Computational Steering of Multi-Scale Biochemical Networks

Von der Fakultät für Mathematik, Naturwissenschaften und Informatik
der Brandenburgischen Technischen Universität Cottbus

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
(Dr. rer. nat.)

genehmigte Dissertation

vorgelegt von

M.Sc. Computer Science
Mostafa Herajy

geboren am 17. 11. 1980 in Aswan, Ägypten

Gutachter: Prof. Dr.-Ing. Monika Heiner

Gutachter: Prof. Dr. rer. nat. habil. Wolfgang Marwan

Gutachter: Prof. Dr. Beih S. El Desouky

Tag der mündlichen Prüfung: 29.01.2013

Eidesstattliche Erklärung

Hiermit erkläre ich Eides statt, dass ich die vorliegende Arbeit selbständig verfasst und angefertigt habe, nur die angegebenen Quellen und Hilfsmittel und keine anderen benutzt bzw. verwendet habe und die wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Die Veröffentlichung der Dissertation verletzt keine bestehenden Schutzrechte.

Mostafa Herajy

Acknowledgements

Unlike the other parts of this thesis, words cannot express what I would like to say in this section. My feeling towards the persons who contributed to my education and my life until I have reached this moment is much more than just saying thank you to them. However, I will try to do it.

First of all I would like to express my sincere acknowledgement to my supervisor Prof. Dr.-Ing Monika Heiner for giving me this research opportunity and for her support and advice during this work.

I would like to thank Prof. Dr. rer. nat. habil. Wolfgang Marwan and Prof. Dr. Beih El Desouky for their careful reviews and valuable comments. I also thank all the other members in my defence committee.

Next, I am very grateful to all members in the Data Structure and Software Dependability Chair for their daily help, assistance and discussions, including Mrs Sigrid Schenk, Christian Rohr, Martin Schwarick, and my former office-mate Fei Liu. Special thank goes to Mary Ann Blätke from the Magdeburg group.

Further, my warmest thanks belong to my family for their support, particularly, my wife and my daughter Amany. I am grateful to all the other people and friends who helped me during this work.

I should also like to thanks BTU and the institute of computer science for providing me with all the required facilities for conducting this research.

Last but not least, I would like to acknowledge the financial support of the GERLS (German Egyptian Research Long Term Scholarships) program, which is administered by the DAAD in close cooperation with the MHESR (Ministry of Higher Education and Scientific Research) and German universities.

All praise and thanks be to Allah

Cottbus, den 29.01.2013
Mostafa

Abstract

Computational steering is an interactive remote control of a long running application. The user can adopt it to adjust the simulation parameters on the fly. Correspondingly, simulation of large scale biochemical networks is computationally expensive, particularly stochastic and hybrid simulation. Such extremely intensive computations necessitate an interactive mechanism to permit users to try different paths and ask simultaneously "what-if" questions while the simulation is in progress.

Furthermore, with the progress of computational modelling and the simulation of biochemical networks, there is a need to manage multi-scale models, which may contain species or reactions at different scales (called also stiff systems). In this context, Petri nets are of considerable importance in the modelling and analysis of biochemical networks, since they provide an intuitive visual representation of reaction networks.

The contributions of this thesis are twofold: firstly, we introduce the definition and present simulation algorithms of Generalised Hybrid Petri Nets ($\mathcal{GHPN}_{bio}$) to represent and simulate stiff biochemical networks where fast reactions are represented and simulated continuously, while slow reactions are carried out stochastically. $\mathcal{GHPN}_{bio}$ provide rich modelling and simulation functionalities by combining all features of Continuous Petri Nets ($\mathcal{CPN}$) and Extended Stochastic Petri Nets ($\mathcal{XSPN}$), including three types of deterministic transitions. Moreover, the partitioning of the reaction networks can either be done off-line before the simulation starts or on-line while the simulation is in progress.

Secondly, we introduce a novel framework which combines Petri nets and computational steering for the representation and interactive simulation of biochemical networks. The main merits of the framework proposed in this thesis are: the tight coupling of simulation and visualisation, distributed; collaborative; and interactive simulation, and intuitive representation of biochemical networks by means of Petri nets.

Generalised hybrid Petri nets and computational steering will together provide an invaluable tool for systems biologists to help them to obtain a deeper system level understanding. $\mathcal{GHPN}_{bio}$ speed up the simulation and simultaneously preserve accuracy, while computational steering enables users of different background to share, collaborate and interactively simulate biochemical models. Finally, the implementation of the proposed framework is given as part of Snoopy - a tool to design and animate/simulate hierarchical graphs, among them qualitative, stochastic, continuous and hybrid Petri nets.

Keywords Systems Biology; Computational Steering; Interactive Simulation; Generalised Hybrid Petri Nets; Hybrid Modelling of Biochemical Networks; Static and Dynamic Partitioning

Zusammenfassung

Computational Steering ist eine interaktive Fernsteuerung von Applikationen mit langer Laufzeit. Der Nutzer kann sie einsetzen, um Paramater "on-the-fly" einzustellen. Die stochastische und hybride Simulation von biochemischer Netzwerk ist sehr rechenintensiv. Derart aufwendige Berechnungen erfordern interaktive Techniken, die es Nutzern ermöglichen, unterschiedliche Ausführungen während der Berechung zu testen.

Durch die rasant fortschreitende Entwicklung der rechnergestützten Modellierung und Simulation biochemischer Netzwerke besteht zunehmender Bedarf, Modelle, in denen Substanzen und Reaktionen unterschiedlicher Skalierung (multi-scale models) auftreten, zu verwalten. Dabei sind Petrinetze von besonderer Bedeutung, da sie eine sehr intuitive visuelle Darstellung von Reaktionsnetzwerken erlauben.

Die vorliegende Arbeit liefert folgenden Beitrag: Zunächst werden verallgemeinerte hybride Petrinetze ($\mathcal{GHPN}_{bio}$) und deren Simulation vorgestellt, um sogenannte "steife" (engl. stiff) biochemische Netzwerke zu modellieren und zu simulieren. Schnelle Reaktionen werden dabei kontinuierlich behandelt, langsame Reaktionen dagegen werden stochastisch behandelt. Durch die Kombination der Eigenschaft von kontinuierlichen Petrinetzen ($\mathcal{CPN}$) und erweiterten stochastischen Petrinetzen ($\mathcal{XSPN}$) bieten $\mathcal{GHPN}_{bio}$ ein hohes Maßen Ausdruckstärke hinsichtlich Modellierung und Simulation. Die Zuordnung der Transitionen zu kontinuierlichen oder stochastischen (Partionierung) kann dabei sowohl statische als auch dynamisch während der Simulation vorgenommen werden.

Daräberhinaus wird ein neues Framework vorgestellt, das Petrinetze und Computational Steering zum Zweck der Darstellung und interaktiven Simulation biochemischer Netzwerke zusammenfährt. Die wesentlichen Besonderheiten sind: die enge Kopplung zwischen Simulation und Visualisierung, die verteilte; kooperative; und die interaktive Simulation und die intuitive Repräsentation biochemischer Netze.

Zusammen stellen verallgemeinerte hybride Petrinetze und Computational Steering für Systembiologen ein nützliches Werkzeug dar, das helfen kann, komplexe Zusammenhänge auf Systemebene zu verstehen. $\mathcal{GHPN}_{bio}$ können dazu verwendet werden, die Simulation biochemischer Netze ohne Genauigkeitsverlust zu beschleunigen. Computational Steering erlaubt es Benutzern mit unterschiedlichem fachlichen Hintergrund biochemische Modelle gemeinsam zu bearbeiten und zu simulieren. Das vorgeschlagende Framework wurde in unserem Modellierungswerkzeug Snoopy implementiert.

Freie Schlagwörter Systembiologie; Computational Steering; interaktiven Simulation; verallgemeinerte hybride Petrinetze; hybride Modellieren von biochemischer Netzen

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols

R_j	Chemical reaction
S_i	Chemical species
N	Number of chemical species
M	Number of chemical reactions
k_j	Reaction rate constant in the continuous setting
c_j	Reaction rate constant in the stochastic setting
$\mathbf{v}_j$	State-change vector of a reaction R_j
$\mathbf{X}_i(\tau)$	System state at time τ
τ_0	Initial time
τ	Current time
[]	Concentration of chemical species
v	reaction velocity
V_{max}	maximum reaction velocity
K_m	Michaelis constant
K_{cat}	The turnover number
v_{ji}	Change in the number of molecules of S_i due to the occurrence of the reaction R_j
α_{ji}	Stoichiometric coefficient of species S_i when participating in reaction R_j
$a_j(\mathbf{x})$	Propensity of a reaction R_j at state $\mathbf{X}(\tau) = \mathbf{x}$
$P_{\mathbf{x}}(\tau)$	Probability that at time τ the system will be in a state $\mathbf{X}(\tau) = \mathbf{x}$
$\delta\tau$	The next time a stochastic reaction to occur
$a_0(\mathbf{x})$	Total (cumulative) propensity
r_i	Random Number
$\tau_{j'}$	Putative firing time
$O()$	Time-complexity function of an algorithm
$n_j(\delta\tau; \mathbf{x}, \tau)$	Number of times a reaction R_j will fire in the time interval $[\tau, \tau + \delta\tau]$
h	ODE integrator step-size
ξ	Random number exponentially distributed with a unit mean
$a_0^s(\mathbf{x})$	Cumulative propensity of slow reactions
p_i	Place
t_j	Transition

$m(p_i)$	Current marking of a place p_i
${}^\bullet t_j$	Set of pre-places of a transition t_j
$t_j^\bullet$	Set of post-places of a transition t_j
${}^\bullet p_i$	Set of pre-transitions of a place p_i
$p_i^\bullet$	Set of post-transitions of a place p_i
$\mathbb{N}$	Set of positive integer numbers
$\mathbb{N}_0$	Set of non-negative integer numbers
$\mathbb{R}_0^+$	Set of non-negative real numbers
$\mathbb{Q}^+$	Set of positive rational numbers
m_0	Initial marking
P	Set of places
T	Set of transitions
A	Set of arcs
$F()$	Mathematical function
$m[t_j\rangle]$	Transition t_j is enabled on the current making m
V	Set of rate functions
H_s	Set of all stochastic hazard functions
H_w	Set of all weight functions
P_{disc}	Set of discrete places
P_{cont}	Set of continuous places
T_{stoch}	Set of stochastic transitions
T_{im}	Set of immediate transitions
T_{timed}	Set of deterministically delayed transitions
$T_{scheduled}$	Set of scheduled transitions
T_{cont}	Set of continuous transitions
A_{cont}	Set of continuous arcs
A_{disc}	Set of discrete arcs
A_{read}	Set of read arcs
$A_{inhibit}$	Set of inhibitor arcs
A_{equal}	Set of equal arcs
A_{reset}	Set of reset arcs
$A_{modifier}$	Set of modifier arcs

List of Abbreviations

$\mathcal{CPN}$	Continuous Petri Nets
$\mathcal{GHPN}_{bio}$	Generalised Hybrid Petri Nets
$\mathcal{QPN}$	Qualitative Petri Nets
$\mathcal{SPN}$	Stochastic Petri Nets
$\mathcal{XSPN}$	Extended Stochastic Petri Nets
API	Application Programming Interface
BDF	Backward Differentiation Formula
CDK	Cyclin-Dependent protein Kinases
CME	Chemical Master Equation
DNA	Deoxyribonucleic Acid
DPN	Differential Petri Nets
FOHPN	First-Order Hybrid Petri Nets
FSPN	Fluid Stochastic Petri Nets
GUI	Graphical User Interface
HDN	Hybrid Dynamic Nets
HFPN	Hybrid Functional Petri Nets
HFPNe	Hybrid Functional Petri Nets with extension
HPN	Hybrid Petri Nets
M phase	Mitosis phase
mRNA	Messenger Ribonucleic Acid
ODEs	Ordinary Differential Equations
S phase	Synthesis phase
SBML	Systems Biology Markup Language
SSA	Stochastic Simulation Algorithm

SSServer	Snoopy Steering Server

1 Introduction

1.1 Overview

With the progress of molecular biology, *systems biology* has been recently gaining renewed interest to examine the structure and dynamics of cellular and organismal functions [Kit02]. To obtain such system-level understanding, we need new 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 the simulation while it is running. Furthermore, with the advances of computing powers, 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 emerging technique which can inevitably contribute in achieving these goals.

