Towards a Computational Steering and Petri Nets Framework for the Modelling of Biochemical Reaction Networks

Mostafa Herajy and Monika Heiner

Brandenburg University of Technology at Cottbus,
Computer Science Institute,
Data Structures and Software Dependability,
Postbox 10 13 44, 03044 Cottbus, Germany
http://www-dssz.informatik.tu-cottbus.de/

Abstract. Computational steering is an interactive remote control of a long running application. The user can adopt it, e.g., to adjust simulation parameters on the fly. Simulation of large-scale biochemical networks is often computationally expensive, particularly stochastic and hybrid simulation. Such extremely time-consuming computations necessitate an interactive mechanism to permit users to try different paths and ask "what-if-questions" while the simulation is in progress. In this context, Petri nets are of special importance, since they provide an intuitive visual representation of reaction networks. In this paper, we introduce a novel framework for combining Petri nets and computational steering for the representation and interactive simulation of biochemical networks. The main merits of the developed framework are: intuitive representation of biochemical networks by means of Petri nets, distributed collaborative and interactive simulation, and tight coupling of simulation and visualisation.

Keywords: Petri Nets; Computational Steering; Biochemical Network Simulation

1 Introduction

With the progress of molecular biology, systems biology has recently gained renewed interest to examine the structure and dynamics of cellular and organismal functions [12]. To provide such systems level understanding, we need dedicated methods and techniques to facilitate dry-lab experiments. Additionally, systems biologists need to collaborate together in conducting one and the same dry-lab experiment and interactively steer a simulation while it is running. Furthermore, with the advances of computing power, simulation engines can be distributed over different machines and therefore they need to be remotely controlled by the user. In this context, computational steering is an interactive technique which can inevitably contribute to achieve these goals.

Computational steering [1,19,21] is the tight coupling of visualisation and simulation; the user can change simulation parameters while the simulation is in progress. In other words, computational steering can be described as a remote control of a long running application [19]. An important property of computational steering is that it allows user interactions with the running simulation and monitoring of intermediate results rather than waiting until the simulation ends and then visualising the results.

Many well-known software tools have been developed to simulate biochemical networks (e.g., COPASI [10], Dizzy [20], and Cell Designer [3]). However, they lack some indispensable features, such as: collaboration, distribution and interactivity, which are helpful to systems biologists. Other web-based applications (e.g., Virtual Cell [18]) support some kind of collaboration and distribution. Nevertheless, using those software, users are not aware of what is happening in the background, which makes the entire simulation process appears as a black box. Systems biologists need to interact with their experiments and with each others by changing some key parameters and asking what-if-questions. Furthermore, such applications are based on the batch approach that disconnects the users from their experiments during the simulation [21]. Our aim is to overcome these limitations by integrating computational steering with the simulation of biochemical reaction networks. Accordingly, using computational steering, systems biologists could drive and guide a simulation in the direction of desirable output by monitoring intermediate results and adapting on-the-fly key parameters of the running application.

On the other side, many computational steering environments have been developed during the last two decades (e.g., CUMULVS [4], RealityGrid [19], POSSE [17], and DISCOVER [15]). However, they are used to build new applications or require modifications of the application source code. With other words, they often assume the existence of legacy simulation code which needs to be integrated in such environments [21]. Thus, they are inapplicable if the source code is unavailable. Additionally, they have a steep learning curve [17,19] which makes them unsuitable for systems biologists.

Correspondingly, Petri nets [2] have been proven to be useful in modelling biochemical networks, see e.g., [5,16], since they provide an intuitive graphical representation and a well-developed mathematical theory comprising a variety of analyses techniques. Thus, Petri nets may help to bridge the gap between theorists and experimentalist. In this paper, we are specifically interested in timed (quantitative) Petri nets [5,8] (stochastic, continuous and hybrid Petri nets) and their high level representations – hierarchical and coloured Petri nets [14]. Integrating Petri nets and computational steering into one framework results in a powerful and interactive tool. We get the elegant representation and the mathematical foundation merits of Petri nets along with the interactive capabilities of computational steering.

