT-1 downto 0);
    variable B : species_index_vector(RXN_COUNT-1 downto 0);

    variable species : std_logic_vector(31 downto 0);
    variable species_index : std_logic_vector(7 downto 0);

    variable reaction_counter : std_logic_vector(31 downto 0);

    variable mult_result : std_logic_vector(PROP_BITS*2-1 downto 0);
    variable selector : std_logic_vector(PROP_BITS-1 downto 0);
```

```
  variable selrxn : std_logic_vector(RXN_IDX_BITS-1 downto 0);

  variable prop_tree : propensity_vector(RXN_COUNT-2 downto 0);
  variable stage : integer range 0 to 100 := 0;
  variable update_stage : std_logic_vector(2 downto 0);

begin
  if (in_clk = '1' and in_clk'event) then

    out_done <= '0';
    mem_we <= '0';

    mem_addr <= "00000000000";

    if (in_we = '1') then

    case in_cmd is

      when "0000" =>  -- Set Propensity

        prop(conv_integer(in_addr(RXN_IDX_BITS-1 downto 0)))
          := in_data(PROP_BITS-1 downto 0);


      when "0001" =>  -- Set Rate Constant k

        k(conv_integer(in_addr(RXN_IDX_BITS-1 downto 0)))
          := in_data(PROP_BITS-1 downto 0);


      when "0010" =>  -- Set Rate Constant k * Species Population A

        kA(conv_integer(in_addr(RXN_IDX_BITS-1 downto 0)))
          := in_data(PROP_BITS-1 downto 0);


      when "0011" =>  -- Set Rate Constant k * Species Population B

        kB(conv_integer(in_addr(RXN_IDX_BITS-1 downto 0)))
          := in_data(PROP_BITS-1 downto 0);

      when "0100" =>  -- Set Species Index of A

        A(conv_integer(in_addr(RXN_IDX_BITS-1 downto 0)))
          := in_data(5 downto 0);


      when "0101" =>  -- Set Species Index of B

        B(conv_integer(in_addr(RXN_IDX_BITS-1 downto 0)))
          := in_data(5 downto 0);


      when "0110" =>  -- Set Species Population to Monitor
```

```
          species := in_data;

        when "0111" =>  -- Set Species Index to Monitor
          species_index := in_data(7 downto 0);

        when "1000" =>  -- Set Memory Location
          mem_addr(10 downto 3) <= in_addr;              -- Reaction
Index
          mem_addr(2 downto 0) <= in_data(18 downto 16); -- Update
Stage
          mem_din(5 downto 0) <= in_data(5 downto 0);    -- Species
Index
          mem_din(6) <= in_data(6);                      -- Add or
Subtract
          mem_din(7) <= in_data(7);                      -- continue?
          mem_we <= '1';

        when "1001" =>  -- Set Reaction Counter
          reaction_counter := in_data;

        when "1111" =>  -- GO!
          stage := 1;

        when others =>
          null;

      end case;

      else

      case stage is
        when 0 =>
          null;
```

inc-types.vxx

```
type propensity_vector is array(integer range <>) of
std_logic_vector(PROP_BITS-1 downto 0);
type species_index_vector is array(integer range <>) of
std_logic_vector(5 downto 0);
```

```
type mult_vector is array(integer range <>) of
std_logic_vector(PROP_BITS*2-1 downto 0);
type reaction_index_vector is array(integer range <>) of
std_logic_vector(RXN_IDX_BITS-1 downto 0);
```

lfsr32.vhd

```
library ieee;
library work;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity lfsr32 is
  port   ( in_clock          : in std_logic;
         out_random_number : out std_logic_vector(31 downto 0));
end entity lfsr32;

architecture a of lfsr32 is
begin
  process(in_clock)
    variable var_current_number : std_logic_vector(31 downto 0) :=
"10111011010010001011101101001000";
    variable var_next_bit : std_logic;
  begin
    if (in_clock = '1' and in_clock'event) then
      var_next_bit := var_current_number(0) XOR
                  var_current_number(26) XOR
                  var_current_number(27) XOR
                  var_current_number(31);
      var_current_number(31 downto 1) := var_current_number(30 downto
0);
      var_current_number(0) := var_next_bit;
        out_random_number <= var_current_number;
    end if;
  end process;
end architecture a;
```

To place the design on the Xilinx XUP Hardware Development board, the following source code is used to wrap the ess.vhdl file and interface to the RS232 serial port.

main.vhd

