BMC Bioinformatics

BioMed Central

Software

Open Access

FERN – a Java framework for stochastic simulation and evaluation of reaction networks

Florian Erhard, Caroline C Friedel* and Ralf Zimmer

Address: LFE Bioinformatik, Institut für Informatik, Ludwig-Maximilians-Universität München, Amalienstraße 17, 80333 München, Germany

Email: Florian Erhard - erhardf@cip.ifi.lmu.de; Caroline C Friedel* - Caroline.Friedel@bio.ifi.lmu.de; Ralf Zimmer - ralf.zimmer@ifi.lmu.de

* Corresponding author

Published: 29 August 2008

BMC Bioinformatics 2008, **9**:356 doi:10.1186/1471-2105-9-356

Received: 21 December 2007

Accepted: 29 August 2008

This article is available from: http://www.biomedcentral.com/1471-2105/9/356

Abstract

Background: Stochastic simulation can be used to illustrate the development of biological systems over time and the stochastic nature of these processes. Currently available programs for stochastic simulation, however, are limited in that they either a) do not provide the most efficient simulation algorithms and are difficult to extend, b) cannot be easily integrated into other applications or c) do not allow to monitor and intervene during the simulation process in an easy and intuitive way. Thus, in order to use stochastic simulation in innovative high-level modeling and analysis approaches more flexible tools are necessary.

Results: In this article, we present FERN (Framework for Evaluation of Reaction Networks), a Java framework for the efficient simulation of chemical reaction networks. FERN is subdivided into three layers for network representation, simulation and visualization of the simulation results each of which can be easily extended. It provides efficient and accurate state-of-the-art stochastic simulation algorithms for well-mixed chemical systems and a powerful observer system, which makes it possible to track and control the simulation progress on every level. To illustrate how FERN can be easily integrated into other systems biology applications, plugins to Cytoscape and CellDesigner are included. These plugins make it possible to run simulations and to observe the simulation progress in a reaction network in real-time from within the Cytoscape or CellDesigner environment.

Conclusion: FERN addresses shortcomings of currently available stochastic simulation programs in several ways. First, it provides a broad range of efficient and accurate algorithms both for exact and approximate stochastic simulation and a simple interface for extending to new algorithms. FERN's implementations are considerably faster than the C implementations of gillespie2 or the Java implementations of ISBJava. Second, it can be used in a straightforward way both as a stand-alone program and within new systems biology applications. Finally, complex scenarios requiring intervention during the simulation progress can be modelled easily with FERN.

Background

Traditionally, wet-lab experiments were focused on describing the function of individual genes or proteins. With the advent of high-throughput technologies, system-level approaches have become common which make it possible to identify the interactions between the individual elements of the cell. Here, mathematical models are crucial in understanding these biological systems. In par-

ticular the dynamic simulation of these models can illustrate and predict quantitative aspects of the system such as gene expression in regulatory networks or signal amplification in signal transduction networks [1].

The most common approach to modeling dynamics is via ordinary differential equations (ODEs) which describe deterministically how the system evolves with time (see e.g. [2-4]). Since the simulation of ODEs is deterministic, successive simulations starting from the same initial conditions lead to the same results. Biological systems, however, are not deterministic which can lead to quite different outcomes for the same initial conditions.

The stochastic nature of biological systems can be simulated using numerical simulation algorithms such as the stochastic simulation algorithm (SSA) of Gillespie [5]. The Gillespie algorithm simulates the system reaction by reaction. A reaction step in this case consists of two parts (see Figure 1). First, the time interval τ until the next reaction is drawn from the exponential distribution $P(\tau) = a \exp(-a\tau)$ using the inversion method. Here, a is the sum over all reaction propensities a_μ. Second, the reaction μ which is to occur in this time interval is drawn with propability $P(\mu|\tau) = a_\mu/a$. At the end of each step, molecule numbers and reaction propensities are updated. Both simulations via ODEs and SSAs assume a well-mixed system with a homogeneous distribution of molecules.

The original Gillespie algorithm (also called the direct method) has been modified in several ways to improve runtime. Here, the most commonly used modification is the next-reaction method by Gibson and Bruck [6]. This algorithm improves on a less efficient variant of the direct method which generates τ_k for each reaction R_k and then fires the reaction with the minimum τ_k. The minimum τ_k and the corresponding reaction are obtained from a priority queue and time intervals are updated without drawing new random numbers only for those reactions whose propensity was changed by the firing of the reaction. Reactions with changed propensities are identified with a dependency graph which contains an edge from reaction R_i to reaction R_j if R_i changes the molecule number of at least one reactant of R_j. It has been been suggested that the