Computational steering [MWL99] is the tight coupling of visualisation and simulation. Through it, the user can change the simulation parameters simultaneously while the simulation is in progress. Further, it is an emerging technology for interactive applications. In other words, computational steering can be described as a remote control of a long running application [PHPP05]. 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.

On the one side, many well known software have recently been introduced to simulate biochemical networks (e.g., COPASI [HSG$^+$06], Dizzy [ROB05], Cell Illustrator [NSJ$^+$10] and Cell Designer [FMJ$^+$08]), however they ignore some useful features such as: collaboration, distribution and interactivity, which are helpful for system biologists. Other web-based applications (e.g., Virtual Cell [MSS$^+$08]) support some kind of collaboration and distribution. Nevertheless, using those software, users are not aware of what is happening in the background, which makes the entire simulation process look like a black box. Systems biologists need to interact with their experiments and with each others by changing some key parameters and asking "what-if" questions. Furthermore, such applications are based on the batch approach that keeps the users away from their experiments during the simulation [SWR$^+$11]. Our aim is to overcome these limitations by integrating computational steering with the simulation of biochemical reactions. Accordingly, using computational steering, systems biologists could drive and guide the simulation in the direction of desirable output by monitoring intermediate

results and adapt the 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 [GKP97], CSE [LMW96], RealityGrid [PHPP05], POSSE [MLP02], DISCOVER [MMMP01, LJPS05], and SCIRun [PJ95]), however they are used to build new applications or require modifications to the source code of the application. With other words, they assume the existence of legacy simulation code which needs to be integrated in such environments [SWR+11]. Thus, they are inapplicable if the source code is unavailable. Additionally, they have a steep learning curve [MLP02, PHPP05] which makes them unsuitable for many systems biologists.

Correspondingly, Petri nets [Mur89, DA10] have been proven to be useful in modelling biochemical networks [RML93, MTA+03, GH06, GHL07], since they provide an intuitive graphical representation and a well-developed mathematical theory for process analysis. In addition, Petri nets might bridge the gap between computational theoretician and experimentalist. In this thesis, we are specifically interested in quantitative Petri nets [GHL07] (stochastic, continuous and hybrid Petri nets) and their high level representations (hierarchical and coloured Petri nets) [Liu12]. Indeed, integrating Petri nets and computational steering into the same framework will result in a powerful and interactive tool for systems biologists. We obtain the elegant representation and the well mathematical foundation merits of Petri nets along with the interactive capabilities of computational steering.

In this thesis, two points are addressed in the context of *systems biology*: the hybrid modelling of biochemical reaction networks, and the development of a computational steering framework to collaboratively and interactively steer the simulation of reaction networks. In the former point, we introduce the definition of Generalised Hybrid Petri Nets ($\mathcal{GHPN}_{bio}$), while in the latter a novel framework is proposed to integrate Petri nets and computational steering for the modelling and simulation of biochemical reaction networks.

$\mathcal{GHPN}_{bio}$ [HH11, HS12, HH12a] combine the power of continuous Petri nets ($\mathcal{CPN}_{bio}$) [GH06] and extended stochastic Petri nets ($\mathcal{XSPN}_{bio}$) [MRH12]. They are particularly well suited to represent and simulate stiff biochemical networks. The modelling power of this class of Petri nets allows the combination of discrete and continuous network parts in one model, which permits to represent, e.g., a biological switch in which continuous elements are turned on/off by discrete elements. The models can be simulated using either static partitioning, in which the partitioning is done off-line before the simulation starts, or using dynamic partitioning, in which the partitioning is done on-line during the simulation.

Furthermore, $\mathcal{GHPN}_{bio}$ extend the framework that was proposed by Heiner and Gillbert in [GHL07] to include hybrid Petri nets as an intermediate approximation of the stochastic and continuous worlds (for more details see Chapter 4).

Further contributions of this work are the introduction as well as the implementation of a computational steering framework which utilises Petri nets as modelling language

for the representation and interactive simulation of biochemical reactions [HH12b]. Its main features are the tight coupling of visualisation and simulation, distributed; collaborative; and interactive simulation of biochemical networks, and intuitive; and understandable representation of the reaction networks with the help of Petri nets, and the extendibility to include external user simulators. The implementation of this framework is given as part of Snoopy [RMH10, HHL+12] - a tool to design and animate/simulate hierarchical graphs, among them qualitative, stochastic, continuous and hybrid Petri nets. During this thesis we use the term kinetic modelling applications (software) to denote software tools that perform the quantitative modelling of biochemical reaction networks.

Although we focus here on the problem of modelling biochemical reaction networks, the two presented tools can be applied to other scientific disciplines, e.g., modelling of technical systems.

The rest of this chapter presents an overview of the thesis by pinpointing the motivations and objectives of our work. Additionally, it provides a high level organisation of the remaining chapters.

1.2 Motivations

- **The recent renaissance of quantitative modelling of biological systems.**

  Systems level understanding has been recently gaining increasing interest. This renders it possible for biological systems having desired properties to be virtually constructed, modified, and tested using the quantitative modelling and simulation before the actual conduction of the wet-lab experiment. Thus, precious efforts and resources could be inevitably saved in comparison to blind trial-and-error methods [Kit02]. Furthermore, quantitative modelling of biological networks contributes to the in-depth understanding of the complexity of biological systems which in turn will have imperative effects to other important fields. For instance, understanding of the biology that underlies certain diseases will increase the productivity of drug discovery [KGM11].

- **The need for a biologist to directly and easily design a quantitative simulation and its multi-purpose evaluation.**

  Software tool support for systems biologists is crucial for the understanding of the biological model dynamics. Although many tools have been recently deployed in the context of systems biology, they still lack some significant key features such as collaboration, distribution and interactivity. Moreover, some tools are acceptable when they operate on small models, however they become impractical when the model complexity increases. Systems biologists need not only to construct wet-lab experiment using the designed tool, but also they would like to do that

comfortably.

- **The interactive nature of computational steering makes it suitable for implementing the biochemical simulation of wet-lab experiments in a straightforward manner.**

  Interacting with the simulation while it is running, monitoring intermediate result, and sharing models and simulation results between different users are some of the important outcomes of incorporating computational steering into biochemical modelling tools. Traditional biochemical modelling software look like a "black-box" when they perform the simulation. Furthermore, the user is not aware of what is taking place in the background. Users need to monitor intermediate result and perform "what-if" analysis simultaneously while the simulation is in progress. Additionally, collaborative and distributed simulation of biochemical networks will certainly promote sharing of knowledge between different scientists. In this thesis, we argue that computational steering could elegantly be applied to the kinetic modelling of biochemical networks. Moreover, it will certainly facilitate the job of systems biologists if it is implemented efficiently in a software tool.

- **Some biological models require to be represented in a hybrid way (cells/molecular interactions in one model).**

  Traditionally, biological networks are simulated either using the purely deterministic or the purely stochastic approach. However, most of the biological systems are better to be considered in a hybrid way [MTA+03, NDMM04, FMJ+08]. For instance, discrete simulation is of paramount importance when studying the role of intrinsic and extrinsic noises of the eukaryotic cell cycle [KBPT09]. However, some phenomena have to be simulated continuously (e.g., cell growth). This model is considered in more detail in Chapter 6. Through this view, quantitative information as well as discrete states could be mixed together in one model. As another example, consider a biological model that combines continuous operations such as translation and transcription of genetic information and discrete operations such as the control system of gene expression. It can not be intuitively modelled using the deterministic or stochastic approach alone [MNM11]. Therefore, the hybrid modelling approach is of significant importance to study the dynamics of such models. Here, we often use the term continuous simulation to refer the deterministic one since the latter can typically be implemented by constructing and solving a set of ordinary differential equations (ODEs) (see Chapter 3). Hence, the ODE system response is continuous over time.

- **Deterministic simulation does not consider the fluctuation of molecules, especially when there are low numbers of them.**

  The deterministic approach is the conventional way of simulating biochemical pathways. In this approach, reactions and their influence on the concentrations of the involved species are represented by a set of ordinary differential equations (ODEs). While this approach has the advantage of a well established mathematical basis and strong documentation, it fails to capture the phenomena which occur due to the underlying discreteness and random fluctuation in molecular numbers [LCPG08, Pah09], especially in situations where the number of molecules is low. Therefore, deterministic simulation is not adequate when the molecular fluctuation is crucial for the model behaviour. In this case, the result of stochastic and deterministic simulation will not be the same. Hence, in this situation, stochastic results are closer to the model behaviour than the continuous one [Gil07].

- **Stochastic Simulation is computational expensive (fast reactions, large number of molecules).**

  Contrary, stochastic simulation [Gil76, Gil77] provides a very natural way of simulating biochemical pathways, since it can successfully capture fluctuations of the underlying model. Furthermore, it deals correctly with the problem of extremely low number of molecules [Gil07]. In stochastic simulation, species are no longer represented as continuous concentrations which change continuously with time, instead, they are represented as discrete entities such that their dynamics can be simulated using the machinery of Markov process theory.

  A major drawback of the stochastic simulation is that it is computationally expensive, when it comes to simulate larger biological models [MA99, LCPG08, Pah09], especially in some cases where species with large numbers of molecules exist. The reason behind this problem comes from the fact that we have to simulate every reaction event when we use stochastic simulation to simulate biological models [Gil07]. This drawback motivates the search for other methods to enhance the capability of the stochastic approach. Hybrid simulation is one of these methods.

  The last two motivations clearly emphasise the importance of a hybrid tool that could deal with models of more than one time scale: e.g., slow and fast.