The main contribution of this paper is the introduction of a computational steering framework which utilises Petri nets as a formal modelling language for the representation and interactive simulation of biochemical reaction networks.

Its main features are: intuitive and understandable representation of reaction networks with the help of Petri nets, distributed collaborative and interactive simulation of biochemical networks, the tight coupling of visualisation and simulation, and the extendibility to include further simulators provided by the users. The implementation of this framework is available as part of Snoopy [6] - a tool to design and animate or simulate hierarchical graphs, among them qualitative, stochastic, continuous and hybrid Petri nets.

This paper is organised as follows: after this short introduction, we present our computational steering framework by discussing its interdependent individual components. In Section 3, we provide a case study to illustrate potential applications of computational steering in the biochemical context. Finally, we sum up by conclusions and future work.

2 Framework

In this section, the developed framework is precisely outlined. We start with a general overview of the high level organisation, followed by a description of the individual components. The related underlying biological context as well as a typical application scenario are also given during the subsequent presentation of the framework.

2.1 Overview

Figure 1 presents the general architecture of the proposed framework. Its main components are: the steering server, the steering graphical user interface, the steering application programming interface (API), and the internal and external simulators. These interdependent ingredients enable systems biologists not only to run their biochemical network models and get results, but also to share, distribute and interactively steer them. Additionally, systems biologists do not have to wait until the simulation ends and then to discover potentially incorrect results. Instead, using the proposed framework, errors could be discovered earlier and be immediately corrected during the simulation and if necessary, the simulation could be restarted using the current setting. Subsequently, the overall required time to carry out wet-lab experiments will substantially decrease.

The main component of the architecture is the steering server. It is the central manager of the model data and communication traffic between the different framework components. It is a multi-user, multi-model, multi-simulator, and multi-threaded server. Inside the server, data is organised in terms of individual models which are in turn defined by means of Petri nets. Section 2.2 gives more information about the operations and functionalities of the steering server.

The steering graphical user interface is the end user's entry point to the overall architecture. Through it, the user can monitor and steer the simulation output and the corresponding key parameters, respectively. Users can flexibly connect and disconnect from their local machines to the available steering servers and view the currently running models. Model dynamics are produced using

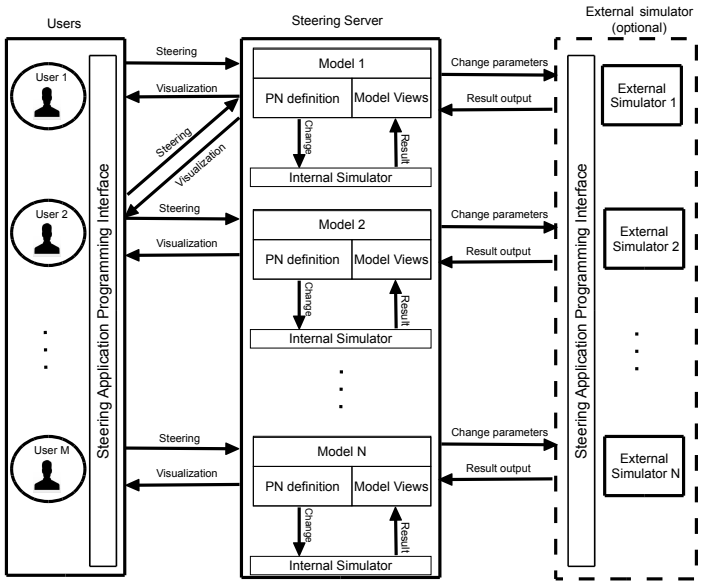

Fig. 1. Petri nets and computational steering framework. The framework consists of four components: steering server, steering graphical user interface (GUI), steering application programming interface (Steering APIs), and simulators (internal and external). The flow of information goes in two opposite directions: from the simulator to the user (monitoring) and from the user to the simulator (steering). The data structure inside the server is organised in terms of Petri nets: places, transitions, arcs and parameters. A model can contain different views. Views are defined by the users and submitted to the server for further download and manipulation.

either an internal or an external simulator. Internal simulators are implemented inside the server which currently supports deterministic, stochastic, and hybrid algorithms, while external simulators are defined by the user and dynamically linked to the running server.