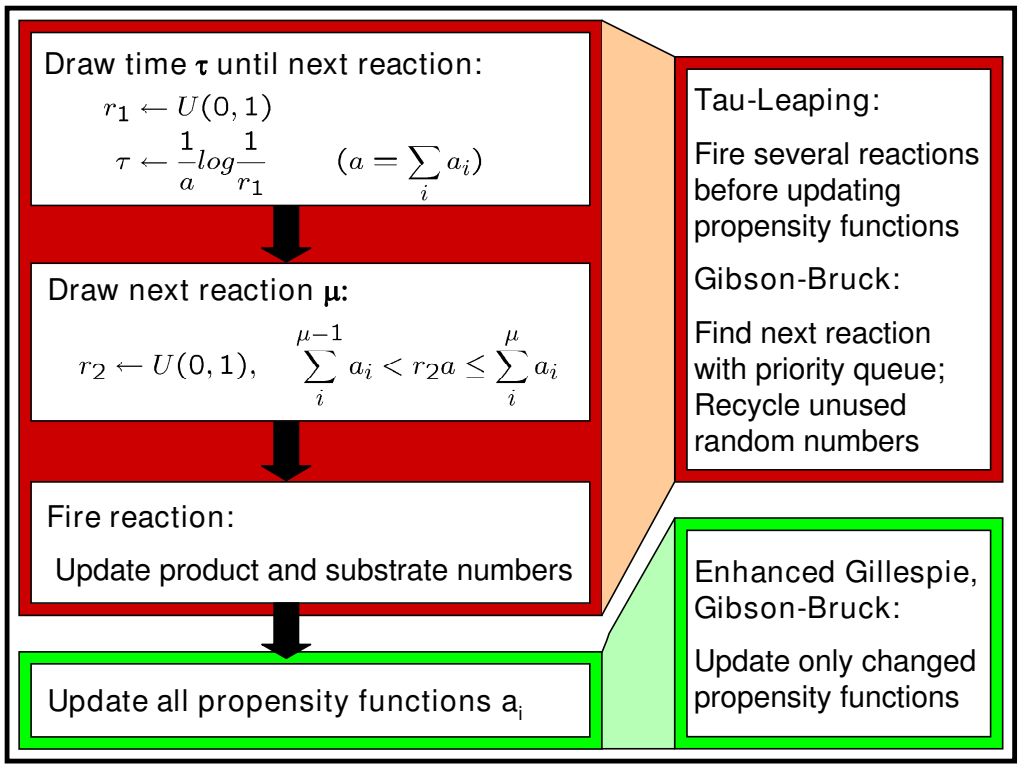

Figure 1
Stochastic simulation. This figure shows the flow of one simulation step. On the left-hand side the flow for the original Gillespie algorithm can be seen. On the right-hand side, we illustrate how the different steps are modified by the Gibson-Bruck, enhanced Gillespie and tau-leaping algorithms. Here, $U(0, 1)$ denotes the uniform distribution on the range of 0 to 1 and a_μ the reaction propensity for reaction μ.

next-reaction method is actually less efficient than improved versions of the direct method [7]. Indeed, we could show in our implementation that a variant of the direct method which uses the dependency graph technique to update propensities is significantly faster than the Gibson and Bruck algorithm.

The direct and next-reaction methods are exact methods. This means reaction propensities are updated after each reaction. Recently, Gillespie [8] proposed an approximative method, *tau-leaping*, which performs all reactions in a certain interval τ before updating the propensity functions. The interval size τ is chosen such that the propensity functions remain almost constant in this interval and reactions may fire multiple times. This, however, can sometimes lead to negative populations and as a consequence, this method has been improved later by Cao et al. [9,10] to avoid this problem. The modified tau-leaping algorithm automatically switches to the exact SSA for a few steps if the choice of τ becomes too small.

Both exact and tau-leaping models cannot be used to efficiently simulate models with multiple scales in molecule concentrations or reaction rates. Exact methods are too inefficient to simulate many fast reactions and high molecule concentrations. On the other hand, the presence of low molecule concentrations and slow reactions in the systems will effectively lead to small τ values for the tau-leaping methods and thus make them behave as the exact methods. To circumvent these problems, hybrid methods have been developed which partition the system into fast and slow reactions [11-26]. The slow reactions are then generally simulated using the exact SSA. The fast reactions are solved either deterministically or with the Langevin equation [11-18] or simulated with tau-leaping methods [18,19]. Alternatively, the model is simplified such that the effect of the fast reactions is incorporated in the simulation of the slow reactions, e.g. using quasi-steady-state assumptions, without actually firing the fast reactions [20-26].

Several implementations of stochastic simulation algorithms are already available, e.g. COPASI [27], Dizzy [28] using the SSA implementations of the ISBJava library, gillespie2 [29], STOCKS [30], StochKit [31], and BioNetS [32]. In general, these programs were designed as standalone programs and as a consequence the user is limited to the functionalities of the user interface. This makes it difficult to use the implementations of the SSAs within other programs. Furthermore, most of these programs provide only one implementation of an exact SSA which is not always fast enough for practical systems biology applications. However, faster SSAs such as e.g. the approximative tau-leaping procedure or new hybrid algorithms cannot be added to the programs easily by the users.

The StochKit software and ISBJava library provide these faster tau-leaping algorithms and the latter was also designed to be used within other systems biology programs. The output of the corresponding SSA implementations, however, is limited to the molecule concentrations. More flexible implementations are necessary to simulate complex high-level models and integrate stochastic simulation algorithms in new and innovative analysis and modeling tools. Two examples which illustrate the need for more flexible tools (see also results) are the visualization of the simulation progress directly in a network and the simulation of cell growth and division. With current simulation tools, it is not possible to implement both examples without having to change the code of the actual simulation algorithms considerably.

In this article, we present FERN, a Java framework for modeling and simulating biological systems which provides accurate and state-of-the-art simulation algorithms (exact, approximate and hybrid) and has been designed to be easily extendable to new ones (see Figure 2). With the help of observers, the simulation progress can be monitored on every level and modifications to the systems can be introduced during simulations in an intuitive way. Even with these additional functionalities, the implemented algorithms are faster than the ISBJava implementations. Results can be visualized easily and networks can be loaded from different sources. Contrary to ISBJava, FERN supports the most current version of SBML and allows arbitrary rate law definitions. FERN can be used as an integral part of other Java applications or as a standalone program in the form of a command-line tool and plugins to Cytoscape and CellDesigner.

Implementation

FERN is an object oriented library implemented in Java (see Additional file 1). Although it consists of more than 100 classes and interfaces, most classes are just implementations of one of three major interfaces and abstract classes (see Figure 2):

1. The interface *Network* provides the network structure of the model.

2. The abstract class *Simulator* performs simulations on a *Network*. It additionally calls the registered observers during the simulation run.

3. The abstract class *Observer* traces the simulation progress and creates the simulation output.

A simple simulation can be performed in only five lines of code, one line for each of: loading a network file, creating a simulator, creating and registering an observer, running the simulations and printing the results (see Figure 3).