1.3 Objectives

- **Integrating stochastic and continuous modelling approaches for the hybrid representation and simulation of stiff biochemical networks.**

  The first objective of this thesis aims to develop a hybrid modelling tool to facilitate the hybrid representation and simulation of biochemical networks. The

resulting apparatus intuitively combines stochastic and continuous parts in the same model and permits multi-scale biochemical networks to be accurately and efficiently simulated. Additionally, cell-molecular interactions could be easily modelled, where cells are modelled as discrete entities while molecular reactions are carried out continuously. The developed tool is based on Petri nets as a visual modelling language of the reaction network. The main motivation of resorting to the hybrid modelling is to reach a compromise between the accuracy of the stochastic simulation and the computation efficiency of the continuous counterpart. More elaborately, using generalised hybrid Petri nets to model biochemical networks, users could increase (decrease) the simulation accuracy by increasing (decreasing) the number of transitions that are treated stochastically.

- **Designing and implementing a computational steering framework which utilises Petri nets for modelling biochemical networks.**

  The second objective is to design and implement a computational steering framework that utilises Petri nets as a modelling language for the representation and interactive simulation of biochemical reactions. The resulting framework consists of four components that are dependent on each other. Moreover, on the one hand; the framework is easy to be used by naive users, since no additional knowledge is required to perform the simulation. On the other hand, it can support experienced users by giving them the opportunity to extend the simulation algorithms according to their needs. The framework is offered to the user as a client/server model, where the server represents the current running simulation, while the client represents the user's remote control that manages remotely running simulations.

- **Testing and validating the proposed framework using a number of case studies.** Finally, we aim to apply the proposed computational steering framework and the developed generalised hybrid Petri nets to some case studies with varying level of complexities. The presented examples illustrate how the two introduced tools operate. These examples are discussed in details in Chapter 6.

1.4 Thesis Outline

Chapter 1: Introduction

Chapter 1 (this chapter) provides an overview of the overall thesis organisation. The motivations and objectives as well as the general outlines are also given in this chapter.

Chapter 2: Computational Steering: An Interactive Simulation Technique

Chapter 2 provides an overview of computational steering as an interactive simulation technique. This chapter starts by comparing computational steering with the traditional visualisation pipeline, then different computational steering approaches and tasks are briefly presented. Finally, the chapter is concluded by discussing current challenges that computational steering faces to be used in the context of biochemical network modelling. As a general view, the applications of computational steering in other areas as well as some of the previously introduced environments are given.

Chapter 3: Simulation Approaches of Biochemical Networks

This chapter presents background information of biochemical networks modelling and simulation using the continuous, stochastic, and hybrid approaches. First, basic definitions are presented, afterwards, different views of biochemical networks are briefly compared with each others as will as their limitations to simulate stiff biochemical networks. To make it self-contained, the hybrid modelling approach is discussed in this chapter in the context of the other two simulation techniques: stochastic and deterministic simulation. Particular attention is given to stiff biochemical networks, since they are intended to be addressed more elaborately using Generalised Hybrid Petri Nets in Chapter 4. With this point in mind, the role of quantitative Petri nets in modelling of reaction networks is outlined. This chapter presents also the relationship between different Petri net classes and the corresponding simulation approaches.

Chapter 4: Generalised Hybrid Petri Nets

This chapter introduces one of the main contributions of the thesis, namely Generalised Hybrid Petri Nets ($\mathcal{GHPN}_{bio}$). The modelling part is given in terms of the graphical notations, connection rules, and the formal definition of $\mathcal{GHPN}_{bio}$. Two related simulation algorithms are presented to simulate $\mathcal{GHPN}_{bio}$ using different approaches: static and dynamic partitioning. Besides, the relationship of qualitative, stochastic, continuous, and generalised hybrid Petri nets is delineated. Afterwards, some of the implementation issues are briefly discussed. Finally, we compare $\mathcal{GHPN}_{bio}$ with other hybrid Petri nets which are being used to model biochemical networks.

Chapter 5: A Computational Steering Framework for Collaborative, Distributed, and Interactive Simulation of Biochemical Networks

The other main contribution of this thesis is introduced in this chapter. Firstly, the requirements and characteristics of a computational steering based kinetic modelling software is propounded. After that, the proposed computational steering framework is elaborately discussed by presenting its main components: the steering server, steering

GUI, steering API, internal and external simulators. Secondly, steering algorithms are discussed for deterministic, stochastic, and hybrid simulation methods. Eventually, the chapter is closed by discussing some of the implementation issues related to the synchronisation and communication of the framework components.

Chapter 6: Case Studies

In this chapter we illustrate the functionality of the developed tools using typical biological examples. Three case studies are used to achieve this goal: the intracellular growth of bacteriophage T7, the eukaryotic cell cycle, and the circadian oscillation model.

Chapter 7: Conclusions and Future Work

Finally, this chapter concludes the thesis by summing up the overall presented information and proposes possible extensions for future work.

2 Computational Steering: an Interactive Simulation Technique

2.1 Introduction

With the advances of computing power and the proliferation of multi-core processors, it becomes essential to execute long running and computationally expensive simulations at powerful and remote computers – which enjoy high speed computational units – to profit from such precious processing resources. However, such powerful computers do not provide a direct interactive visualisation and analysis of the resulting simulation data due to either the intrinsic batch processing approach of these computers or the sharing of their resources between different users. Thus, there is a need to remotely mange and analyse the simulation output traces simultaneously while the simulation is in progress. Correspondingly, many different techniques have been proposed to overcome these limitations [ASM+11]. Computational steering is among the elegant and promising tools that provide a tight coupling between simulation and visualisation modules of scientific models.

Computational steering is an interactive technique that permits the manual and automatic guidance and intervention of long running applications. Through it, the user can change the simulation key parameters and immediately obtain a feedback from the computational modules. Furthermore, it closes the loop of the traditional visualisation pipeline and eventually speeds up the scientific discovery process [MWL99]. In other words, computational steering can be defined as runtime control of an application and of the resources it uses for the purposes of experimenting with the application parameters or improving application performance [VS96]. For recent applications of computational steering see [DWB+12, BMS+12, LR12].

Initially, computational steering was inspired in 1987 by the National Science and Foundation Visualisation in Scientific computing workshop report [DeF87]. At that time, the batch simulation approach was the dominating method for organising scientific simulation code. However, the workshop emphasised that scientists need to be able to interact with and steer their simulation and play an active rather than a passive role. "Scientists not only want to analyse data that results from super-computations; they also want to interpret what is happening to the data during super-computations. Researchers want to steer calculations in close-to-real-time; they want to be able to change parameters, resolution or representation, and see the effects. They want to drive the

scientific discovery process; they want to interact with their data. The most common mode of visualisation today at national supercomputer centres is batch. Batch processing defines a sequential process: compute, generate images and plots, and then record on paper, videotape or film. Interactive visual computing is a process whereby scientists communicate with data by manipulating its visual representation during processing. The more sophisticated process of navigation allows scientists to steer, or dynamically modify computations while they are occurring. These processes are invaluable tools for scientific discovery".

Currently, there are many computational steering environments that are based on the general idea which was proposed by the National Science and Foundation Visualisation in Scientific Computing workshop (e.g., CUMULVS [GKP97], CSE [LMW96], RealityGrid [PHPP05], POSSE [MLP02], DISCOVER [MMMP01, LJPS05], and SCIRun [PJ95]). The differences between these environments are either the intended application, the steering approach or the location of the simulation module. However, they presume or enforce the existence of legacy simulation code which needs to be integrated into the computational steering framework. Moreover, none of these software tools is dedicated to quantitatively simulate biochemical reaction networks. Such quantitative simulation is an imperative task of *systems biology*.

Systems biology, as an emerging inter-disciplinary field, shares some common features with traditional scientific applications (e.g., Computational Fluid Dynamics). For instance, to carry out complex kinetic pathway analysis, we need long running simulations and extensive computational resources. Thus, in this thesis, the potential contributions of computational steering for kinetic modelling software are investigated to remotely and interactively manage and simulate the multi-scale biochemical networks. Indeed, using computational steering, computational expensive biochemical network applications can inevitably benefit from remote computational resources and the "what-if" analysis nature of computational steering which eventually might lead to new discoveries. Moreover, computational steering will locate systems biologists at the centre of their dry-lab experiments.

This chapter is organised as follows: we commence with comparing the conventional non-interactive simulation with the interactive one. In this context, we distinguish between two different yet related interactive techniques: interactive visualisation and interactive simulation. Afterwards, a general overview of the ingredients of a typical computational steering environment is provided. Later, the approaches and tasks of the computational steering are compared with each other. Additionally, this chapter also lists some of the early developed computational steering environments. Finally, we conclude by discussing some challenges that face computational steering to become more popular in the context of scientific simulation, particularly *systems biology*.

2.2 Batch versus Interactive Simulation

The traditional scientific computing approach usually consists of three sequential steps: modelling, computation and visualisation [PJB97, MWL99]. The major drawback of this approach of computation is that: it is a sequential process and does not provide the user with the opportunity to interact with the simulation module, since the data analysis and visualisation stages are done off-line completely after the simulation terminated. Furthermore, because the visualisation of the simulation result is done as a post processing step, errors at the simulation phase may be discovered only during the final visualisation [PJB97, JPH+99], which requires repeating the entire simulation process from scratch and consequently extends the overall experiment time. Errors might result from setting inappropriate model parameters or initial values. Therefore, they are logical errors. Figure 2.1a is a graphical illustration of this approach.

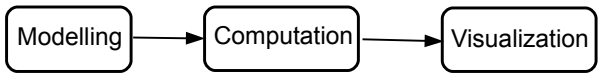
Contrary to this approach, Figure 2.1b illustrates the interactive simulation technique which closes the loop between the presentation of the data and the running simulation. Users are allowed to view intermediate results, interpret them and make changes to the parameter values that are critical to the response of the running model. Typically, the user will change parameters after having interpreted the results. Next, the user change is enacted by the steering application to the running simulation. However, the type of user data is application-dependent. For instance, if the computational steering task is model exploration (see Section 2.4), then the user will be mainly interested in the application input and output. However, in case of performance optimisation, the user will need to view the application structure [MWL99].

Furthermore, the National Science and Foundation Visualisation Report distinguished between two types of interactivity: interactive visualisation and the more sophisticated interactive approach, called computational steering. In the following, these two related approaches are compared to each other.

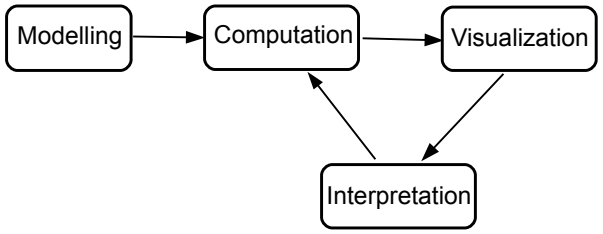
2.2.1 Interactive Visualisation

The classical model of visualisation that was described by Haber and McNabb [HM90] consists of three related steps as it can be noticed from Figure 2.2: filtering – the data quantity is reduced or interpolated, mapping – an abstract geometric object is constructed, and rendering – the final image is produced. This model of visualisation is typically used to implement many open source visualisation packages (e.g., the Visualisation Tool Kit, VTK [VTK12]) and commercial one (e.g., AVS [AVS12]).