```
library ieee;
library work;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity main is
  port (SYSTEM_CLOCK : in std_logic;
      RS232_RX_DATA : in std_logic;
      RS232_TX_DATA : out std_logic);
end entity main;

architecture a of main is

  component uart is
    port (in_clk : in std_logic;
        in_rs232 : in std_logic;
        out_rs232 : out std_logic;
        in_data : in std_logic_vector(7 downto 0);
        in_signal : in std_logic;
        out_data : out std_logic_vector(7 downto 0);
        out_signal : out std_logic);
  end component;

  component clkdiv is
    port (in_clk : in std_logic;
        out_clk : out std_logic);
  end component;

   component ess is
     port (in_clk : in std_logic;
        in_we : in std_logic;
        in_cmd : in std_logic_vector(3 downto 0);
        in_addr : in std_logic_vector(7 downto 0);
        in_data : in std_logic_vector(31 downto 0);
        out_done : out std_logic;
        out_species : out std_logic_vector(31 downto 0));
   end component;

  signal clk50MHz : std_logic;
  signal clk25MHz : std_logic;
  signal clk13MHz : std_logic;
  signal clk6MHz : std_logic;
  signal clk3MHz : std_logic;
  signal clk1562KHz : std_logic;
  signal clk781KHz : std_logic;
  signal clk391KHz : std_logic;
  signal clk195KHz : std_logic;
  signal clk98KHz : std_logic;
```

```
  signal clk49KHz : std_logic;

  signal uart_din : std_logic_vector(7 downto 0);
  signal uart_dout : std_logic_vector(7 downto 0);
  signal uart_sigin : std_logic;
  signal uart_sigout : std_logic;

   signal ess_we : std_logic;
   signal ess_cmd : std_logic_vector(3 downto 0);
   signal ess_addr : std_logic_vector(7 downto 0);
   signal ess_data : std_logic_vector(31 downto 0);
   signal ess_done : std_logic;
   signal ess_species : std_logic_vector(31 downto 0);


begin

  cd1 : clkdiv PORT MAP(SYSTEM_CLOCK,clk50MHz);
  cd2 : clkdiv PORT MAP(clk50MHz,clk25MHz);
  cd3 : clkdiv PORT MAP(clk25MHz,clk13MHz);
  cd4 : clkdiv PORT MAP(clk13MHz,clk6MHz);
  cd5 : clkdiv PORT MAP(clk6MHz,clk3MHz);
  cd6 : clkdiv PORT MAP(clk3MHz,clk1562KHz);
  cd7 : clkdiv PORT MAP(clk1562KHz,clk781KHz);
  cd8 : clkdiv PORT MAP(clk781KHz,clk391KHz);
  cd9 : clkdiv PORT MAP(clk391KHz,clk195KHz);
  cd10 : clkdiv PORT MAP(clk195KHz,clk98KHz);
  cd11 : clkdiv PORT MAP(clk98KHz,clk49KHz);

  u1 : uart PORT MAP(clk49KHz,RS232_RX_DATA,RS232_TX_DATA,uart_din,
                uart_sigin,uart_dout,uart_sigout);

  ess1 : ess PORT MAP(clk50MHz,ess_we,ess_cmd,ess_addr,ess_data,
                 ess_done,ess_species);

  process(clk50MHz,clk49KHz)
    variable uart_stage : integer range 0 to 10;
    variable uart_cmd : std_logic_vector(3 downto 0);
    variable uart_addr : std_logic_vector(7 downto 0);
    variable uart_data : std_logic_vector(31 downto 0);

    variable ess_write : std_logic := '0';
    variable ess_write_last : std_logic := '0';

    variable ess_is_done : std_logic := '0';
    variable ess_is_done_reset : std_logic := '0';

  begin
    if (clk49KHz='1' and clk49KHz'event) then
      ess_write := '0';
      uart_sigin <= '0';

      ess_is_done_reset := '0';
      if (ess_is_done = '1') then
      uart_din <= "00000001";
```