The steering application programming interfaces (APIs) are used to incorporate an external simulator into the steering server. Additional responsibility of the API library is to facilitate the connections between the different framework components. More specifically, it is used to carry out the communication between the steering GUI and the steering server.

Finally, this versatile framework permits the simulation to be executed remotely using an external simulator developed by the user (optional component). The communication between this external simulation modules and the other architecture components takes place through the steering APIs. This means that with modest effort, users can include their own favourite simulators and perform the monitoring and steering tasks by help of the other framework components.

2.2 Steering Server

At the core of the architecture is the steering server. To support true collaboration between different systems biologists, the steering server is designed to be multi-user, multi-threaded, multi-model and multi-simulator. The server records internally information about users, model specification, as well as Petri net definition. Moreover, the server is shipped with a default set of simulators to permit the simulation of complex biological pathways without any additional components.

Multi-users feature allows for more than one user to simultaneously share the same model and collaboratively steer the running simulation to get more insights of the problem under study. Computational steering could promote knowledge sharing between users of different background [13]. Furthermore, multi-models and multi-threaded features coupled by multi-simulators capabilities of Snoopy render the concurrent execution of multiple models and flexible switching between different intermediate results.

The primary building block of the steering server is the user model. Users submit their models remotely to the steering server and permit others to use them. A model consists of the Petri net definition, user views, simulation result, and the currently allocated simulator to produce model dynamics. Thus, models inside the steering server are defined in terms of Petri nets, which in turn are specified by places, transitions, parameters and the connections between places and transitions. More elaborated tutorials of how to use Petri nets to model biochemical networks can be found in [5]. The internal representation of the server data structures correspond to the graphical representation of the biochemical networks. The model kinetics are specified by the transition rates, while the initial state is represented by initial place markings. Additionally, each model has a set of views associated with it. Views are manipulated by users using the steering GUI and submitted to the server for further download by other users. The main functionalities of model views is to give users the opportunity to monitor simulation results from different perspectives with different settings. Moreover, intermediate and final results of the simulator are maintained and viewed by collaborative users on their terminals. Finally, each model has its own simulator associated with it. Model simulator runs independently from other simulators which are simultaneously running on the server.

Furthermore, different users can connect and disconnect from running servers without affecting the other connected users and running models. Moreover, data coherence is maintained transparently from the users by using internally synchronised objects. Users can share the same model simultaneously and learn from each others through steering of model parameters.

The steering process takes place through an internal scheduler of the steering server. Each model has its own scheduler which coordinates the operation of the associated simulator. When a user submits a remote command from the GUI client, the current model scheduler adds this command to what is called TO-DO list. Later on, when the simulator is ready to dispatch this command, the command is executed and its effect is displayed to peer users. The steering

commands can be: altering model parameters, altering place marking, restarting, pausing, stopping the simulator, etc. The reason behind such an organisation is that we can not steer the biochemical simulation at any point of execution, we have to wait for a suitable time point before the change can take place. Furthermore; using such an approach, the simulator does not need to wait for the user input and accordingly eliminates the delay due to the incorporation of computational steering into the simulation algorithm. The appropriate time point of a change depends on the simulation algorithm (i.e., continuous, stochastic, or hybrid algorithm). For instance; appropriate time points to change in continuous simulation is between integration steps of the ordinary differential equations. In case of conflicts between different users sending the same steering command to the same running model at the same simulation time point (e.g., two users want to change model parameters at time 20), only the latest command will take effect and consequently other users are informed of such situation.

Nevertheless, the aforementioned issues should rather be kept hidden from the user. Users might view the server as a simulator which produces model dynamics. All of the interactions between the user and the steering server are carried out using the graphical user interface. Accordingly, it does not matter from users point of view, where the steering server is located. Moreover, in case of legacy code, there is no direct relationship between the user and the external simulator. Instead, the steering server plays the role of mediator between the user interface and the external data source.