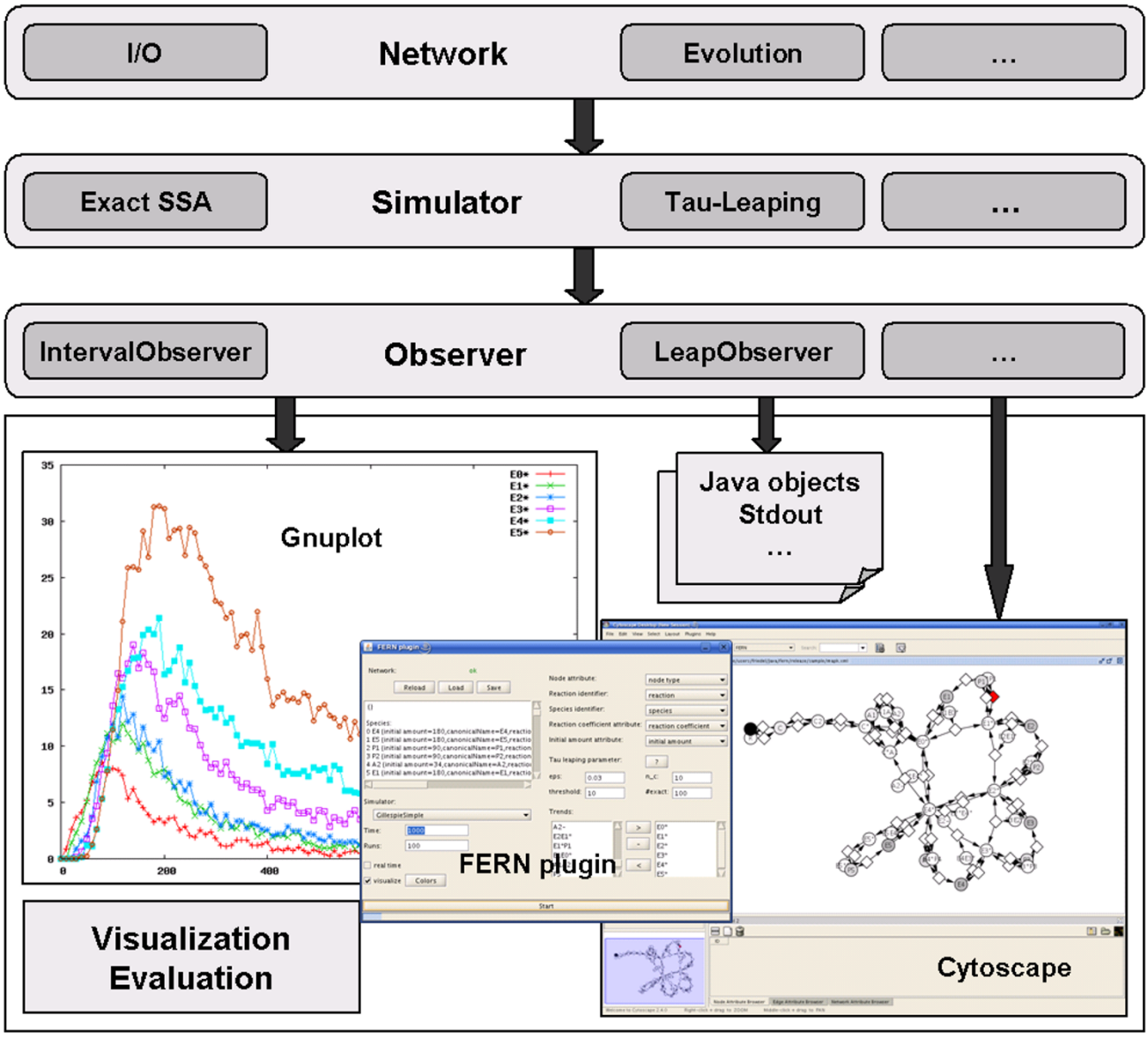

Figure 2
FERN design. The figure illustrates the overall design of FERN into three layers. Each layer is represented by one interface or abstract class: Interface *Network* and abstract classes *Simulator* and *Observer*.

More complex examples for using FERN can be found in the FERN distribution. In the following the three layers of FERN are described in more detail.

Networks

The interface *Network* describes the network's structure, i.e. the reactions and species in the networks. For this purpose, each reaction and each species is described by an integer value. Furthermore, the network stores basic information like species names and their initial molecule numbers. For the simulation more information is necessary which is stored in three additional classes (see Figure 4):

- The *AmountManager* controls the amount of each molecular species during the course of a simulation.

- The *AnnotationManager* can store additional annotations for the network, its species and reactions.

```
public static void main(String[] args)
            throws FeatureNotSupportedException
{
   // Load network
    SBMLNetwork net = new SBMLNetwork(
                        new File("some_net.xml"));

    // Create simulator
    Simulator sim = new GillespieEnhanced(net);

    // Register observer
    Observer obs = sim.addObserver(
           new AmountIntervalObserver(sim,1,"X"));

    // Register events
    net.registerEvents(sim);

    // Start simulation
    sim.start(50);

    // Print results
    System.out.println(obs);
}
```

Figure 3
Example program. This figure shows a small example on how to use FERN for running a simulation on a network. First, a network is loaded from an SBML file and then a simulator is created. In the next step, an observer is created and registered with the simulator. In this example, the observer records the current amount of molecule *X* every second of simulated time. The SBML events are registered with the simulator and the simulation is started to run for 50 seconds. Finally, the recorded results for *X* are printed.

• The *PropensityCalculator* calculates the propensities for the reactions by the specified kinetic laws.

There are three types of implementations of the *Network* interface:

• *Readers* which can read network data from files (e.g. FernMLNetwork, SBMLNetwork)

• *Wrappers* which redirect method calls to existing network classes (e.g. CytoscapeNetworkWrapper)

• *Evolution algorithms* which create networks from scratch by certain rules (e.g. AutocatalyticNetwork)

For each network, stochastic simulations can be performed with all implemented simulation algorithms.

Import and export of networks
FERN supports two formats for loading and exporting networks: the SBML format [33] as well as the simpler but also XML based FernML format. For reading and writing the SBML format, FERN uses the Java bindings of the C library (libSBML) available at http://www.sbml.org. Thus, it can be easily adapted to new developments of the SBML format. Currently, SBML version 2 levels 1–3 are supported.

From the model loaded by libSBML from the SBML file, a FERN *SBMLNetwork* is created using the list of compartments, species, reactions, parameters and events in the model. Events have to be registered with a simulator by the *SBMLNetwork* if they are to be triggered during the simulation (see Figure 3 for an example). Triggering of events is handled by specific observers.