```
  uart_sigin <= '1';
  ess_is_done_reset := '1';
  elsif (uart_sigout = '1') then
  case uart_stage is
    when 0 =>
      if (uart_dout = "00000001") then
        uart_din <= "00000001";
        uart_sigin <= '1';
      elsif (uart_dout = "00000010") then
        uart_stage := 1;
      elsif (uart_dout = "00000100") then
        uart_din <= ess_species(31 downto 24);
        uart_sigin <= '1';
      elsif (uart_dout = "00000101") then
        uart_din <= ess_species(23 downto 16);
        uart_sigin <= '1';
      elsif (uart_dout = "00000110") then
        uart_din <= ess_species(15 downto 8);
        uart_sigin <= '1';
      elsif (uart_dout = "00000111") then
        uart_din <= ess_species(7 downto 0);
        uart_sigin <= '1';
      end if;

    when 1 =>
      uart_cmd := uart_dout(3 downto 0);
      uart_stage := 2;

    when 2 =>
      uart_addr := uart_dout;
      uart_stage := 3;

    when 3 =>
      uart_data(31 downto 24) := uart_dout;
      uart_stage := 4;

    when 4 =>
      uart_data(23 downto 16) := uart_dout;
      uart_stage := 5;

    when 5 =>
      uart_data(15 downto 8) := uart_dout;
      uart_stage := 6;

    when 6 =>
      uart_data(7 downto 0) := uart_dout;
      ess_write := '1';
      uart_stage := 0;

    when others =>
      null;
  end case;
  end if;
end if;
```

```
    if (clk50MHz='1' and clk50MHz'event) then
      ess_we <= '0';

      if ((ess_write_last = '0') and (ess_write = '1')) then
      ess_cmd <= uart_cmd;
      ess_addr <= uart_addr;
      ess_data <= uart_data;
      ess_we <= '1';
      end if;

      if (ess_done = '1') then
      ess_is_done := '1';
      end if;

      if (ess_is_done_reset = '1') then
      ess_is_done := '0';
      end if;

      ess_write_last := ess_write;
    end if;
  end process;
end architecture a;
```

clkdiv.vhd

```
library ieee;
library work;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity clkdiv is
  port (in_clk : in std_logic;
      out_clk : out std_logic);
end entity clkdiv;

architecture a of clkdiv is
begin
  process(in_clk)
    variable myvar : std_logic;
  begin
    if (in_clk = '1' and in_clk'event) then
      myvar := not myvar;
      out_clk <= myvar;
    end if;
  end process;
end architecture a;
```

uart.vhd

```
library ieee;
```

```
library work;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity uart is
  port (in_clk : in std_logic;
      in_rs232 : in std_logic;
      out_rs232 : out std_logic;
      in_data : in std_logic_vector(7 downto 0);
      in_signal : in std_logic;
      out_data : out std_logic_vector(7 downto 0);
      out_signal : out std_logic);
end entity uart;

architecture a of uart is
begin
  process(in_clk)
    variable rx_reg : std_logic_vector(7 downto 0);
    variable rx_stage : integer range 0 to 9 := 0;
    variable rx_countdown : integer range 0 to 244;
    variable tx_reg : std_logic_vector(7 downto 0);
    variable tx_stage : integer range 0 to 10 := 0;
    variable tx_countdown : integer range 0 to 163;

  begin
    if (in_clk='1' and in_clk'event) then
      if ((tx_stage = 0) and (in_signal = '1')) then
      tx_reg := in_data;
      tx_stage := 1;
      tx_countdown := 20;
      end if;

      out_signal <= '0';

      case rx_stage is
      when 0 =>
        if (in_rs232 = '0') then
          rx_stage := 1;
          rx_countdown := 30;
        end if;

      when 9 =>
        rx_countdown := rx_countdown - 1;
        if (rx_countdown = 0) then
          rx_stage := 0;
          out_data <= rx_reg;
          out_signal <= '1';
        end if;

      when others =>
        rx_countdown := rx_countdown - 1;
        if (rx_countdown = 0) then
          rx_reg(rx_stage-1) := in_rs232;
          rx_countdown := 20;
          rx_stage := rx_stage + 1;
```

```
        end if;
      end case;

      case tx_stage is
      when 0 =>
        out_rs232 <= '1';

      when 1 =>
        out_rs232 <= '0';
        tx_countdown := tx_countdown - 1;
          if (tx_countdown = 0) then
          tx_countdown := 20;
          tx_stage := tx_stage + 1;
          end if;

      when 10 =>
        out_rs232 <= '1';
        tx_countdown := tx_countdown - 1;
          if (tx_countdown = 0) then
          tx_stage := 0;
          end if;

      when others =>
        out_rs232 <= tx_reg(tx_stage-2);
        tx_countdown := tx_countdown - 1;
        if (tx_countdown = 0) then
          tx_countdown := 20;
          tx_stage := tx_stage + 1;
        end if;
      end case;
    end if;
  end process;
end architecture a;
```

The following script file is used to synthesize the design using the Xilinx design flow.

synth