2.3 Graphical User Interface

The ultimate goal of the Steering GUI is to provide the user with a remote control-like facility to interact with the currently running models. The connection between the steering GUI and the steering server is dynamically established, meaning that a connection does not need to be established in advance before the simulation start.

Among the helpful features that Snoopy's steering GUI provides are: viewing the running models inside a remote server, selecting between different simulator algorithms, changing the simulation key parameters and the current Petri net marking, providing the user with different views of the simulation results including intermediate and final outputs, and remotely changing the simulator properties. Figure 2 provides a screenshot of Snoopy's steering GUI.

In a typical application scenario, a user constructs the biochemical reaction network using a Petri net editing tool (e.g., Snoopy). Afterwards, the Petri net model is submitted to one of the running servers to quantitatively simulate it. Later, other users can adapt their steering GUIs to connect to this model. One of the connected users initialises the simulation while others could stop, pause, or restart it. When the simulator initially starts, it uses the current model settings to run the simulation. Later, other users can remotely join the simulation and change model parameters and the current marking. See Figure 3 for a graphical illustration of such an application scenario.

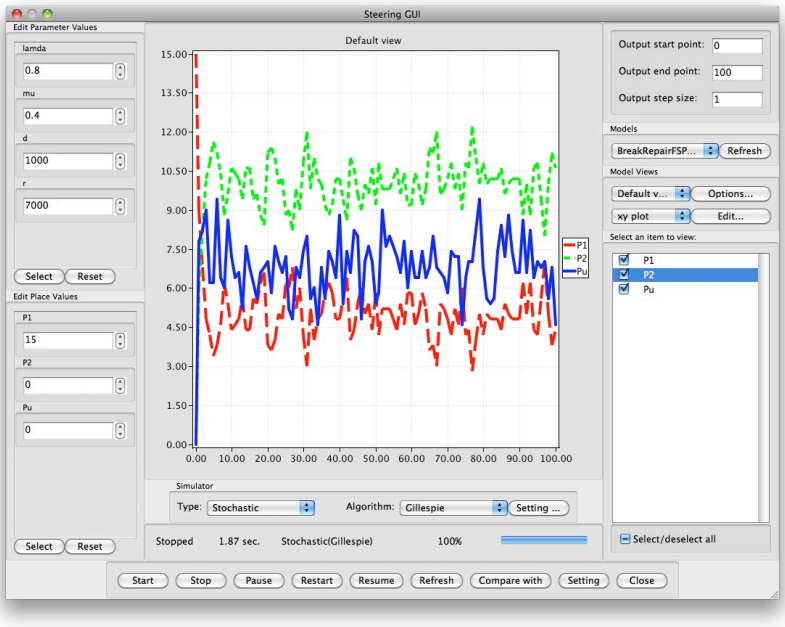

Fig. 2. Snoopy's steering GUI: steering panel (left), output panel (middle), control panel (bottom) and manipulation panel (right). The user can select a subset of the model parameters to change their values during the simulation. The output can be viewed as table, xy plot, or histogram plot. The model result could be exported in csv or image format.

2.4 Application Programming Interface

To keep the computational steering framework simple, yet extendable; an API library will be of paramount importance. Modern software permits users to extend existing capabilities by adding new features or improving existing ones. Such extensions could be deployed using, e.g., plug-in or API calls. For our purpose, we adapt the concept of APIs to provide involved functionalities to advanced users. The main roles of the API library in our framework are: extension of the introduced framework to include additional simulators, communication between different framework components, and user ability to design a new user interface as well as visualisation modules that are compatible to communicate with other components. Figure 4 illustrates the different classes of our implementation of the steering API library.

While our implementation of the framework comes with a set of full-fledged simulators (see Section 2.5), it is possible for users to have their own simulation code included in the framework of Snoopy. Snoopy's steering API library renders it possible to convert such batch simulation code into an interactive one.