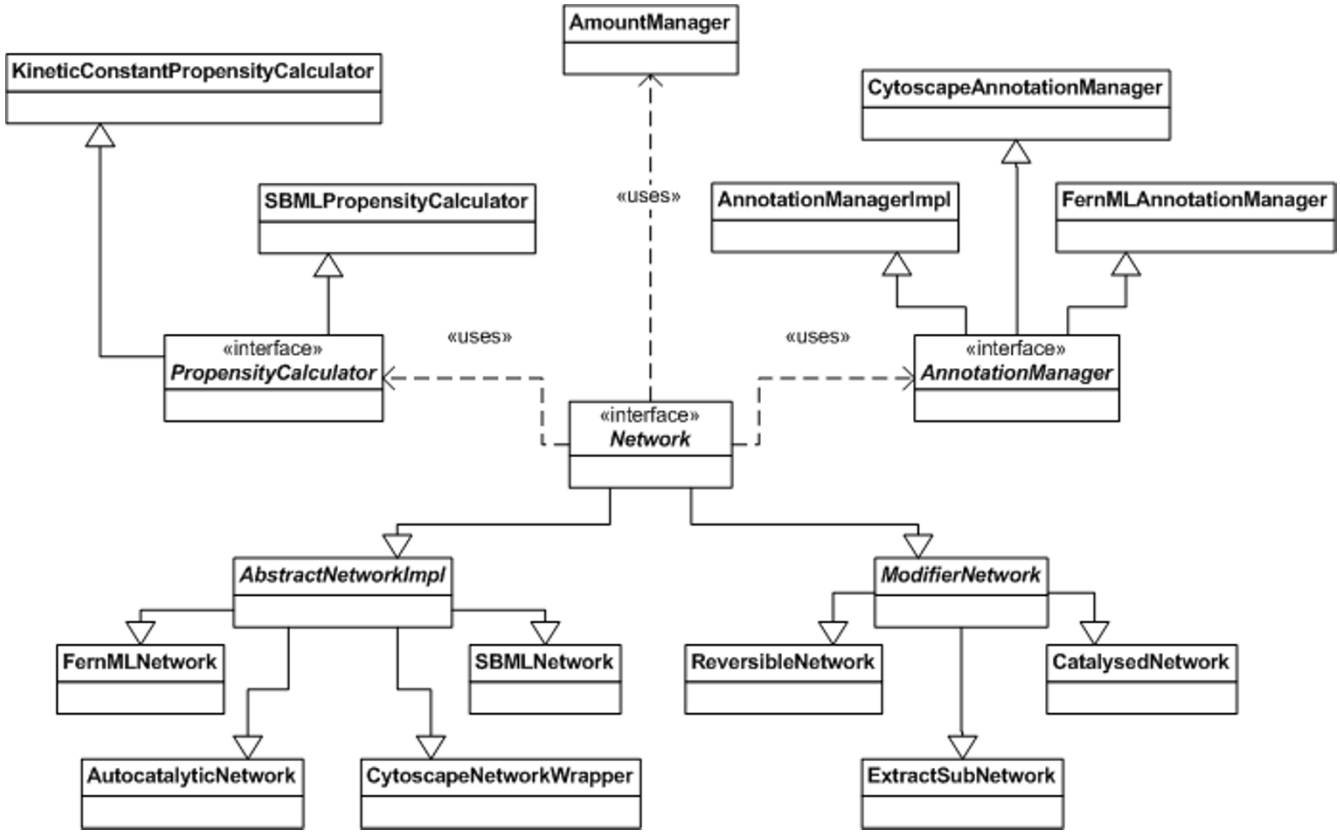

Figure 4
UML diagram of Network related classes. This figure shows an UML diagram for the *Network* interface and related interfaces and classes.

Currently, the *SBMLNetwork* class uses only the features of SBML necessary for the simulation of the network. It supports MathML to define complex reaction mechanisms but not rules, constraints or function definitions. If these features are required they can be incorporated easily by extending the SBMLNetwork class and loading these features from the SBML model created by libSBML. Since many systems biology applications support SBML (e.g. CellDesigner [34]), the SBML format can be used as an interchange format between FERN and these other applications.

SBML is a powerful format which can provide lots of information about a model. In contrast, FernML stores only the topology of the reaction network, optional annotations and the simulation parameters (see Additional file 2 and Additional file 3). This results in a much more simplified input format. More complex aspects, such as volume change due to cell growth and division, can then be modeled in Java using the FERN library in a straightforward way (see Results for an example). As a consequence, arbitrarily complex models can be designed.

Since FernML supports only the reaction rate equations used by Gillespie [5], the propensities can be recalculated at each step efficiently by a few arithmetic operations. SBML uses MathML to store the kinetics of a reaction. This allows for more complex reaction mechanisms and is particularly useful if the model cannot be formulated exclusively with first or higher order rate equations. To evaluate MathML expressions, FERN creates expression trees from them which have to be evaluated every time a propensity is calculated. Since this is one of the essential steps of SSAs, the simulation of an SBML network in FERN can be significantly slower than the simulation of the same network as a FernML network (see Figure 5). Thus, if only simple reaction rate equations are used, an SBML network should be converted to a FernML network using the provided conversion methods before performing the simulation.

FERN is not restricted to the input formats currently available. Any new input format can be easily included by implementing the *Network* interface or extending the *AbstractNetworkImpl* class.

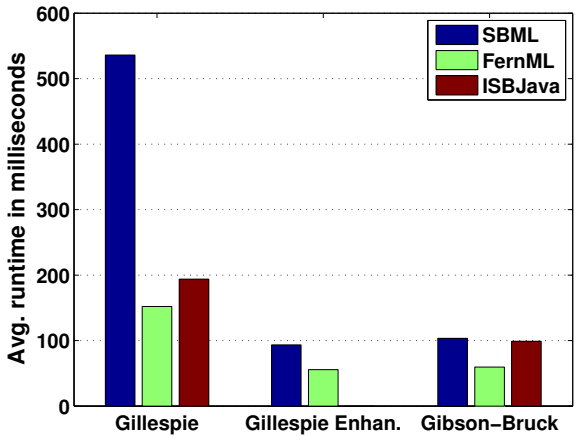

Figure 5
Runtime Comparisons. The EGF signaling pathway described by Lee et al. [38] was simulated with the three exact methods provided by FERN (original and enhanced Gillespie algorithm and the next reaction method by Gibson and Bruck) for a simulated time of 800 seconds both with an SBML network using expression trees to represent MathML expressions and a FernML network. For each combination of network type and stochastic simulation algorithm, 1,000 simulations were performed and the average runtime in milliseconds was calculated. The same simulations were performed with the Gillespie and Gibson-Bruck algorithms of ISBJava. All results were obtained on one processor of an Intel Core2Duo with 2.4 GHz. Standard errors in all cases were < 1.5 milliseconds.

Simulation algorithms

FERN provides implementations for three exact stochastic simulation algorithms, three state-of-the-art tau leaping procedures (see [8,10]) and a hybrid method combining SSA and tau-leaping [19]. The exact SSAs implemented include the original direct method of Gillespie [5], the next reaction method of Gibson and Bruck [6] and an enhanced version of the direct method. This enhanced method uses the dependency graph technique of the next reaction method to only update the propensity functions which are affected by the firing of a reaction. Apart from this improvement, it is identical to the direct method.

The tau-leaping algorithms are all based on the modified tau-leaping procedure proposed by Cao et al. [9] which avoids the problem of negative populations observed for the original tau-leaping procedure. This method switches to an exact SSA (in our implementations the enhanced Gillespie) for a few steps if the selected τ becomes too small. The three implementations differ only in the way the error is bounded (see [10] for details). The error is bounded either by the sum of all propensity functions (*TauLeapingAbsoluteBoundSimulator*), the relative changes in the individual propensity functions (*TauLeapingRelativeBoundSimulator*) or the relative changes in the molecular populations (*TauLeapingSpeciesPopulationBound Simulator*).