In this viewpoint, interactivity could be supported only at the visualisation level by closing the loop between the filtering of the data and its rendering. For instance, if the user wants to view different levels of granularities of the rendered image, a few visualisation parameters could be changed without repeating the overall simulation process.



(a) Batch simulation approach



(b) Interactive simulation approach

Figure 2.1: Two simulation techniques: (a) batch simulation: modelling, computation, and visualisation are done sequentially. Users have to wait until the computation process finish completely in order to observe the final results and correct potential errors. (b) interactive approach: whereby the user can get insights from the simulation while it is running.

Although this technique provides some kind of interactivity, it does not utilise a full interaction between simulation and users since they are still separated from each other. Moreover, the process of visualisation is done off-line which necessitates the reading of the simulation's output from a file. A better view of enhancing such framework is to connect the pipeline directly to the simulation without writing intermediately to a file and to provide two ways of communication between the simulation and visualisation.

2.2.2 Computational Steering

As a further degree of interactivity, computational steering could provide the user with full access to both the simulation and visualisation simultaneously while the simulation is in progress. Such tight coupling of computation and data analysis tools is essential for the deep understanding of what is happening in the background.

Figure 2.3 presents an overview of a typical computational steering architecture. Almost all of the computational steering environments follow this architecture as can be seen in [VS96, PJB97, MWL99], and [MP02b], however, they might differ on how and where these layers and components are implemented. In this figure, we can notice

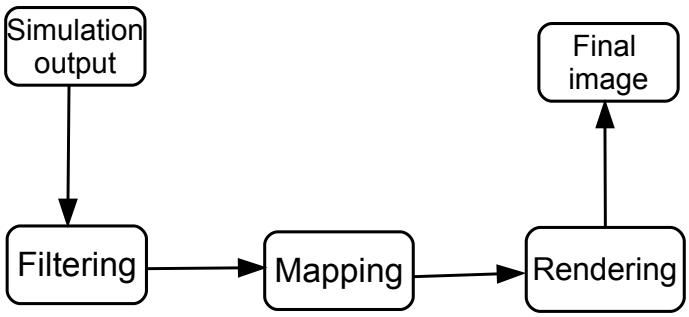


Figure 2.2: Visualisation pipeline: the process of visualisation is done in three sequential steps: filtering, mapping and rendering [CW01].

obviously the closed loop between the user and the simulation. Moreover, three layers of computations could be distinguished: application, steering, and user layers.

Application Layer

The application layer represents the data source unit. This could be either a running simulation which produces the dynamics of a certain model or a parallel application in which its load needs to be adjusted. Usually, the main operation of the application layer is to accept new parameter values and perform a re-calculation of the simulation result or a reconfiguration of the application load. Moreover, the application layer module could be run at the same machine or distributed across different computers.

Steering Layer

As an intermediate layer, the steering layer serves the communication and the synchronisation between the application and user layers. When a user changes some parameter values, these changes are propagated to the running application by means of this layer. Furthermore, other essential and optional functionalities are also supported at this level. An example of the required functionalities is the synchronisation of the user's changes to the application, while checkpointing and optimisation are examples of optional functionalities.

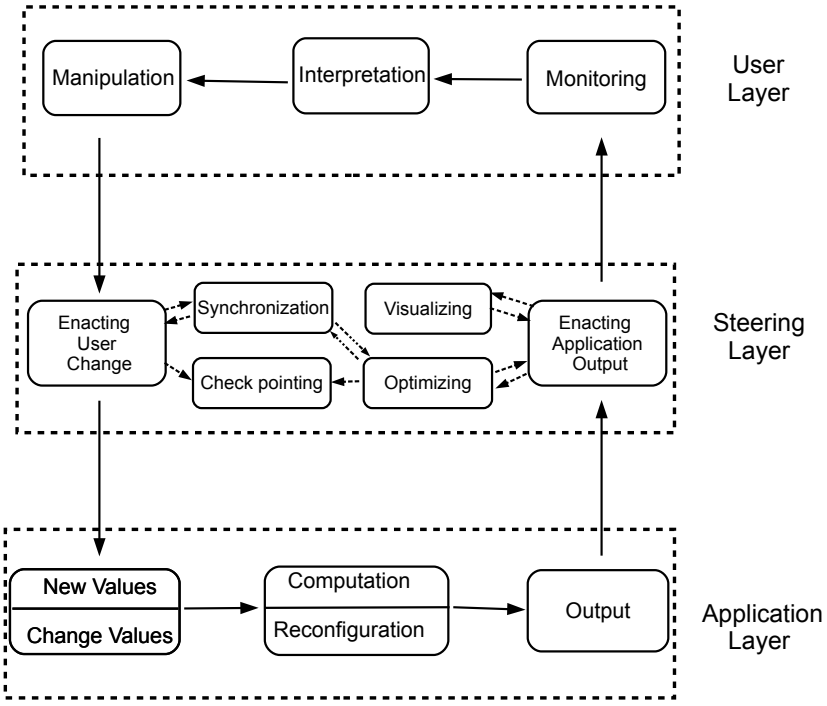


Figure 2.3: Computational Steering Framework

User Layer

At the top layer is the user who will control (steer) the running application. It is worth mentioning here that the user could be a human being or another client application which adjust parameter values based on the application's current state as it was indicated by Vetter et al. [VS96].

Nonetheless, the user performs three related functions: monitoring, interpretation, and steering. During the monitoring phase, the steering agent observes the current application's output and the rendered image that was produced by the application layer, while in the interpretation phase, the results are mined to get a useful hypothesis. Finally, during the steering stage an appropriate action is taken. This action is propagated back to the application and the entire circle is repeated until approaching the final result.

2.3 Approaches

There are different methods to implement computational steering environments, among them are: program annotation, redesigning simulation application, steering by scripting, and high level abstractions. The choice between these approaches is application-dependent. In the following, a detailed discussion of these methods is provided. At the end of this section, a comparison between these approaches is conducted to aid in the selection between them to implement a certain application scenario.

2.3.1 Program Annotation

Program annotation (also called program instrumentation) allows non-interactive simulation applications to be transformed into steerable ones by making a minor modification to the original code. Such modification will provide an access point to the parameters and results to render their steering and visualisation, respectively, by the users [PJB97]. As a typical example of this approach, the simulation code is divided into different parts and injected with a set of application programming interface (API) calls, which are developed as part of a specific computational steering environment. These routines could be better described as a communicator between the simulation and the data analysis modules.

While this technique has the advantage of easy transformation of the batch simulation code into an interactive one with minimal efforts, it does not provide much control over the original application [PJB97]. Moreover, sometimes it becomes inefficient, if the data are being transmitted to a separate visualisation process due to the transmission overhead [MWL99]. More elaborately, the actual data transmission could be implemented in a separate thread. However this adds additional complication to the implementation and introduces many synchronisation problems [Par99]. Algorithm 2.1 provides an illustration of steps of typical instrumentation which are required to integrate existing simulation code into computational steering environments.

At the beginning of these steps, the application registers a number of variables that need to be monitored or steered and gives access to their values. These variables could be of simple data type (e.g., integer or float) or complex variables (e.g., 2-dimensional matrix). Afterwards, the application initialises itself and might require to read some values from the steering agent. As it was previously indicated, the steering agent could be a human being or another program. Later, during the application main loop; a set of API routines are called to execute the commands that are issued by the steering agents.

A wide range of computational steering environments are based on this method (e.g., CUMULVS, Magellan, and POSSE), as it is very easy to integrate existing code into the steering tool and after that the steering library could apply whatever algorithm to the simulation's input/output.

Algorithm 2.1: Example for program annotation approach

```
1: Register the variables which need to be monitored or steered;
2: Read the initial parameters and initial conditions;
3: Initialise the simulation;
4: while the simulation is running do
5:     Compute one step;
6:     Send the intermediate result to the data analysis tool;
7:     Receive the parameter change from the graphical user interface;
8:     Update the simulation parameters using the new values;
9: end while
10: Send the final result to the data analysis tool;
```

2.3.2 Redesigning the Simulation Application

Redesigning the simulation application involves designing the new computational approach with the concept of computational steering in mind [PJB97]. This approach will give the user more freedom to get full utilisation of the computational steering power and it will require the simulation code to be completely rewritten to permit the use of computational steering technique to the problems which are intended to be solved.

An example of the software platforms that use such an approach is SCIRun [PJ95]. It uses a visual programming language to allow the user to construct a simulation model from scratch. It is worth mentioning here that this approach is not always applicable to implement computational steering in all cases, since scientists usually have their own simulation code which they trust due to long maintenance and debug period. Designing a model from scratch will not give them the opportunity to reuse their own code. Thus, some software tools, which follow this approach provide other scenarios to attract users that have their own simulation application.

2.3.3 Steering by Scripting

Another approach to include computational steering into existing scientific simulation code is to divide the whole simulation into various small modules and let the user interacts with them [PJB97]. Using this technique, the output of one module could be used as the input of another one and the change of key parameters might be performed during the in-between modules. Moreover, other modules like data analysis and visualisation could easily be added [Par99]. The interaction between the different modules is carried out by means of a scripting language (e.g., Matlab [Mat12], R [R12], Python [Pyt12]). The advantages of this technique is that it is easy to be implemented and it could use much of the original scientific code. Additionally, experienced users could use the utilities which are provided by a scripting language to

extend their simulation code. Nevertheless, the drawback of this approach is that it is not highly interactive [PJB97], since it does not provide an ultimate easy way for naive users to use such a technique. Moreover, the resulting computational steering architecture will not be intuitive to be used by users which require "press button" and get results.

2.3.4 High-level Abstractions

The high-level abstraction approach [BJBH93, MP02b] is another variation of program annotation. Instead of injecting API calls into the application code, the entire application is viewed as a set of computational objects [MP02b]. In this scenario, an object represents a data structure and its associated algorithm. The real collection of the computational object's resulting data and the enacting steering command are done by inheriting the computational object from a set of C++ classes. The actual implementation of the commands and the extraction of data is carried out by overriding the appropriate member functions. Furthermore, non-object oriented computational application objects could be transformed into object oriented classes using API wrappers. However, not all of the application code will require to be transformed or inherited from the base class, instead, only computational objects which need to be used in the interactive mode will be forced to do such inheritance.

This technique provides a better organisation of the application when it is integrated into a computational steering environment compared with the program instrumentation approach. Moreover, it is adequate for distributed applications which span multiple processors or different memory spaces. In this case, the application object can be migrated, created, or dynamically deleted and it can still support providing information and accepting steering commands.

Nevertheless, this approach still requires the user to modify the source application code which implies that it shares the disadvantages of the aforementioned program annotation method. Moreover, it assumes users with object-oriented programming knowledge which is not always viable.

Examples of software that use this approach are : DISCOVER [MMMP01] and Magellan [VS99].