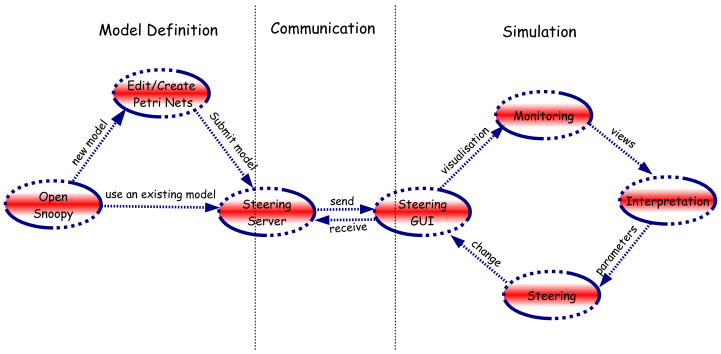

Fig. 3. Application scenario of using Snoopy computational steering framework

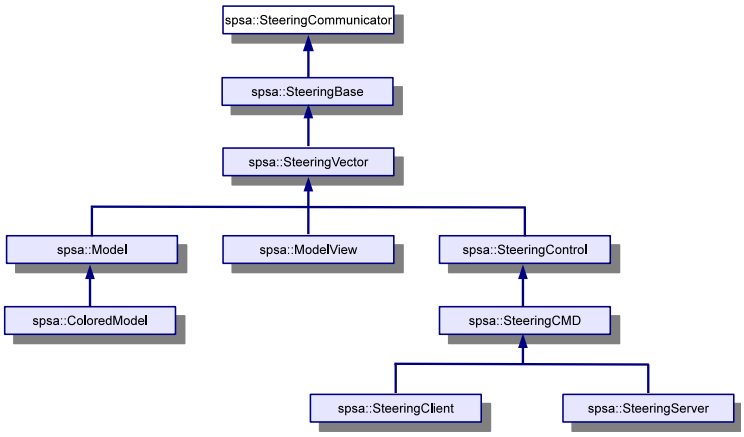

Fig. 4. Inheritance diagram of Snoopy's steering API (SPSA)

Furthermore, the API library makes the entire design of the framework easy to be implemented and simultaneously promotes the reuse of existing code. For instance, the steering server and the steering GUI use the same API library to communicate with each other. Additionally, users are not restricted to use the same user interface illustrated in Figure 2; instead, they could implement their own dialogs and use their favourite visualisation libraries. The existence of such an API library ensures that the newly designed GUI is compatible to communicate with other framework components.

In terms of functionality, Snoopy's steering API can be grouped into four components: communication, data structures, control commands, and end point components. The communication part provides an easy-to-use and portable tool to send and receive data between two connected entities. The provided APIs could send and receive simple data types (e.g., integer, double, or character),

or compound types (e.g., 1D vector or 2D vectors). Moreover, the API library provides two special data structures to help organising the simulation code inside clients and servers: models and model views. The former one is used to encapsulate the Petri net model including all the necessary information about places, transitions, and arcs, while the latter data structure facilitates the organisation of model views inside individual models. Models could contain multiple views to provide more elaborated ways of understanding the simulation results. Please notice that models can be extended to include coloured information as illustrated in Figure 4.

Generally speaking, computational steering provides three main functionalities: simulation, visualisation, and control. The control commands enable the user to start, restart, stop, and pause the simulation. They provide a way to manage the remotely running simulation. Additionally, changing parameter values or asking to refresh the current simulation output is also considered as a steering command.

Finally, the overall framework can be viewed as consisting of two communicating entities: clients and servers. Clients issue requests and servers reply to these queries. The API library supports the implementation of those two entities by providing two classes: spsa::SteeringServer and spsa::SteeringClient (compare Figure 4).

2.5 Simulators

To combine both extendibility and ease of use, the proposed framework comprises two types of simulators: internal and external simulators.