```
rm -rf route_dir
rm -rf synth_dir

cd java
rm ess.vhd

java hwess 32 4 > ess.vhd

cd ..
mkdir synth_dir
cd synth_dir
cp ../src/* .
mv ../java/ess.vhd .
echo "vhdl work ess.vhd" > xst.prj
echo "vhdl work main.vhd" >> xst.prj
echo "vhdl work uart.vhd" >> xst.prj
echo "vhdl work clkdiv.vhd" >> xst.prj
echo "vhdl work lfsr32.vhd" >> xst.prj
echo "run -ifn xst.prj -top main -ifmt MIXED -opt_mode SPEED -opt_level
2 -ofn test.ngc -p 2vp30ff896-6 -uc xup.xcf" > xst.in
xst -ifn xst.in
cd ..
mkdir route_dir
cp src/* route_dir
cp synth_dir/test.ngc route_dir
cd route_dir
ngdbuild -aul -p 2vp30ff896-6 test.ngc -uc xup.ucf
map test.ngd
par -ol high test.ncd -w test_par.ncd
trce test_par.ncd -a
bitgen test_par.ncd -w test.bit
cd ..
```

The following files are constraint files for the design.

xup.ucf

```
NET "LED_0" LOC = "AC4";
NET "LED_1" LOC = "AC3";
NET "LED_2" LOC = "AA6";
NET "LED_3" LOC = "AA5";

NET "LED_0" IOSTANDARD = LVTTL;
NET "LED_1" IOSTANDARD = LVTTL;
NET "LED_2" IOSTANDARD = LVTTL;
NET "LED_3" IOSTANDARD = LVTTL;

NET "LED_0" DRIVE = 12;
NET "LED_1" DRIVE = 12;
NET "LED_2" DRIVE = 12;
NET "LED_3" DRIVE = 12;

NET "LED_0" SLEW = SLOW;
NET "LED_1" SLEW = SLOW;
NET "LED_2" SLEW = SLOW;
NET "LED_3" SLEW = SLOW;

NET "SYSTEM_CLOCK" LOC = "AJ15";
NET "SYSTEM_CLOCK" IOSTANDARD = LVCMOS25;

NET "SW_0" LOC = "AC11";
NET "SW_1" LOC = "AD11";
NET "SW_2" LOC = "AF8";
NET "SW_3" LOC = "AF9";

NET "SW_0" IOSTANDARD = LVCMOS25;
NET "SW_1" IOSTANDARD = LVCMOS25;
NET "SW_2" IOSTANDARD = LVCMOS25;
NET "SW_3" IOSTANDARD = LVCMOS25;

NET "RS232_TX_DATA" LOC = "AE7";
NET "RS232_RX_DATA" LOC = "AJ8";
NET "RS232_DSR_OUT" LOC = "AD10";
NET "RS232_CTS_OUT" LOC = "AE8";
NET "RS232_RTS_IN"  LOC = "AK8";

NET "RS232_TX_DATA" IOSTANDARD = LVCMOS25;
NET "RS232_RX_DATA" IOSTANDARD = LVCMOS25;
NET "RS232_DSR_OUT" IOSTANDARD = LVCMOS25;
NET "RS232_CTS_OUT" IOSTANDARD = LVCMOS25;
NET "RS232_RTS_IN"  IOSTANDARD = LVCMOS25;

NET "RS232_TX_DATA" DRIVE = 8;
NET "RS232_DSR_OUT" DRIVE = 8;
NET "RS232_CTS_OUT" DRIVE = 8;

NET "RS232_TX_DATA" SLEW = SLOW;
```

```
NET "RS232_DSR_OUT" SLEW = SLOW;
NET "RS232_CTS_OUT" SLEW = SLOW;
xup.xcf

BEGIN MODEL main
NET SYSTEM_CLOCK PERIOD = 10 ns;
END;
```

To get the PC to communicate with the XUP system, I wrote the following Visual C++ program to serve as an interface.

HWESSInterface.cpp

```
// HWESSInterface.cpp : Defines the entry point for the console
application.
//
#include "stdafx.h"
#include "windows.h"
#include <iostream>
using namespace std;
#include <time.h>