Furthermore, FERN implements the hybrid method by Puchalka and Kierzek [19] which partitions the system during the simulation into slow reactions which involve only small molecule numbers and fast reactions which involve large molecule numbers. The slow reactions are then simulated using an exact SSA while the fast reactions are simulated with tau-leaping. This algorithm was chosen over other hybrid methods for two reasons. First it uses only stochastic simulation algorithms, i.e. exact SSA and tau-leaping, and no further assumptions such as quasi-steady state. Second, the partitioning of the system is performed dynamically according to the state of the system and updated after each reaction step. Our implementation of the hybrid method uses our more efficient enhanced Gillespie algorithm (see Figure 5) instead of the Gibson and Bruck algorithm used by Puchalka and Kierzek. On the model of LacZ and LacY gene expression by Kierzek [30], the hybrid method speeds up the runtime by a factor of 98 compared to the enhanced Gillespie algorithm.

Each simulation algorithm is implemented by extending the abstract class *Simulator* or one of its subclasses. A simulation consists of the following steps (see also Additional file 3). First, the data structures are initialized and the simulation is started by passing a simulation controller implementing the *SimulationController* interface. The simulation controller decides after each step if the simulation can continue. The most basic one is the *DefaultController* which lets the simulation run until a given simulated time is reached.

In each step, the behavior of the simulator depends on the simulation algorithm implemented (see Figure 1). The direct and enhanced Gillespie algorithms draw the time interval τ till the next reaction from an exponential distribution. The reaction to be fired is then drawn with a probability proportional to its reaction propensity. For this purpose, a random variable r_2 is first drawn from the uniform distribution between 0 and 1. The corresponding reaction μ is then identified via a linear search such that $\sum_i^{\mu-1} a_i < r_2 \sum_i a_i \leq \sum_i^{\mu} a_i$. The Gibson-Bruck algorithm generates τ_k for each reaction R_k and at each step fires the reaction with the minimum τ_k obtained from a priority queue. Tau-leaping methods also choose a time interval τ but in this case several reactions can be fired during this interval (for more details see [9]).

After each step, the molecule numbers for reactants and products and the propensity functions for the reactions are recalculated. Here, reactants and products of a reaction can be identified efficiently from the adjacency list for this reaction stored in the network structure. Propensities are updated efficiently for all simulation algorithms apart from the original Gillespie algorithm using a dependency graph which stores for each reaction all reactions whose propensity is changed by the firing of this reaction.

Future developments of the algorithms can easily be included into FERN by extending one of the SSA implementations or the original *Simulator* class. In the same way ODE solvers or simulators for spatial models which are not provided by FERN can be integrated.

Observer system

FERN uses observers to trace the simulation progress and react to events. For this purpose, each observer has to implement functions which describe its response at specific time points of the simulations. Such responses may occur either at the beginning or the end of a simulation, before each step, after a reaction is fired or when a certain time is reached. In order to be notified of these events, observers have to be registered with the simulator.

Observer implementations are provided for tracing the molecule numbers for some species in arbitrary intervals, recording the firings of reactions, computing distributions of molecule numbers at a certain time over many simulation runs as well as many others. Several observers can be registered for a simulation at the same time and most of them can also handle repeated simulation runs, e.g. to create average curves or curves containing all trajectories for the individual simulation runs.

Visualization

In general, the observers use gnuplot to present their results. Once gnuplot is installed on a system and accessible e.g. via the path variable, the *Gnuplot* class makes it possible to easily create plots and retrieve them as *Image* objects, save them as files or present them online in a window. Plots can be customized using appropriate gnuplot commands.

Furthermore, FERN was used to implement Cytoscape [35] and CellDesigner [34] plugins for running and visualizing the simulations from within the Cytoscape or CellDesigner environments (for more details see Results).

Stochastics

An important feature of FERN is that random number generation is handled by the singleton class *Stochastics*. Accordingly, only one instance of this class is instantiated during a FERN run and all calls for random numbers are referred to this instance. This has several advantages. First, the underlying random number generator can be easily replaced if faster and better random number generators are developed. Currently, the Mersenne Twister implementation of the Colt Project is used http://dsd.lbl.gov/~hoschek/colt/. Second, by setting the seed value for the random number generator explicitly, the simulation can be made deterministic and e.g. interesting trajectories can be reproduced. Third, it is possible to count the number of random number generations necessary for different implementations of SSAs.

Results

Cytoscape plugin for stochastic simulation

Cytoscape [35] is a software platform for visualizing and integrating networks with an emphasis on biological data. It provides a flexible plugin architecture which can be used to enrich the platform with additional methods. We used this functionality to create a plugin which uses FERN to simulate networks loaded into Cytoscape (see Additional file 3). This plugin makes it possible to track the simulation progress directly on the network. Furthermore, it shows how FERN can be easily integrated into other applications and how the observer system can be used to visualize more than just the changes in molecule numbers. Each network readable by Cytoscape can be used for simulation by the plugin if it consists of two distinct types of nodes, namely reactions and molecular species. Furthermore, the initial amount of each molecular species and the reaction rate coefficient for each reaction have to be given. These parameters and the node type (species or reaction) can be read from arbitrary node attributes specified in Cytoscape. Additionally, the plugin provides access to FernML files in both directions. Thus every Cytoscape network can be saved as FernML, and every FernML file can be loaded into Cytoscape.

Simulations can be performed with every stochastic simulation algorithm provided by FERN and the simulation progress can be visualized directly on the network. Reaction nodes flash up whenever the corresponding reaction is fired and the species nodes are colored according to their molecule numbers. Furthermore, simulations can be run in real-time, which causes the algorithms to pause between two reaction events according to the simulated time. Trend curves of molecular species can also be created using gnuplot.

The implementation of the Cytoscape plugin is straightforward. A central plugin class integrates FERN into the Cytoscape platform by creating a menu item to start the plugin and to load the user interface. Apart from the classes defining the user interface, only a few additional classes are necessary. The most important ones are a wrapper class implementing the *Network* interface to map the

Cytoscape network structure to FERN and an *Observer* class to make the visualization possible. Additionally, FERN provides its own *Visual Style* (which defines how nodes and edges are colored and shaped) to guarantee a proper display of the network and to handle the flashing and recoloring of reaction and species nodes, respectively. The Cytoscape plugin was also adapted as a plugin to CellDesigner [34] which now offers a plugin functionality with the recent version 4.0 beta.