2.3.5 Selecting the Appropriate Approach

In this part, we summarise the pros and cons of the approaches that have been presented in this section. Table 2.1 provides a comparison between these approaches. The used criteria are: methodology, advantages, disadvantages, as well as suitability of a certain approach for implementing existing or new simulation model.

Table 2.1: Comparison between different computational steering approaches

	Program Annotation	**Redesigning the Simulation Application**	**Steering by Scripting**	**High-level abstractions**
Methodology	inject API calls to the application code	reorganise the application code with computational steering in mind	divide the simulation code into small modules and let the user interacts with them using a scripting language	view the application as a set of computational objects. Application code, which requires steering features, should be inherited from the appropriate classes
Advantage	well suited to transform a non-interactive simulation code into an interactive one	full utilisation of computational steering power	adapts existing tools to be used in computational steering framework	better organisation of application code, well suited for parallel programs
Disadvantage	inefficient in case of separate visualisation, does not provide much control over the application code, requires the user to have simulation code	enforce rewriting the scientific application	it is not intuitive, not highly interactive	require the user to have object-oriented knowledge
Support of existing applications	high	low	medium	medium
Support of new applications	low	high	low	low
Examples	CUMULVS, Magellan	SCIRun	Matlap, R	DISCOVER (DIOS)

2.4 Tasks

Computational steering environments are usually used to perform one or more of the following tasks: model exploration, algorithm experimentation, or performance optimisation. Although the technique is the same in all of these functionalities which involves monitoring, interpretation and parameter change, the extracted data as well as the steering agent might differ. For example, in model exploration usually a human being performs the interpretation and takes actions. However, in performance optimisation an automated algorithm is likely to take the decision based on the current system state.

In the following, we discuss briefly these tasks.

2.4.1 Model Exploration

In model exploration, the primary goal is to explore the simulated model behaviour under the change of various parameter values. Therefore, the focus here is the input/output relationship. This is an early and an actual application of computational steering in the scientific discovery process. Additionally, model exploration helps scientists to get a deeper understanding of the problem under study by permitting them to ask "what-if" questions simultaneously while the simulation is in progress. The interpretation and visualisation processes are usually carried out by a human user. In this context, the steered application is typically simulation code that responds to the user changes. Examples of software which belong to in this category are: CSE, SCIRun, and CUMULVS.

2.4.2 Algorithm Experimentation

Algorithm experimentation allows the underlying code to be modified or refined at run time. It is similar in its function to the model exploration task, however, it is different in its purpose. In algorithm experimentation, running code could change its internal data structure in response to the user changes. Furthermore, the expected output to the user is slightly different from the previous task. Here, the user is primary interested in the internal data structure organisation and other runtime information. VASE is an early example of computational steering environments which can perform such tasks.

2.4.3 Performance Optimisation

Performance optimisation [VS99] is used to change the computational resource allocation that affects simulation performance at runtime such as load balancing [Par99]. It is another task of computational steering. In parallel and distributed computation, there are multiple computational units cooperating to solve the same problem. The equal distribution of the workload between these working processors is a well known problem. Computational steering can obviously contribute in solving such problems

by permitting a separate steering agent to change the workload at runtime. Unlike the previous two tasks, the steering agent is likely to be an automatic algorithm. An example of steering software which fall in this category is Magellan [VS99].

2.5 Software

Different variations of computational steering tools have been developed during the recent two decades with different purposes and goals. The ultimate aims of these environments range from performance modification of running applications to the modification of the underlying computational simulation [Par99]. Previous work has classified these environments based on different criteria. For example, in [VS96], computational steering environments are classified primary based on the steering agent (human or automatic algorithm), while in [MWL99] different criteria are used (e.g., user interface, architecture). In this section, we combine these criteria and provide a more recent survey of these tools. Our taxonomy is based on the following measures:

- model size that could be studied using the tool
- supported steering approach (e.g., program annotation, redesigning scientific application, etc.)
- application area (e.g., computational fluid dynamics, molecular dynamics, etc.)
- steering task (model exploration, performance optimisation, or algorithm experimentation)
- steering agent (human or automated algorithm)
- underlying architecture
- simultaneous steering of different applications or models
- support for parallel applications
- how are the simulation and visualisation coupled together
- collaboration between different users
- ability to integrate existing code into the computational steering framework
- portable implementation (can the tool run on different platforms?)
- distinguishing features of an environment compared with the others
- other functionalities

2.5.1 CUMULVS

CUMULVS, Collaborative User Migration; User Library for Visualisation and Steering, [GKP97] is a steering library which allows the flexible incorporation of interactive visualisation and computational steering into existing parallel programs. It was developed by Oak Ridge National Laboratory. A distinguished feature of CUMULVS over the other developed computational steering environments is the fault tolerance capabilities which is useful when running a distributed application. Fault tolerance is realised by means of check-pointing. The role of these check-pointing functions is not only to recover the application in case of failure, but also to improve the performance of the running code by interactively migrating application tasks across heterogeneous platforms.

The architecture of CUMULVS consists of two parts: one for the application program and the other one for the possibly commercial visualisation and steering front end. It is dedicated to existing applications which can be transformed to a steering one by instrumenting the simulation code with library calls. However; it does not provide dedicated visualisation modules, instead existing visualisation packages could be used, which does not provide much control over the simulation and visualisation process.

CUMULVS supports the steering of parallel programs which are built on the parallel virtual machine framework (PVM) [PVM12], and it supports multiple views of the same running application to assist collaboration between different users. Furthermore, it can run across different platforms. CUMULVS has no specific application, however, it can be used whenever a fault tolerance mechanism is required

Although there are no reported usage in which the steering agent is an automated algorithm, it is viable to use CUMULVS to automatically balance the load of a distributed application running on different machines. Additionally, CUMULVS permits different users to collaboratively steer the same application. A token scheme is used to prevent conflicts between different user, however, it does not support steering of multiple applications.

Since CUMULVS could support distributed and parallel applications, it permits the steering of applications ranging from medium to big size.

2.5.2 CSE

The computational steering environment (CSE) [LMW96, LW97], which was developed at the Center for Mathematics and Computer Science in Amsterdam, is based on a centralised data manager with different number of clients that can connect to and disconnect from it. These clients are called satellites. The main functions of the data manager is to maintain a database of variables and to notify the connected clients of changes in the variable values, while satellite clients could be used to perform calculations and visualisations.

The primary computational steering task of CSE is model exploration and the application source code needs to be annotated in order to support computational steering.

CSE supports multiple applications to be steered simultaneously with different users. Furthermore, these users and application clients could be distributed over different platforms which means that it is a platform independent. However, CSE allows only small and medium size models (ranging from 10 to 1,000 of variables) to be steered and it is not able to steer parallel code.

Since CSE is tailored to model exploration, the steering agent could be only a human being.

In CSE, visualisation and simulation are tightly coupled with each other. CSE is provided with a satellite user interface called PGO editor which carries out all of the visualisation tasks. The PGO editor allows a user to create custom 2D, 3D user interfaces to visualise and manipulate the data on the data manager side.

2.5.3 DISCOVER

DISCOVER (Distributed Interactive Steering and Collaborative Visualisation EnviRonment) [MMMP01], developed at Rutgers University, is a computational platform that brings together key technologies in interactive application frameworks. It is a virtual, interactive, and collaborative problem solving environment that enables remote users to collaboratively monitor and manipulate high performance parallel applications through web-based portals [LJPS05].

DISCOVER supports a three-tier architecture. The user client is at the front. It is accessible through a web browser. The running application is at the back end, while at the middle layer the network of interacting servers.

The DIOS (Distributed interactive object Substrate) library [MP02b] is used to provide the runtime monitoring, interaction and computational steering of parallel and distributed applications as part of DISCOVER.

Multiple users can steer different applications simultaneously. Moreover the clients and servers could be distributed over different machines.

Additionally, DISCOVER provides tight coupling between simulation and visualisation and permits users to adjust the visualisation primitive at run time through a rule based visualisation [LJPS05]. Moreover, it permits the steering and visualisation of high-performance parallel/distributed applications.

DISCOVER is used to include existing computational code into collaborative and interactive environments. It is based on the high-level abstractions approach of computational steering and it supports the model exploration task. DISCOVER is implemented in Java and therefore could run on different platforms.

Moreover, the web-based framework of DISCOVER distinguishes it from other computational steering environments. The steering agent is often a human which interacts with the running computational objects.

2.5.4 POSSE

POSSE [MLP02, Ani02] (Portable Object-oriented Scientific Steering Environment) is a general-purpose, lightweight, portable software system based on a simple client/server model. The server is implemented to run on the simulation code side and controls registered application data. The steering server is created as a separated thread on the running simulation.

POSSE supports the collaboration between different users by permitting different clients to be attached to the running server, however, only one application can be steered by the same group of users. Moreover, parallel simulation is supported by using the message passing interface library.

Using POSSE, a steering client receives runtime information from the application and conveys this information to the user. POSSE supports both types of steering agents, however, no typical case study is published to illustrate the use of algorithms to perform the steering.

Model exploration and performance optimisation tasks can be supported using POSSE. Additionally, it uses different clients to perform the visualisation and steering.

Originally, POSSE was developed to simulate and visualise aerospace models in real-time [MLP02] using a multi-processor MIMD architecture.

2.5.5 RealityGrid

The RealityGrid computational steering library [PHPP05] has been developed as part of the RealityGrid project. Its ultimate goal is to allow computational scientists to have their own code interactively run over the grid. Therefore, a distinguished feature of this library over the other mentioned tools is the use of computational steering over the grid. The computing architecture of the RealityGrid is spilt into: computation, visualisation, and the steering client. Additionally, it supports the steering of the sequential and parallel applications, however, no explicit support is given for the latter case (i.e., it is independent from any parallelisation library, e.g., MPI or PVM). RealityGrid also supports http file transfer to permit the communication between different steering components. The steering API is platform independent, which makes it available to be used under different software and hardware platforms. The RealityGrid API supports applications written by C, C++, or Fortran.

Likewise, RealityGrid requires the modification of the source code in order to permit steering and visualisation. It has been used in many different areas of scientific simulation. The steering task of the RealityGrid computational steering library is mainly model exploration.

2.5.6 SCIRun

SCIRun [PJ95, PJB97] is an integrated shared memory based scientific programming environment that permits the interactive construction, debugging, and steering of large scale scientific computations. It is targeted towards medium to large size problems and it is based on a data flow programming model to allow scientists designing and interactively modifying the simulation. SCIRun is developed at the Scientific Computing and Imaging Institute, Utah university. The SCIRun data flow paradigm is common to many scientific visualisation packages which makes it easy to be used by users which are familiar with those scientific programs (e.g., AVS). Moreover, it can be used both on single workstation and Symmetric Multi-Processing (SMP) environments [Mar96].

Although SCIRun provides the capability of constructing a new interactive application, it does not allow the steering of multiple applications at the same time, nor the collaborative interaction between multiple users.

SCIRun is usually referred to in the computational steering literature as an example of a computational steering software which involves designing the computational steering application from scratch. Users can use the provided tool to graphically design a computational module. Afterwards, they are able to interact with the previously created application.