Internal simulators are implemented as part of the steering server. No additional work is required to use them directly from Snoopy. Currently, three categories of simulation approaches are implemented: continuous, stochastic and hybrid [8]. Under each category, some specific algorithms are provided. For instance, under continuous simulation, users can select from simple fixed-step-size unstiff solvers (e.g., Euler) to more sophisticated variable-order, variable-step, multi-step stiff solvers (e.g., Backward Differentiation Formulas [9]), or hybrid simulation with either static or dynamic partitioning [8]. Snoopy provides steering commands to all of these algorithms. See [6], for more information about simulating continuous, stochastic and hybrid Petri nets using Snoopy.

External simulators are developed by the user to implement a particular simulation algorithm or to reuse existing code. In the latter case, the simulation code may be maintained and debugged for a long period of time. Trying to build it from scratch as an interactive one, will require substantial amount of work. Integrating such code into Snoopy's computational steering framework will save the user precious time and perform the required task.

When an external simulator is integrated into the framework, the simulation code and the server will share the same memory space which in turn saves communicating the simulation results from/to the running server. The API library supports the registration of the simulation data to the server which could later be used to accommodate the GUI requests.

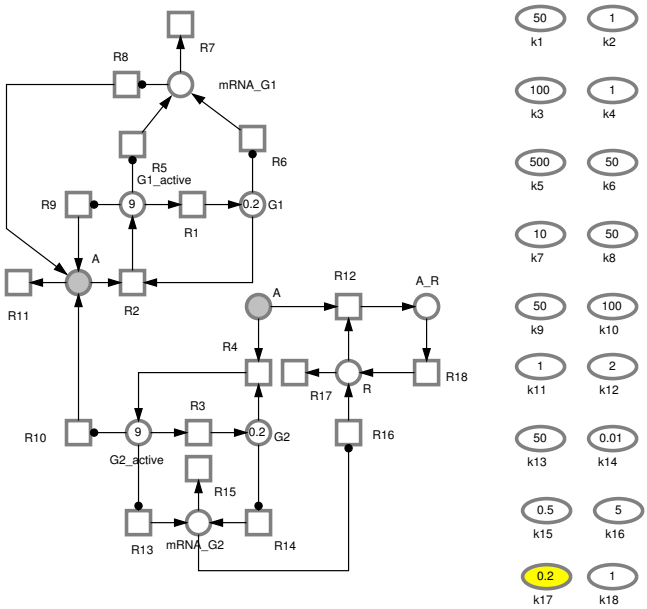

Fig. 5. A Petri net representation of the circadian oscillation model. Places given in grey are logical places and values in ellipses are kinetic parameters. The highlighted parameter, k17, is a key parameter in this model.

3 Case Study

Generally speaking, there are many potential applications of computational steering in the context of kinetic modelling of biochemical networks. Among these application scenarios are oscillation, steady state analysis, parameter estimation and bifurcation analysis. For instance, while the system is in a steady state, the user can perturb it by changing some parameter values and monitor how the model reacts to such changes. In this section, we consider one of these applications as a case study to illustrate the operations of the proposed framework, namely the circadian oscillation model.

In some organisms, there is a control mechanism which is responsible for ensuring a periodic oscillation of certain molecular species [7]. This phenomenon is known as circadian rhythm and it can be found in many organisms (e.g., Drosophila). In this example, we consider a simple model [11] which demonstrates such oscillation. The model consists of two genes which are represented in Figure 5 by two places G_1 and G_2. It includes also one activator and one repressor which are represented by the places A and R, respectively. The activator and repressor control the two genes and their mRNAs, mRNA_G_1 and mRNA_G_2. A and R can be activated to form a complex A_R which takes place through reaction R_{12}.