Simulation of cell growth and division using observers

The Cytoscape plugin is one example how observers can be used to track the simulation progress on various levels. Another example which illustrates the potential of the observer system is the simulation of the LacZ model described by Kierzek et al. [30,36] and based on experimental results by Kennell and Riezman [37]. This model requires the simulation of cell division. After each cell division, the stochastic simulation is continued with one promotor molecule and all other molecule numbers divided by 2. RNA polymerase and ribosome molecules are assumed to remain approximately constant with natural variations. For this purpose, the number of these molecules has to be adjusted after each simulation step by drawing from normal distributions. Furthermore, cell growth leads to a linear volume change.

With existing stochastic simulation programs, this model can, in general, only be simulated by changing the code of the actual simulation algorithms. Contrary to that, the model can be easily simulated with FERN by simply defining a cell growth observer. Before each simulation step, the observer checks if a generation has been completed. If this is the case, all molecule numbers are adjusted as described before. In any case, the volume size is adjusted to account for either cell division or cell growth, and the RNA polymerase and ribosome molecule numbers are drawn randomly.

This approximation was also used by Kierzek et al. and assumes that cell volume does not change during a simulation step. To perform an exact simulation of volume change, propensity functions would have to be defined which handle the cell volume as a function of time. However, since the volume change during one reaction is extremely small, the differences between the approximate and exact results should be negligible. Using the cell growth observer, we simulated the LacZ model with the enhanced Gillespie algorithm. Our results for the concentration of the LacZ protein (see Figure 6) show clearly the periodic oscillation in the protein numbers due to cell growth and division. From these results, we can estimate the rate of LacZ protein synthesis by a linear fit to the increasing LacZ concentrations during the first generation.

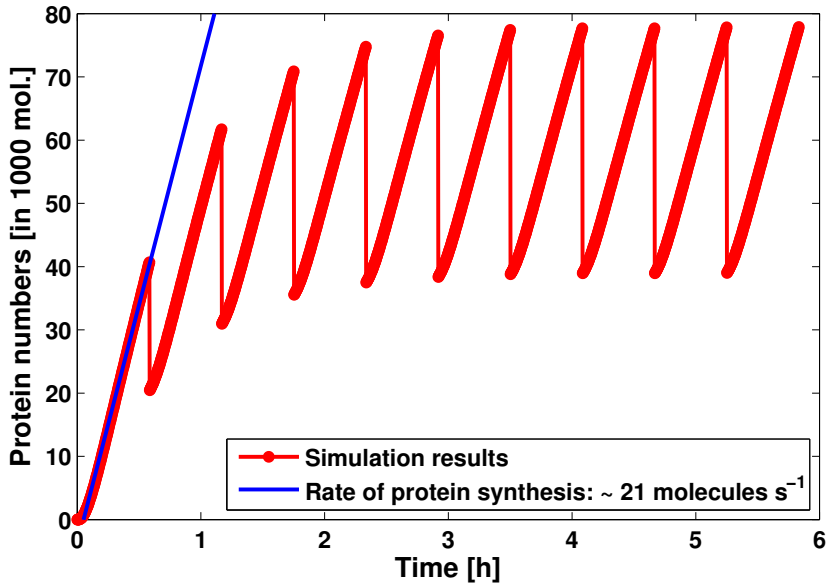

Figure 6
Results for the LacZ model. Average results of 1,000 simulations are shown for the LacZ protein over ten bacterial generations (red). After each generation (2100 s) the number molecules for each species was divided by 2 to simulate cell division. The blue line shows a linear fit to the increasing LacZ concentration during the first generation. This yields a rate of protein synthesis of 21s^{-1}.

Here, we obtained a rate of protein synthesis of $21s^{-1}$ which is close to the $22s^{-1}$ obtained by Kierzek et al. [30] and the $20s^{-1}$ reported by Kennell and Riezman [37].

The LacZ model both in FernML and SBML format and the code for running the simulation is included with the FERN distribution along with several other models such as the model of the EGF signaling pathway by Lee et al. [38].

Accuracy of stochastic simulation algorithms

To test the accuracy of the implemented stochastic simulation algorithms we used the Discrete Stochastic Models Test Suite (DSMTS) [39]. This test suite provides 36 stochastic models in the SBML format which have been solved either analytically or numerically. To test the implementation of a stochastic simulation algorithm, simulations have to be performed a large number of times (in general 10,000 times) for each individual model. The test is failed for a model if the distribution of the results is statistically significantly different from the known underlying distribution.

All three exact stochastic simulation algorithms in FERN were tested with the DSMTS test suite. Of the 36 models only the test 3.4 is failed. Models 1.10 and 1.11 are rejected because hasOnlySubstanceUnits is not declared to be true. If the rejection is overridden, Model 1.11 is failed, too. According to the DSMTS user guide, failure is expected for this model due to the inappropriate definition of the SBML model. In model 3.4, molecule numbers are reset whenever one molecule exceeds a certain number. This may lead to larger variations than accounted for by the thresholds used in the tests.

To asses our results we also evaluated gillespie2, the stochastic simulation program by some of the authors of the test suite [29]. Since the version of gillespie2 available online does not support level 2, version 3 of SBML, only 33 of the 36 models could be evaluated. We found that tests for model 1.11 and 3.4 are also failed, as well as tests for models 1.3, 1.14, 3.6 and 3.7. Two other models (1.17 and 1.18) could not be simulated as the rate law definition was not accepted by the program. Furthermore, we compared the runtime of FERN and gillespie2 on the DSMTS models which could be run by both programs and found that the runtime of FERN was significantly less than the runtime of gillespie2 (see Additional file 3).

Runtime performance

Runtime performance of the exact SSA implementations of FERN was compared against the performance of the Gillespie and Gibson-Bruck algorithms of ISBJava and the Gillespie algorithm of gillespie2. For this purpose, simulations were performed for the EGF signaling pathway by Lee et al. [38] which contains 39 molecular species and 19 reversible and 12 irreversible reactions. For each implementation, 1,000 simulations were performed for a simulated time of 800 seconds and results were obtained for the activated enzymes of the signaling cascade (see Figure 5 and Additional file 3).

Our results show that the implementations of the original Gillespie and Gibson-Bruck algorithm of FERN are always more efficient for the FernML network than the implementations provided by ISBJava. For the SBML network, the performance is similar for the Gibson-Bruck simulator but significantly worse for the original Gillespie algorithm. This is due to the evaluation of the MathML expressions required at each step of the simulation for each molecular species. However, this allows for more complex definitions of kinetic laws than possible in ISBJava which supports SBML only up to level 1, version 2 without MathML. If we compare FERN's implementation of the Gillespie algorithm to gillespie2 which also supports MathML, we observe that FERN is more than three times faster than gillespie2 on the SBML network.