Steering of programs developed by SCIRun takes place using a human and no support for an automatic algorithm is provided to perform the monitoring and steering of the running program.

2.5.7 Magellan

Magellan [VS99] is a prototype computational steering system that uses interpreter language to control multi-threaded asynchronous steering servers which cooperatively steer an application. Magellan's basic structure consists of a group of steering servers that run in the same application address space. To support steering for an application, at least one embedded steering server should be running. Only one server is needed to support steering in shared-memory, thread-based applications. However, distributed memory applications must have one server for every message passing interface process. In case of multiple servers, one additional server will be needed to coordinate the work of other servers. Such master servers usually run at process ID =0 and coordinate the operation of the other servers where using MPI. The steered application must be annotated by a set of API calls to allow the server to communicate with it. The steering agent is interacting with the annotated application through a language called ACSL. The steering agent could be a human that issues commands at a command line terminal or another application running in the same or another machine to control the annotated applications.

An example of the command primitives which are supported by ACSL are: probe

– retrieves or overwrites an application-specific object value regardless of the application's current state, sensor – provides consistent monitor information, and actuator - alters an application object value. The steering agent does not communicate directly with the steered application, instead it sends requests to the servers which process it and reply to the steering client with adequate action. However, to render the command primitive on the agent side, a corresponding annotation must be added to the original application code.

Magellan supports the computational steering tasks model exploration and performance optimisation. However, only one application can be steered at a given time.

2.5.8 STEEL

As a different application scenario of computational steering, we have developed, in previous work, STEEL (STeering Environment for E-Learning) [HEA07] to aid students in understanding abstract scientific concepts. The STEEL framework is based on the client/server/client architecture, with an intermediate server that stands between different clients. The client could be either the simulation code or the steering client. The steering client performs visualisation and parameter changes. STEEL supports also the collaboration between different users as a means to implement collaborative learning, however it does not support running more than one model simultaneously. In case of multiple users, the steering server will act as a coordinator between them. STEEL could also integrate parallel applications, but without any assumption of the parallelisation library. Additionally, the different STEEL components could be distributed over network connected-computers.

Influenced by CUMULVS, STEEL supports check-pointing to provide the opportunity to restart the running simulation in the case of sudden failures.

2.5.9 EPSN

EPSN [ERC06] is a computational steering environment which permits the interconnection between parallel simulations with parallel visualisation systems. It has been influenced by other computational steering environments that support the steering of parallel simulation (e.g., CUMULVS). Therefore, this environment is based on a framework coupling parallel simulation and parallel visualisation components. Furthermore, EPSN is based on a client/server architecture whereby the simulation code and the visualisation modules could be considered as servers that fulfil client requests.

The integration of an existing legacy simulation code into EPSN is done by annotating the original simulation code with API calls that perform the decomposition and collection of data from different processors. Later on, the steering process is supported by a request command. Three types of request commands are provided: control (play, step, and stop), data access (allows parameter read/write), and action (allows to invoke

user defined routines).

As already mentioned above, EPSN components can be distributed over different computers and both the simulation and visualisation are possible to be performed using parallel processing, however it does not support the collaboration between different users.

EPSN is used to perform model exploration and the users of EPSN are mainly human beings, since no facility is given to allow automated algorithms to steering the parallel simulation. EPSN provides its own parallel viewer to visualise the results of parallel simulation and its steering client which gives some sort of control over the simulation.

2.5.10 Others

There are many other computational steering environments which are not under development any more. For example, VASE (the Visualisation and Application Steering Environment) [BJBH93] allows the steering of existing programs in a way which is similar to programming language debuggers. On the other hand, computational environments which were not originally developed with computational steering as in mind (e.g., Matlab, R, Mathematica, etc.) can also be used to implement computational steering through scripting.

2.6 Challenges

There are some challenges which need to be taken into considerations, to efficiently implement computational steering technique in problem solving environments. In this section we discuss three of them, namely: the performance of computational steering, steering parallel and distributed applications, and the consistency of the computational application.

2.6.1 Performance of Computational Steering System

The performance of a computational steering system is crucial for its success to attract users. Furthermore, the overall goal of using computational steering is to accelerate the scientific discovery process or to enhance the performance of a certain application by redistributing the computational load over different processors. The performance of a computational steering system can be improved by considering these three factor: latency of the monitoring process, latency of the steering agent, and the cost of enacting steering decisions [VS96]. In Chapter 5, we discuss some strategies to increase the performance of the computational steering environment that is developed during the course of this thesis.

The Latency of Monitoring Process

One of the main functions of computational steering is to analyse, extract and present the simulation data from the target simulation to the steering agent in a timely fashion to allow for up-to-date visualisation and interpretation of the simulation results [VS99]. Such a process is usually referred to as monitoring. On the other side, some computational steering environments tightly integrate the collection of results with the simulation process itself by running a steering server in the same memory space as the running simulation. While this technique will save some communication overhead, it consumes additional memory due to the internal data structures. Some computational steering environments overcome the problem of memory overhead by running the data analysis stage in a remote computer by using the client/server model. Nonetheless, simulation engines produce large amount of data. Transmitting such huge amounts of data in the scale of petascale or exascale is still a challenge for the implementation of computational steering environments.

The Latency of Steering Agent

Another factor that influences the performance of a computational steering is the response time of the steering agent. In the case of a human agent, we cannot expect a big improvement of the system performance [VS96]. However, the system should give the user the ability to run the simulation step by step or in a complete batch way. In the latter case, the performance will be determined by the algorithm reaction time.

The Cost of Enacting Steering Decision

The steering layer in Figure 2.3 is responsible for communicating the necessary information from/to the user. The performance of this layer is also crucial for the overall system performance, particularly, when the application and the user layers are distributed across different machines.

2.6.2 Steering of Parallel and Distributed Applications

With the advances of scientific computation, the application code itself could be run on different computers or parallelised using multi-core machines. This issue bring more challenges to include such applications into a computational steering framework. Furthermore, the processes of collecting data and sending the user interactions become more complicated. Additionally, care should be paid to the performance and the consistency of the original application when incorporating it into the computational steering framework. For example, CUMULVS provides a fault tolerance mechanism to recover the application in the case of failure instead of starting it again.

2.6.3 Application Consistency

Application consistency (i.e., ensuring that the application's internal data is coherent) is another major issue when considering computational steering, particularly in the case of permitting multiple users to steer the same model simultaneously. In such a situation, a synchronisation mechanism is required to prevent different users editing the same data structures at the same time.

2.7 Closing Remarks

In this chapter, we have presented an overview of computational steering techniques, including various approaches, tasks, and related work. Computational steering is particularly useful to be used whenever considering solving large-scale problems that require much time to be simulated. Although many computational steering environments have been developed in the last years, none of them is dedicated to the task of interactive modelling and analysing of biochemical networks. In Chapter 5, we present a more specific and novel application of computational steering, the kinetic simulation of biochemical networks by the use of Petri nets.

3 Simulation Approaches of Biochemical Networks

3.1 Introduction

Computer simulation is an essential tool for studying biochemical systems. The deterministic approach is the traditional way of simulating biochemical pathways [WUKC04, Gil07, Pah09]. In this approach, reactions and their influence on the concentrations of the involved species are represented by a set of ordinary differential equations (ODEs). While this approach has the advantage of having a well-established mathematical basis and documentation, it lacks the ability to capture the phenomena which may occur due to the underlying discreteness and random fluctuation in molecular numbers [MA99, Pah09], especially in situations where the number of molecules is small.

The stochastic approach [Gil76, Gil77] overcomes the drawbacks of deterministic simulation and provides a natural way of simulating biochemical pathways, since it can successfully capture fluctuations in the underlying model. Furthermore, it deals correctly with the problem of extremely low numbers of molecules [MA99, ACT$^+$05]. Nevertheless, a major drawback of stochastic simulation is that it is computationally expensive when it comes to simulating larger biological models [ACT$^+$05, LCPG08, Pah09], particularly when there are large numbers of molecules of some chemical species.

The situation becomes even more complicated if a model combines different reaction scales, i.e., slow and fast reactions and/or species with small and large numbers of molecules. In this case, neither stochastic nor deterministic simulation is appropriate to efficiently analyse it, because stochastic simulation will be very slow and the continuous one will fail to capture the fluctuation caused by species with few copies of molecules.

Hybrid simulation of biochemical networks has been previously studied in, for example, [HR02, KMS04, SK05, ACT$^+$05, GCPS06]. To overcome the problem of stiffness (see Section 3.4.5), reactions are divided into two subsets: slow and fast. The slow set is simulated stochastically, while the fast one is simulated continuously using an ODE system.

In this chapter, we discuss frequently used approaches to simulate biochemical networks, including stochastic, deterministic and hybrid. Moreover, we pinpoint the Petri net counterparts of these approaches: stochastic, continuous, and hybrid Petri nets. The relationships between the simulation algorithms and Petri net notations are also given. Based on the algorithms and Petri net classes discussed here, we will introduce in

the next chapter a new net class that simulates biochemical networks using the hybrid approach.

3.2 Preliminaries

More formally in this chapter and the next one, we are interested in the following specific problem: consider a well mixed system of N chemical species $S_1, \ldots, S_N$, which interact using M chemical reactions $R_1, \ldots, R_M$. Each reaction has a rate which determines how often a reaction occurs. The reaction rates are calculated in terms of the reactions' substrates and the rate constants $k_1, \ldots, k_M$ (in the stochastic approach, k_j is usually denoted by c_j to emphasise that they are not the same, however, they can be calculated in terms of each other). Besides, the reactions $S \rightarrow \phi$ and $\phi \rightarrow S$ are used to denote the degradation and synthesis of species S, respectively.

Additionally, each reaction R_j is associated with a state change vector, $\mathbf{v}_j$, which determines the system changes when the reaction R_j takes place. The state of the system at any time τ, can be represented by an N-vector $\mathbf{X}(\tau) = x_1(\tau), \ldots, x_N(\tau)$, where $x_i(\tau)$ gives the number of molecules of species S_i at time τ, i.e., $x_i(\tau) = S_i(\tau)$. The goal is to find an estimated evolution of the vector $\mathbf{X}$ over the time τ, starting from an initial state $\mathbf{X}(\tau_0)$ [Gil07].

In the following we often refer to a reaction R_j by just giving its index j. Moreover, to simplify the mathematical notations, the vector $\mathbf{X}(\tau)$ is sometimes denoted by just $\mathbf{x}$ when it is not important to emphasise the notation of time. Later in Section 3.6, we derive a correspondence between the notations introduced here and the Petri net ones. Indeed, biochemical reaction networks can be intuitively modelled using Petri nets.

The system states are evolving with time. At each time step a reaction R_j occurs, we get a new state of the system. For example, suppose the system is at time τ in a state $\mathbf{X}(\tau)$, if a reaction R_j occurs at time $\tau + d\tau$; the system will reach a new state, $\mathbf{X}(\tau + d\tau) = \mathbf{X}(\tau) + \mathbf{v}_j$. The problem of how to calculate the system state accurately and efficiently over the time is dissected in many research studies.