HANDLE port;

void init() {
      port =
CreateFile("COM2",GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_EXISTING,0,NUL
L);
      SetupComm(port, 128, 128);

      DCB dcb;
      GetCommState(port, &dcb);
      dcb.BaudRate = 2400;
      dcb.ByteSize = 8;
      dcb.Parity = NOPARITY;
      dcb.StopBits = TWOSTOPBITS;
      dcb.fAbortOnError = TRUE;
      SetCommState(port, &dcb);
}

unsigned char read() {
      unsigned char data[2];
      DWORD junk;
      BOOL returnval = ReadFile(port, &data, 1, &junk, NULL);
      return data[0];
}

void write(unsigned char c) {
      unsigned char data[2];
      DWORD junk;
      data[0] = c;
      WriteFile(port,&data,1,&junk,NULL);
}


void check() {
      write(1);
      int data = read();
      if (data == 1) {
            cout << "Check Passed" << endl;
      } else {
            cout << "Check FAILED! -- " << data << endl;
```

```
	}
}

void sendCommandToESS(unsigned char command, unsigned char addr,
unsigned long data) {
	write(2);
	write(command);
	write(addr);
	write((data >> 24) & 0xff);
	write((data >> 16) & 0xff);
	write((data >> 8) & 0xff);
	write(data & 0xff);
}

unsigned long getSpecies() {
	write(4);
	unsigned long a = read();
	a = a << 8;
	write(5);
	a += read();
	a = a << 8;
	write(6);
	a += read();
	a = a << 8;
	write(7);
	a += read();
	return a;
}

unsigned long getSpecies(int idx) {
	write(4+4*(idx*2));
	unsigned long a = read();
	a = a << 8;
	write(5+4*(idx*2));
	a += read();
	a = a << 8;
	write(6+4*(idx*2));
	a += read();
	a = a << 8;
	write(7+4*(idx*2));
	a += read();
	return a;
}


void setPropensity(unsigned char addr, unsigned long value) {
	sendCommandToESS(0,addr,value);
}

void setK(unsigned char addr, unsigned long value) {
	sendCommandToESS(1,addr,value);
}

void setKA(unsigned char addr, unsigned long value) {
	sendCommandToESS(2,addr,value);
```

```
}

void setKB(unsigned char addr, unsigned long value) {
      sendCommandToESS(3,addr,value);
}

void setA(unsigned char addr, unsigned long value) {
      sendCommandToESS(4,addr,value);
}

void setB(unsigned char addr, unsigned long value) {
      sendCommandToESS(5,addr,value);
}

void setSpeciesValue(unsigned long value) {
      sendCommandToESS(6,0,value);
}

void setSpeciesIndex(unsigned long value) {
      sendCommandToESS(7,0,value);
}

void writeToMemory(unsigned char reaction, unsigned char stage,
                           unsigned long cont, char op, unsigned long
value) {
      if (op == '+') {
            sendCommandToESS(8,reaction, stage*65536 + cont*128 +
value);
      } else if (op == '-') {
         sendCommandToESS(8,reaction, stage*65536 + cont*128 + 64 +
value);
      } else {
         cout << "Invalid op" << endl;
      }
}

void setReactionCounter(unsigned long value) {
      sendCommandToESS(9,0,value);
}

void run() {
      sendCommandToESS(15,0,0);
      read();
}

void setReaction(unsigned char index,
                         unsigned long k,
                         unsigned long aIndex,
                         unsigned long aValue,
                         unsigned long bIndex,
                         unsigned long bValue) {
      setK(index,k);
      setKA(index,k*aValue);
      setKB(index,k*bValue);
      setPropensity(index,k*aValue*bValue);
```