Furthermore, the enhanced implementation of the Gillespie algorithm provided by FERN is more efficient both for FernML and SBML than any of the exact methods provided by ISBJava. This shows that the powerful observer system of FERN does not come at the cost of a reduced runtime performance. Accordingly, FERN is a useful library for stochastic simulation even if the observer tools are not used.

Discussion

In this article, we presented FERN, a Java framework for modeling and simulating biological reaction networks. FERN is subdivided into three layers which are represented by either one interface or abstract class. The functionalities of the package are then provided by implementations of these classes. Accordingly, FERN can be easily extended. For instance, any network class can use the algorithms of FERN by implementing the *Network* interface. New simulation algorithms can be implemented easily by overriding only a few methods of the abstract *Simulator* class and filling them with the new functionality. In this way, arbitrary FERN-readable networks can be simulated in different ways and the presentation system can be exploited.

It is possible to do reasonable simulations with FERN in just five lines of Java code. Each of the five steps can be expanded to cover more complex scenarios and simulations can be controlled at different levels. For instance, to simulate cell growth, an observer can be modified to change the volume of the simulation space. Alternatively, an interesting subnetwork can be selected on which simu-

lations can then be run. FERN can be easily integrated into other applications making its functionalities available within different environments. We have illustrated this by implementing FERN plugins to Cytoscape and CellDesigner. With only few additional classes, the Cytoscape plugin enables the users to follow the simulation progress directly on the network. This was made possible by the powerful observer system provided by FERN which is one of its major advantages compared to other available simulation programs.

The accuracy of our SSA implementations was analyzed by applying them to the Discrete Stochastic Models Test Suite. All three of the exact simulation algorithms passed 94.4% of the DSMTS models which is significantly better than the performance of gillespie2 which passes only 80.6% of the tests. This shows that the SSAs provided by FERN are highly accurate as well as fast. Even though FERN is implemented in Java which is often claimed to be less efficient than C, FERN's original gillespie algorithm is significantly faster than the C implementation of gillespie2 (see Additional file 3).

Compared with the ISBJava library, FERN has several advantages. First, FERN is more flexible than ISBJava and offers more functions for tracking and interacting with simulations. Second, it implements both a hybrid algorithm as well as the most current tau-leaping methods which resolve the problem of negative concentrations. Furthermore, its stochastic simulation algorithms are significantly faster than the ISBJava implementations. Finally, it supports the current version of SBML and allows arbitrary rate laws.

Conclusion

FERN is an easy-to-use framework for modeling and simulating reaction networks and can be easily integrated into other systems biology applications implemented in Java. It provides state-of-the-art stochastic simulation algorithms, efficient representations of networks with several input and output options and various ways of tracing and visualizing simulation data. Although some available stochastic simulation programs offer a few specialized features not yet supported by FERN such as e.g. time-delayed dynamics, none of them offer such a wide range of features and can be extended to new features as easily as FERN. Thus, FERN is a useful tool for biochemical network analysis or the development of new analysis methods or applications.

Availability and requirements

- Project name: FERN
- Project home page: http://www.bio.ifi.lmu.de/FERN/
- Operating system(s): Platform independent
- Programming language: Java
- Other requirements: Java 1.5 or higher; Colt package http://dsd.lbl.gov/~hoschek/colt/; JDOM http://www.jdom.org/
- Optional: libSBML http://www.sbml.org/software/libsbml/ for SBML version 2 level 1–3 support; Cytoscape 2.4.0 or higher http://www.cytoscape.org/ to use the Cytoscape plugin; CellDesigner 4.0 beta or higher http://www.systems-biology.org/cd/ to use the CellDesigner plugin; gnuplot http://www.gnuplot.info/.
- License: FERN is freely available under the GNU Lesser General Public License (LGPL).

Authors' contributions

EF designed and implemented the framework. CCF helped with the design and coordinated the project. EF and CCF drafted the manuscript. RZ provided advice and guidance for the project and helped to revise the manuscript. All authors read and approved the final manuscript.

Additional material

Additional file 1

FERN distribution, Version 1.3. This archive contains the FERN source code and binaries as well as documentation and example models in FernML and SBML.
Click here for file
[http://www.biomedcentral.com/content/supplementary/1471-2105-9-356-S1.zip]

Additional file 2

FERN user guide. The user guide provides instructions on installing and using FERN, as well as a description of the software architecture and a specification of FernML and supported features of SBML.
Click here for file
[http://www.biomedcentral.com/content/supplementary/1471-2105-9-356-S2.pdf]

Additional file 3

Supplementary Figures. This file contains Supplementary Figures on FernML, the simulation cycle, the Cytoscape plugin and runtime comparisons between FERN and gillespie2.
Click here for file
[http://www.biomedcentral.com/content/supplementary/1471-2105-9-356-S3.pdf]

Acknowledgements

We thank the anonymous reviewers for many helpful comments on the earlier versions of the manuscript and the FERN software.