In the discrete approach (e.g., stochastic simulation algorithm (SSA) [Gil76]), the states of the system are represented as vectors of non-negative integer numbers which change discretely by occurring reactions, while in the continuous approach (e.g., using ordinary differential equations to simulate the biochemical reaction network) system states are represented by vectors of non-negative real numbers which change continuously with respect to time. The hybrid approach combines discrete and continuous states of the system. Continuous states can be viewed as average concentrations when considering the hybrid setting.

In the sequel, we discuss some of the widely used methods to simulate biochemical reaction networks.

3.3 Deterministic Approach

If the thermodynamic limit condition holds (i.e., the number of molecules and the volume of the system approach infinity), then the evolution of the above system can be represented as a set of ODEs [HR02, WUKC04, Gil07] in the form of (3.1), where the concentration of species S_i is denoted by $[S_i]$.

$$\frac{d[S_i]}{d\tau} = f_i([S_1], \ldots, [S_N]), \tag{3.1}$$

where $f_i([S_1], \ldots, [S_N])$ is a function of the species concentration.

The deterministic simulation approach is based on the assumption that the system under study has sufficiently many molecules that the number of molecules can be approximated by a continuous variable which can be solved using ordinary differential equations [GB00]. Under this assumption, species' concentrations are varying deterministically with time (i.e., if we repeat the simulation multiple times from a certain initial state, we always reach the same state in a future time point). Indeed, deterministic simulation describes $\mathbf{X}(\tau)$ as a continuous deterministic process [Gil01].

The system of ODEs can be derived from the biochemical reactions using either elementary kinetic rate laws (e.g., mass-action) or non-elementary phenomenological rate laws (e.g., Michaelis-Menten kinetics). The difference between the two laws is the required level of detail. For instance, a simple enzymatic reaction can be represented as in (3.2) using Michaelis-Menten kinetics.

$$S \xrightarrow{E} P \tag{3.2}$$

By applying the law of Michaelis-Menten in (3.3) [BGHO08],

$$v = V_{max} \frac{[S]}{K_m + [S]} \tag{3.3}$$

The corresponding ODEs can be obtained in (3.4)

$$\frac{d[P]}{d\tau} = -\frac{d[S]}{d\tau} = K_{cat}[E_t] \frac{[S]}{K_m + [S]} \tag{3.4}$$

where $V_{max} = K_{cat} \cdot [E_t]$, $[E_t]$ is the total concentration of enzyme E, and K_{cat}, the turnover number, is the maximum number of substrate molecules converted to product per enzyme molecule per second.

Using mass-action kinetic law, the reaction in (3.2) can be further detailed into three elementary reactions.

$$\begin{aligned} S + E &\xrightarrow{k_1} S|E \\ S|E &\xrightarrow{k_2} S + E \\ S|E &\xrightarrow{k_3} P + E \end{aligned} \tag{3.5}$$

where $S|E$ is a complex formed by the substrate S and the enzyme E. The ODEs can be derived using (3.6) as shown in (3.7).

$$\frac{d[S_i]}{d\tau} = \sum_{j=1}^{M} v_{ji} k_j \prod_{l=1}^{N_j} [S_l]^{\alpha_{jl}} \tag{3.6}$$

where v_{ji} is the change in the number of molecules of S_i due to the occurrence of the reaction R_j, N_j is the number of reactant species in reaction j, and α_{jl} is the stoichiometric coefficient of reactant species S_l when participating in reaction R_j. For the models presented in this thesis we often refer to the term $k_j \prod_{l=1}^{N_j} [S_l]^{\alpha_{jl}}$ by just the pattern $MassAction(k)$.

$$\begin{aligned} \frac{d[S]}{d\tau} &= -k_1 \cdot [S] \cdot [E] + k_2 \cdot [S|E] \\ \frac{d[P]}{d\tau} &= k_3 \cdot [S|E] \\ \frac{d[E]}{d\tau} &= -k_1 \cdot [S] \cdot [E] + (k_2 + k_3) \cdot [S|E] \\ \frac{d[S|E]}{d\tau} &= k_1 \cdot [S] \cdot [E] - (k_2 + k_3) \cdot [S|E] \end{aligned} \tag{3.7}$$

There are a multitude of ODE solvers to deal with (3.7) and produce a numerical solution of it. In the following we briefly discuss some of the available options.

3.3.1 Types of ODE Solvers

Traditionally, ODE solvers can be classified based on different criteria, e.g., fixed-step vs variable-step size, explicit vs implicit, fixed-order vs variable-order, single-step vs multi-step, etc. For more details about this classification see [HNW93, HW96]. Usually, ODE libraries provide an implementation of multiple different solvers that cover more than one of the aforementioned criteria.

Fixed-step solvers advance the solution from one point to another by a fixed-step, h.

A good approximation of the underlying ODEs is achieved if the step-size is kept very small [PTVF02]. Examples of the ODE algorithms that fit in this category are Euler and the classical Runge-Kutta method. However, using very small steps to obtain a good approximation will result in a slow solver. Nevertheless, the very small step size might not be required during the whole solution. Therefore, a good ODE integrator should vary the step size throughout the solution process. Small steps can be used in non-smooth regions, while relatively big step sizes are used in smooth solutions. Such ODE solvers are called adaptive or variable-step size solvers. Usually all modern ODE integrator are adaptive. The step size is usually determined through the satisfaction of certain criteria, which are called accuracy in this context. The accuracy is a problem-dependant. The main gains of using those solver type is the speed-up of the integration time. An example of the algorithms in this category is the Cash-Karp Runge-Kutta method.

Explicit solvers find the solution of the set of ODEs using the explicit formula:

$$y_{n+1} = y_n + hf(x_n, y_n). \tag{3.8}$$

That is they advance the solution using a previous solution point(s). A major problem with this approach is that they cannot deal with stiff ODEs (see Section 3.4.5 for an example). Contrary, implicit ODE solvers can be used to deal with such problem. Implicit ODE solvers approximate the solution at point y_{n+1} in terms of the solution at y_{n+1}. That is, they find a solution by solving an equation involving both the current state of the system and the later one. For each known explicit algorithm, an implicit formula can be constructed. For instance, we have the explicit as well as the implicit Euler method. Some solvers implement the two formulae by switching between the explicit and implicit method when solving the same problems.

The order of the ODE solver is determined by the error term $O(h)$. An ODE algorithm is called of order n if it produces an error of $O(h^{n+1})$. For example, the explicit Euler method is of the first order, since it produces an error of $O(h^2)$ while the mid-point method (or second-order Runge-Kutta) is of the second order, since it produces an error $O(h^3)$ [PTVF02]. Reducing the error term corresponds to improving the accuracy of the solution (i.e., solvers of higher orders are of higher accuracy). Similar to the step size, some solvers use a fixed order throughout the solution, e.g., Euler, while others use adaptive order, e.g., BDF (Backward Differentiation Formula).

Single-step algorithms (e.g., Runge-Kutta) produce the solution using only one history point. By other words, single-step ODE solvers approximate the behaviour of the model at time $\tau + dt$ by taking into account only the behaviour of the model between times τ and $\tau + d\tau$. Contrary, multi-step methods (e.g., BDF) use multiple history points to produce the solution in a more efficient way. Therefore, multi-step algorithms use the concept of memory to gain more performance index. For example, CVODE find a solution at a certain point of time using a high order polynomial to multiple

predicted points. Thus, they outperform (with many stages) single-step algorithms. We return back to discuss this point in Section 4.4.2 after introducing the simulation of $\mathcal{GHPN}_{bio}$.

3.3.2 Problem of the Deterministic Approach

Nevertheless, if the system contains some species with low numbers of molecules, then the thermodynamic limit condition will be violated and the deterministic approach will not reflect the actual model behaviour [Gil76]. In this case, stochastic simulation can be used to simulate the model at the molecular level which takes into account the inherently discrete and stochastic nature of chemical reactions [MA99].

3.4 Stochastic Approach

Unlike the deterministic approach, stochastic methods [McQ67, Gil76, Gil77, MA99, Gil07] simulate the system evolution with respect to time in a way that takes into account the discrete and stochastic nature of reacting chemical species.

In the stochastic settings, each reaction R_j is characterised by a propensity function denoted by a_j which is defined by 3.9 [Gil76].

$$a_j(\mathbf{x})d\tau \triangleq \text{the probability, given } \mathbf{X}(\tau) = \mathbf{x} \text{ that one reaction } R_j \text{ will occur in the next infinitesimal interval } [\tau, \tau + \delta\tau) \tag{3.9}$$

The value of $a_j(\mathbf{x})$ in 3.9 is calculated based on the reaction type: unimolecular or bimolecular reaction.

$$a_j(\mathbf{x}) = \begin{cases} c_j \cdot x, & R_j \text{ of the form } S \rightarrow P_1 + \cdots + P_n \\ c_j \cdot x_1 \cdot x_2, & R_j \text{ of the form } S_1 + S_2 \rightarrow P_1 + \cdots + P_n \\ \frac{1}{2} c_j \cdot x_1 \cdot (x_1 - 1), & R_j \text{ of the form } S + S \rightarrow P_1 + \cdots + P_n \end{cases} \tag{3.10}$$

where c_j is the stochastic reaction constant. It gives the probability that a particular molecule or a randomly chosen pair of molecules will react in the next infinitesimal time $d\tau$, and $P_1, \ldots, P_n$ are the reaction products.

In a similar mathematical form to (3.6) and (3.10) can be written as in (3.11).

$$a_j(\mathbf{x}) = c_j \cdot \prod_{i=1}^{N_j} \begin{pmatrix} x_i \\ \alpha_{ji} \end{pmatrix}. \tag{3.11}$$

In (3.10), the first reaction type is called unimolecular reaction, while the two other reactions are called bimolecular reactions. The constant c_j can be calculated in terms

of the deterministic reaction rate constant k_j. More details about how to calculate c_j given k_j can be found in [WUKC04, Gil07].

For example, in a reaction of the form $S_1 + 3S_2 + 2S_3 \xrightarrow{c_1} P_1 + P_2$, $a(\mathbf{x})$ is calculated as follows

$$a(\mathbf{x}) = c_1 \cdot \begin{pmatrix} x_1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} x_2 \\ 3 \end{pmatrix} \cdot \begin{pmatrix} x_3 \\ 2 \end{pmatrix}$$

$$= (c_1 \cdot x_1) \cdot (\frac{1}{3} \cdot x_2 \cdot (x_2 - 1) \cdot (x_2 - 2)) \cdot (\frac{1}{2} \cdot x_3 \cdot (x_3 - 1))$$

In the following we briefly summarise some of the widely used stochastic methods to simulate biochemical reaction networks.