```
      setA(index,aIndex);
      setB(index,bIndex);
}

void setReaction(unsigned char index,
                         unsigned long k) {
      setReaction(index,k,0,1,0,1);
}

void setReaction(unsigned char index,
                         unsigned long k,
                         unsigned long aIndex,
                         unsigned long aValue) {
      setReaction(index,k,aIndex,aValue,0,1);
}

void setupChaos8Model() {
      cout << "Writing Chaos8 Model." << endl;

      unsigned char R = 1;
      unsigned char M = 2;
      unsigned char B = 3;
      unsigned char G = 4;

      setReaction(0,784);
      setReaction(1,2,R,0);
      setReaction(2,2,R,0);
      setReaction(3,58,M,0);
      setReaction(4,78);
      setReaction(5,2,B,0);
      setReaction(6,22,M,0,B,0);
      setReaction(7,3,G,0);

      writeToMemory(0,0,0,'+',R);
      writeToMemory(1,0,0,'-',R);
      writeToMemory(2,0,0,'+',M);
      writeToMemory(3,0,0,'-',M);
      writeToMemory(4,0,0,'+',B);
      writeToMemory(5,0,0,'-',B);
      writeToMemory(6,0,0,'+',G);
      writeToMemory(7,0,0,'-',G);

      setSpeciesIndex(G);
      setSpeciesValue(0);
}

void setupChaos8x15Model() {
      cout << "Writing Chaos8x15 Model." << endl;

      unsigned char R = 1;
      unsigned char B = 2;
      unsigned char M[15];
      unsigned char G[15];
      for(int i=0; i<15; i++) {
            M[i] = 3+i;
```

```
            G[i] = 3+15+i;
      }

      setReaction(0,784);
      setReaction(1,2,R,0);
      setReaction(2,78);
      setReaction(3,2,B,0);
      writeToMemory(0,0,0,'+',R);
      writeToMemory(1,0,0,'-',R);
      writeToMemory(2,0,0,'+',B);
      writeToMemory(3,0,0,'-',B);

      for(int i=0; i<15; i++) {
            setReaction(4+i*4+0 ,2  ,R    ,0);
            setReaction(4+i*4+1 ,58 ,M[i] ,0);
            setReaction(4+i*4+2 ,22 ,B    ,0 ,M[i] ,0);
            setReaction(4+i*4+3 ,3  ,G[i] ,0);

            writeToMemory(4+i*4+0,0,0,'+',M[i]);
            writeToMemory(4+i*4+1,0,0,'-',M[i]);
            writeToMemory(4+i*4+2,0,0,'+',G[i]);
            writeToMemory(4+i*4+3,0,0,'-',G[i]);
      }

      setSpeciesIndex(G[3]);
      setSpeciesValue(0);
}
void setupDiffusionModel() {
      cout << "Writing Diffusion Model." << endl;

      unsigned char out = 1;
      unsigned char in[32];
      for(int i=0; i<32; i++) {
            in[i] = i+2;
      }

      for(int i=0; i<32; i++) {
            setReaction(i*2,10,out,10000);
            setReaction(i*2+1,10,in[i],0);

            writeToMemory(i*2,0,1,'+',in[i]);
            writeToMemory(i*2,1,0,'-',out);
            writeToMemory(i*2+1,0,1,'+',out);
            writeToMemory(i*2+1,1,0,'-',in[i]);
      }

      setSpeciesIndex(in[4]);
      setSpeciesValue(0);
}

int _tmain(int argc, _TCHAR* argv[])
{
      clock_t start, end;

      init();
```

```
	check();

//	setupChaos8x15Model();
	setupChaos8Model();
	//setupDiffusionModel();

	check();

	setReactionCounter(10000000);

	double total = 0;

	start = clock();
	run();
	end = clock();

	double duration = (double)(end - start) / CLOCKS_PER_SEC;
	cout << "Time: " << duration << endl;

	for(int i=0; i<1; i++) {
		cout << "Species: " << getSpecies(i) << endl;
	}

	check();


	check();

	return 0;
}
```

VITA

James Michael McCollum was born in Pittsburgh, PA on September 7, 1979. He was raised in the North Hills of Pittsburgh and attended grade school at Montessori Center Academy in Glenshaw, PA. James went to junior high school at Hampton Middle School in Allison Park, PA, and graduated from Hampton High School in 1997.

From there, he attended the University of Dayton in Dayton, OH and graduated with a B.E.E. in Electrical Engineering and a B.S. in Computed Science in 2001.

In June of 2001, James was hired by Neolinear, Inc in Pittsburgh, PA as a software engineer. Neolinear was acquired by Cadence Design Systems in 2004.

In December of 2001, he married his wife, Sarah.

In August of 2002, James began graduate school at the University of Tennessee – Knoxville. In 2004 he completed his M.S. in Electrical Engineering and in 2006 he completed his Ph.D. in Computer Engineering.

James is currently working as a tenure-track assistant professor in the Department of Electrical and Computing Engineering at Miami University in Oxford, OH.