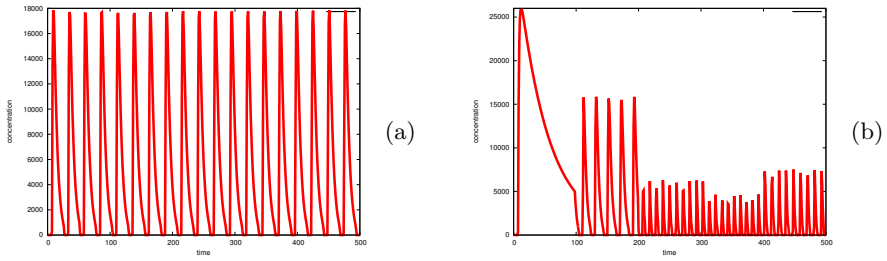

Fig. 6. Simulation results of the circadian oscillation model. (a) $K17$=0.2 is used throughout the whole simulation time. (b) Different values of $K17$ are used. $K17$=0.02 is used through time points 0 - 100, and the model does not produce any oscillation, while for time points 100 - 200, 200 - 300, 300 - 400, 400 - 500, values of $K17$ equal to 0.3, 2, 3, 4, respectively.

In previous studies (e.g., [7], [11]), it has been shown that there are some key parameters of this model that control its oscillation. Among these parameters is $K17$. Using small values of $K17$ (e.g., less than 0.08), the model does not produce oscillation when it is simulated deterministically (see [8] for the other simulation approaches), while the model exhibits oscillation for other values of $K17$ (e.g., greater than 0.1). This example demonstrates how computational steering can be intuitively used to study the effects of repetitively changing $K17$ during the simulation after the model has been formally defined as shown in Figure 5.

Figure 6a gives simulation results of the model defined in Figure 5 for $K17 = 0.2$. Testing other values of $K17$ would require repeating the computational experiment from scratch using the traditional simulation approach. Figure 6b shows effects of changing the value of $K17$ five times during the simulation. For computational intensive models, computational steering will inevitably reduce the overall required time to carry out such different experiments.

For stress testing of our implementation of the proposed framework, a colored version of the model has been used (see [14] for more details of creating a colored version of a low level-Petri net). The created colored Petri net represents $50,000$ copies of the one in Figure 5; it consists of $450,000$ places, $900,018$ transitions and $1,750,035$ arcs.

4 Conclusion and Future Work

In this paper, we have introduced a framework for combining computational steering and Petri nets to model and simulate biochemical networks. The proposed architecture consists of four interdependent components which can run on the same computer or could be distributed across different machines. Our implementation of the steering GUI is provided as part of Snoopy which can be downloaded from `http://www-dssz.informatik.tu-cottbus.de/snoopy.html`. The steering server is available upon request. Moreover, this paper pro-

poses new features of biochemical kinetic modelling software to support interactivity.

The current framework could be extended from different perspectives to provide more features to the user. Condition-based steering is one of these extensions. Condition-based steering means that the user defines some conditions and their corresponding response before the simulation starts. Later, during the simulation, the simulator checks them and if one or more conditions hold, the corresponding actions take place. It is similar to scheduled transitions which have been presented in [5]. However, condition-based steering will give more flexibilities to include other rules which can not be implemented on the Petri net level.

The implementation of the steering server could be extended to add some security rules to prevent unauthorised users to access it. Moreover, users roles could be defined on the model base (i.e., which user can access which model and the type of access).

Backtracking is another important aspect to permit the user to rollback to a previous path of the simulation. This feature will enhance the user interaction of biochemical simulation and consequently give the user better understanding of the problem under consideration.

Acknowledgments

Mostafa Herajy is supported by the GERLS (German Egyptian Research Long Term Scholarships) program, which is administered by the DAAD in close cooperation with the MHESR and German universities. The authors would like also to thank Christian Rohr for his valuable comments during the implementation.