References

1. Szallasi Z, Stelling J, Periwal V: *System Modeling in Cellular Biology* MIT Press; 2006.
2. Clodong S, Dühring U, Kronk L, Wilde A, Axmann I, Herzel H, Kollmann M: **Functioning and robustness of a bacterial circadian clock.** *Mol Syst Biol* 2007, **3:**90.
3. Shimoni Y, Friedlander G, Hetzroni G, Niv G, Altuvia S, Biham O, Margalit H: **Regulation of gene expression by small non-coding RNAs: a quantitative view.** *Mol Syst Biol* 2007, **3:**138.
4. Calzone L, Thieffry D, Tyson JJ, Novak B: **Dynamical modeling of syncytial mitotic cycles in Drosophila embryos.** *Mol Syst Biol* 2007, **3:**131.
5. Gillespie DT: **A general method for numerically simulating the stochastic time evolution of coupled chemical reactions.** *Journal of Computational Physics* 1976, **22(4):**403-434.
6. Gibson M, Bruck J: **Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels.** *Journal of Physical Chemistry A* 2000, **104(9):**1876-1889.
7. Cao Y, Li H, Petzold L: **Efficient formulation of the stochastic simulation algorithm for chemically reacting systems.** *J Chem Phys* 2004, **121(9):**4059-4067.
8. Gillespie DT: **Approximate accelerated stochastic simulation of chemically reacting systems.** *J Chem Phys* 2001, **115:**1716-1733.
9. Cao Y, Gillespie DT, Petzold LR: **Avoiding negative populations in explicit Poisson tau-leaping.** *J Chem Phys* 2005, **123(5):**054104.
10. Cao Y, Gillespie DT, Petzold LR: **Efficient step size selection for the tau-leaping simulation method.** *Journal of Chemical Physics* 2006, **124:**044109.
11. Haseltine EL, Rawlings JB: **Approximate simulation of coupled fast and slow reactions for stochastic chemical kinetics.** *The Journal of Chemical Physics* 2002, **117:**6959-6969.
12. Burrage K, Tian T, Burrage P: **A multi-scaled approach for simulating chemical reaction systems.** *Prog Biophys Mol Biol* 2004, **85(2–3):**217-234.
13. Vasudeva K, Bhalla US: **Adaptive stochastic-deterministic chemical kinetic simulations.** *Bioinformatics* 2004, **20:**78-84.
14. Kiehl TR, Mattheyses RM, Simmons MK: **Hybrid simulation of cellular behavior.** *Bioinformatics* 2004, **20(3):**316-322.
15. Salis H, Kaznessis Y: **Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions.** *J Chem Phys* 2005, **122(5):**54103.
16. Cao Y, Gillespie D, Petzold L: **Multiscale stochastic simulation algorithm with stochastic partial equilibrium assumption for chemically reacting systems.** *Journal of Computational Physics* 2005, **206:**395-411.
17. Chiam KH, Tan CM, Bhargava V, Rajagopal G: **Hybrid simulations of stochastic reaction-diffusion processes for modeling intracellular signaling pathways.** *Phys Rev E Stat Nonlin Soft Matter Phys* 2006, **74(5 Pt 1):**051910.
18. Harris LA, Clancy P: **A "partitioned leaping" approach for multiscale modeling of chemical reaction dynamics.** *J Chem Phys* 2006, **125(14):**144107.
19. Puchalka J, Kierzek AM: **Bridging the gap between stochastic and deterministic regimes in the kinetic simulations of the biochemical reaction networks.** *Biophys J* 2004, **86(3):**1357-1372.
20. Rao CV, Arkin AP: **Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the Gillespie algorithm.** *The Journal of Chemical Physics* 2003, **118(11):**4999-5010.
21. Cao Y, Gillespie DT, Petzold LR: **Multiscale stochastic simulation algorithm with stochastic partial equilibrium assumption for chemically reacting systems.** *Journal of Computational Physics* 2004, **206:**395-411.
22. Salis H, Kaznessis YN: **An equation-free probabilistic steady-state approximation: dynamic application to the stochastic simulation of biochemical reaction networks.** *J Chem Phys* 2005, **123(21):**214106.
23. Cao Y, Gillespie DT, Petzold LR: **The slow-scale stochastic simulation algorithm.** *J Chem Phys* 2005, **122:**14116.
24. Goutsias J: **Quasiequilibrium approximation of fast reaction kinetics in stochastic biochemical systems.** *J Chem Phys* 2005, **122(18):**184102.
25. Samant A, Vlachos DG: **Overcoming stiffness in stochastic simulation stemming from partial equilibrium: a multiscale Monte Carlo algorithm.** *J Chem Phys* 2005, **123(14):**144114.
26. Samant A, Ogunnaike BA, Vlachos DG: **A hybrid multiscale Monte Carlo algorithm (HyMSMC) to cope with disparity in time scales and species populations in intracellular networks.** *BMC Bioinformatics* 2007, **8:**175.
27. Hoops S, Sahle S, Gauges R, Lee C, Pahle J, Simus N, Singhal M, Xu L, Mendes P, Kummer U: **COPASI-a COmplex PAthway SImulator.** *Bioinformatics* 2006, **22(24):**3067-3074.
28. Ramsey S, Orrell D, Bolouri H: **Dizzy: stochastic simulation of large-scale genetic regulatory networks.** *J Bioinform Comput Biol* 2005, **3(2):**415-436.
29. Gillespie CS, Wilkinson DJ, Proctor CJ, Shanley DP, Boys RJ, Kirkwood TBL: **Tools for the SBML Community.** *Bioinformatics* 2006, **22(5):**628-629.
30. Kierzek AM: **STOCKS: STOChastic Kinetic Simulations of biochemical systems with Gillespie algorithm.** *Bioinformatics* 2002, **18(3):**470-481.
31. Li H, Cao Y, Petzold L, Gillespie D: **Algorithms and Software for Stochastic Simulation of Biochemical Reacting Systems.** *Biotechnol Prog* 2008 , **24(1):**56-61.
32. Adalsteinsson D, McMillen D, Elston TC: **Biochemical Network Stochastic Simulator (BioNetS): software for stochastic modeling of biochemical networks.** *BMC Bioinformatics* 2004, **5:**24.
33. 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, Hunter PJ, Juty NS, Kasberger JL, Kremling A, Kummer U, Novère NL, Loew LM, Lucio D, Mendes P, Minch E, Mjolsness ED, Nakayama Y, Nelson MR, Nielsen PF, Sakurada T, Schaff JC, Shapiro BE, Shimizu TS, Spence HD, Stelling J, Takahashi K, Tomita M, Wagner J, Wang J, SBML Forum: **The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models.** *Bioinformatics* 2003, **19(4):**524-531.
34. Funahashi A, Tanimura N, Morohashi M, Kitano H: **CellDesigner: a process diagram editor for gene-regulatory and biochemical networks.** *BIOSILICO* 2003, **1:**159-162.
35. Shannon P, Markiel A, Ozier O, Baliga NS, Wang JT, Ramage D, Amin N, Schwikowski B, Ideker T: **Cytoscape: a software environment for integrated models of biomolecular interaction networks.** *Genome Res* 2003, **13(11):**2498-2504.
36. Kierzek AM, Zaim J, Zielenkiewicz P: **The effect of transcription and translation initiation frequencies on the stochastic fluctuations in prokaryotic gene expression.** *J Biol Chem* 2001, **276(11):**8165-8172.
37. Kennell D, Riezman H: **Transcription and translation initiation frequencies of the Escherichia coli lac operon.** *J Mol Biol* 1977, **114:**1-21.
38. Lee DY, Zimmer R, Lee SY, Park S: **Colored Petri net modeling and simulation of signal transduction pathways.** *Metab Eng* 2006, **8(2):**112-122.
39. Evans TW, Gillespie CS, Wilkinson DJ: **The SBML Discrete Stochastic Models Test Suite.** *Bioinformatics* 2008 , **24(2):**285-286.