References

1. Bazilevs, Y., Marsden, A., di Scalea, F.L., Majumdar, A., Tatineni, M.: Toward a computational steering framework for large-scale composite structures based on continually and dynamically injected sensor data. Procedia Computer Science 9(0), 1149 – 1158 (2012)
2. David, R., Alla, H.: Discrete, Continuous, and Hybrid Petri Nets. Springer (2010)
3. Funahashi, A., Matsuoka, Y., Jouraku, A., Morohashi, M., Kikuchi, N., Kitano, H.: CellDesigner 3.5: A Versatile Modeling Tool for Biochemical Networks. Proceedings of the IEEE 96(8), 1254 – 1265 (2008)
4. Geist, G., Kohl, J., Papadopoulos, P.: CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications. International Journal of High Performance Computing Applications 11(3), 224–236 (1997)
5. Heiner, M., Gilbert, D., Donaldson, R.: Petri nets for systems and synthetic biology. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.) Formal Methods for Computational Systems Biology, Lecture Notes in Computer Science, vol. 5016, pp. 215–264. Springer Berlin / Heidelberg (2008)
6. Heiner, M., Herajy, M., Liu, F., Rohr, C., Schwarick, M.: Snoopy – a unifying petri net tool. In: Haddad, S., Pomello, L. (eds.) Application and Theory of Petri

Nets, Lecture Notes in Computer Science, vol. 7347, pp. 398–407. Springer Berlin / Heidelberg (2012)
7. Hellander, A., Lötstedt, P.: Hybrid method for the chemical master equation. J. Comput. Phys. 227, 100–122 (2007)
8. Herajy, M., Heiner, M.: Hybrid representation and simulation of stiff biochemical networks. Nonlinear Analysis: Hybrid Systems 6(4), 942–959 (2012)
9. Hindmarsh, A., Brown, P., Grant, K., Lee, S., Serban, R., Shumaker, D., Woodward, C.: Sundials: Suite of nonlinear and differential/algebraic equation solvers. ACM Trans. Math. Softw. 31, 363–396 (2005)
10. 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 22, 3067–74 (2006)
11. Jose, J., Hao, H., Naama, N., Stanislas, S.: Mechanisms of noise-resistance in genetic oscillators. Proceedings of the National Academy of Sciences of the United States of America 99(9), 5988–5992 (2002)
12. Kitano, H.: Systems Biology: A Brief Overview. Science 295(5560), 1662–1664 (2002)
13. Kitano, H., Ghosh, S., Matsuoka, Y.: Social engineering for virtual 'big science' in systems biology. Nature chemical biology 7(6), 323–326 (2011)
14. Liu, F.: Colored Petri Nets for Systems Biology. Ph.D. thesis, Brandenburg University of Technology Cottbus - Computer Science Institute (2012)
15. Mann, V., Matossian, V., Muralidhar, R., Parashar, M.: DISCOVER: An environment for Web-based interaction and steering of high-performance scientific applications. Concurrency and Computation: Practice and Experience 13, 737–754 (2001)
16. Matsuno, H., Tanaka, Y., Aoshima, H., Doi, A., Matsui, M., Miyano, S.: Biopathways representation and simulation on hybrid functional Petri net. In silico biology 3(3) (2003)
17. Modi, A., Long, L., Plassmann, P.: Real-Time Visualization of Wake-Vortex Simulations using Computational Steering and Beowulf Clusters. In: the Fifth International Conference on Vector and Parallel Processing Systems and Applications (VECPAR). pp. 787 – 800 (2002)
18. Moraru, I., Schaff, J., Slepchenko, B., Blinov, M., Morgan, F., Lakshminarayana, A., Gao, F., Li, Y., Loew, L.: Virtual Cell modelling and simulation software environment. IET Syst Biol. 2(5), 352–62 (2008)
19. Pickles, S., Haines, R., Pinning, R., Porter, A.: A Practical Toolkit for Computational Steering. Phil Trans. R. Soc. 363, 1843–1853 (2005)
20. Ramsey, S., Orrell, D., Bolouri, H.: Dizzy: stochastic simulation of large-scale genetic regulatory networks. J Bioinform Comput Bio 3(2), 415–36 (2005)
21. Shu, J., Watson, L., Ramakrishnan, N., Kamke, F., Deshpande, S.: Computational Steering in the Problem Solving Environment WBCSim. Engineering Computations 28(7), 888 – 911 (2011)