3.4.1 Chemical Master Equation

The ultimate goal of the stochastic approach is to find the molecule numbers of each molecular species S_i at a certain time τ. This can be expressed in terms of probability rules by

$$P_{\mathbf{x}}(\tau) = P\{\mathbf{x}, \tau | \mathbf{X}(\tau_0), \tau_0\}, \tag{3.12}$$

meaning the probability at time τ that the system will be in state $\mathbf{X}(\tau) = \mathbf{x}$ given the initial state $\mathbf{X}(\tau_0)$ [Gil76, WUKC04].

Next, consider the following two system state changes:

$$\mathbf{x} - \mathbf{v}_j \xrightarrow{a_j(\mathbf{x}-\mathbf{v}_j)} \mathbf{x} \tag{3.13}$$

and

$$\mathbf{x} \xrightarrow{a_j(\mathbf{x})} \mathbf{x} + \mathbf{v}_j \tag{3.14}$$

which denote the change to state $\mathbf{x}$, and the change away from state $\mathbf{x}$, respectively by the occurrence of the reaction R_j.

From (3.13) and (3.14) we get the chemical master equation in (3.15) [McQ67].

$$\frac{\partial P_{\mathbf{x}}(\tau)}{\partial \tau} = \sum_{j=1}^{M} [a_j(\mathbf{x} - \mathbf{v}_j) P_{(\mathbf{x}-\mathbf{v}_j)}(\tau) - a_j(\mathbf{x}) P_{\mathbf{x}}(\tau)] \tag{3.15}$$

While (3.15) can determine (3.12) and in turn could calculate the molecular numbers at time τ, it is extremely difficult to solve it either analytically or numerically, except for very simple models. See [WGMH10, HMMW10] for some examples of how to solve (3.15) numerically.

The reason for this difficulty is that (3.15) requires one ODE for each possible combination of reactant molecules. In other words, the number of ODEs does not depend

on the number of reactions or the number of species only, instead it depends on any possible combination of molecules of any species [Gil76, WUKC04, WGMH10].

Gillespie [Gil76, Gil77] proposed a method to construct numerical realisations of the number of molecules in each species. The resulting trajectory is equivalent to the one obtained by applying (3.15) but requires less computations. In the following subsections we briefly discuss the general idea of generating simulated trajectories of the system state as well as other various of these methods.

3.4.2 Direct Method

Instead of using (3.12), Gillespie [Gil76] defined a new probability function $p(\delta\tau, j|\mathbf{x}, \tau)$ to generate simulated trajectories of $\mathbf{X}(\tau)$:

$$p(\delta\tau, j|\mathbf{x}, \tau) = a_j(\mathbf{x})\exp(-a_0(\mathbf{x}), \delta\tau), \tag{3.16}$$

where $p(\delta\tau, j|\mathbf{x}, \tau)$ is the probability that the next reaction in the system will occur in the infinitesimal time interval $[\tau+\delta\tau, \tau+\delta\tau+\epsilon)$ given that the system is in a certain state $\mathbf{X}(\tau) = \mathbf{x}$, $\delta\tau$ the next time for a reaction to occur, ϵ is a small error, j the type of this reaction and

$$a_0(\mathbf{x}) = \sum_{j=1}^{M} a_j(\mathbf{x}) \tag{3.17}$$

is the total (cumulative) propensity.

The function $p(\delta\tau, j|\mathbf{x}, \tau)$ is the joint probability function of two random variables: $\delta\tau$ and j. Using Monte Carlo simulation, we can generate samples for both $\delta\tau$ and j [Gil76].

According to the direct method [Gil76, Gil77], the next time $\delta\tau$ at which a reaction will occur is specified by

$$\delta\tau = -\frac{1}{a_0(\mathbf{x})} \ln r_1 \,, \tag{3.18}$$

and the reaction R_μ to occur is determined by

$$\sum_{j=1}^{\mu-1} a_j(\mathbf{x}) < r_2 a_0(\mathbf{x}) \leq \sum_{j=1}^{\mu} a_j(\mathbf{x}) \,, \tag{3.19}$$

where r_1 and r_2 are two random numbers which are generated from a uniform distribution $(0, 1)$ [Pah09].

The steps involved in simulating a set of reactions using the direct method are summarised in Algorithm 3.1.

Algorithm 3.1: Direct method

1: $\tau_{end} \leftarrow$ simulation end time
2: Initialise the simulation using the initial state $\mathbf{x} = \mathbf{x_0}$, $\tau = \tau_0$;
3: **while** $\tau \leq \tau_{end}$ **do**
4: For each reaction R_j, calculate $a_j(\mathbf{x})$;
5: Calculate a_0 using (3.17);
6: Generate two random numbers r_1, r_2 from the uniform distribution (0,1);
7: Calculate $\delta\tau$, μ using (3.18) and (3.19) respectively;
8: Fire the reaction R_μ;
9: Update the system state and current time using $\mathbf{X}(\tau + \delta\tau) = \mathbf{X}(\tau) + \mathbf{v}_\mu$ and $\tau = \tau + \delta\tau$, respectively;
10: **end while**

3.4.3 First Reaction Method

An alternative to the direct method is the first reaction method [Gil76]. The main difference between the two methods is how to generate $\delta\tau$ and j. In the first reaction method, M random numbers, $r_1, \ldots r_M$, are drawn from the uniform distribution (0,1).

Then, the putative firing times of all of the reactions, $\delta\tau_{j'}$, are calculated according to (3.20).

$$\delta\tau_{j'} = -\frac{1}{a_{j'}(\mathbf{x})} \ln r_{j'} \tag{3.20}$$

After that, the next time a reaction to occur is $\delta\tau$ = the smallest of the $\delta\tau_{j'}$ and the next reaction type, j, to fire is the one with minimum $\delta\tau_{j'}$.

Algorithm 3.2 summarises the steps needed to generate simulated trajectories using the first reaction method.

The first reaction method is empirically slower than the direct method for a system which contains a substantially large number of reactions [Gil07]. However, the first reaction method is useful in certain applications where some extensions are added to it. For instance the next reaction method, which will be presented in the next subsection, is an extension of the first reaction method that outperforms the efficiency of the direct method. Moreover, it can be parallelised much easier than the direct method.

3.4.4 Next Reaction Method

The direct and first reaction methods are very slow when they simulate models with many species and many reactions. Therefore, there are a number of extensions to enhance the computational efficiency of these basic algorithms. In [GB00], the first reaction method is extended by two ways to speed up the basic stochastic simulation

Algorithm 3.2: First reaction method

1: $\tau_{end} \leftarrow$ simulation end time
2: Initialise the simulation using the initial state $\mathbf{x} = \mathbf{x_0}$, $\tau = \tau_0$;
3: **while** $\tau \leq \tau_{end}$ **do**
4: For each reaction R_j, calculate $a_j(\mathbf{x})$;
5: Draw M random numbers from the uniform distribution (0,1);
6: Calculate the putative firing time, $\delta\tau_{j'}$, for each reaction using (3.20);
7: Calculate $\delta\tau$=Min$\{\delta\tau_{j'}\}$ and j=the reaction index of the minimum $\delta\tau$;
8: Fire the reaction R_j;
9: Update the system state and current time using $\mathbf{X}(\tau + \delta\tau) = \mathbf{X}(\tau) + \mathbf{v}_j$ and $\tau = \tau + \delta\tau$, respectively;
10: **end while**

algorithm (SSA): the introduction of dependency graphs and priority queues.

One reason behind the low performance of the SSA algorithms is the necessity to keep the reaction propensities up-to-date after the occurrence of a certain reaction. One naive solution is to recompute propensities of all the reactions when one of them took place. Following this idea, $O(M)$ basic operations will be needed each time a reaction occurs. Indeed, it is a source of low performance. Gibson and Bruck [GB00] propose an idea to minimise the required time for this step by introducing the dependency graph. The idea behind the dependency graph data structure is to record for each reaction R_j the other dependent reactions which need their propensities to be updated after the firing of R_j.

While the dependency graph outperforms the performance of the two other basic exact methods, it needs more efforts to be implemented. Furthermore, it requires extra space to store the additionally introduced data structure.

The other extension is the introduction of an indexed priority queue to decrease the number of generated random numbers. Indeed, the next reaction method uses only one random number per reaction firing. In this data structure, the next reaction to fire is always put in the root of a binary tree. Initially, M random numbers are needed to initialise the priority queue. Afterwards, when a reaction occurs, its putative time is replaced by a new value and its position is updated so that the basic premise of the data structure is maintained (i.e., the reaction with minimum firing time is kept in the root).

It has been repeatedly asserted that the next reaction method could increase the simulation performance, while preserving the exactness of the SSA. Nevertheless, it requires substantial implementation efforts. The detailed algorithm can be found in [GB00, Gib00].

3.4.5 Tau-leaping Method

Although the two stochastic simulation algorithms (direct and first reaction methods) and their many variations are exact and accurate, they are very slow to solve many practical problems [Gil07, Pah09]. The reason for such low performance is that SSA insists to simulate each reaction individually.

The tau-leaping method [Gil01, GP03, RPCG03, CGP05, CGP06] has been proposed to overcome such limitations by sacrificing the exactness of the SSA and gaining some speed-up by firing multiple reactions together in one step. The idea behind such procedure is to advance the simulation time by a certain step size, $\delta\tau$, such that during the time period $[\tau, \tau + \delta\tau]$, no propensity function is changed by a significant amount. The constraint of keeping the change in the reaction propensity very small is called the leaping condition [Gil01]. Calculating an appropriate value of $\delta\tau$ is the central and most difficult question in using the tau-leaping method.

The tau-leaping method provides one further approximation step to simulate biochemical reaction networks. Indeed, it can provide a smooth transition from the exact SSA to the chemical Langevin equation and finally to the deterministic simulation [Gil01].

The tau-leaping method defines for each reaction R_j a constant n_j such such that:

$$\begin{aligned} &n_j(\delta\tau; \mathbf{x}, \tau) \triangleq \text{the number of times a reaction} \\ &R_j \text{ will fire in the time interval } [\tau, \tau + \delta\tau], \text{given } \mathbf{X}(\tau) = \mathbf{x}. \end{aligned} \tag{3.21}$$

Under the assumption of the leaping condition, $n_j(\delta\tau; \mathbf{x}, \tau)$ will be the Poisson random variable defined by (3.22) [Gil01, GP03].

$$n_j(\delta\tau; \mathbf{x}, \tau) = P(a_j(\mathbf{x}), \delta\tau) \tag{3.22}$$

Let $\mathbf{d} = \sum_{j=1}^{M} n_j \mathbf{v}_j$, then the tau-leaping condition can be rewritten mathematically as in (3.23).

$$\text{select } \delta\tau : \forall R_j, |a_j(\mathbf{x} + \mathbf{d}) - a_j(\mathbf{x})| \text{ is effectively small.} \tag{3.23}$$

Explicit Tau-leaping

The first realisation of the tau-leaping procedure is the explicit tau-leaping. It is called explicit because it uses an explicit updating formula (notice the correspondence in terminology between ODEs and tau-leaping methods).

The explicit tau-leaping method produces an approximate trajectory of M reaction channels by firstly selecting a value for the step size $\delta\tau$ that satisfies 3.23. Then it

generates M statistically independent random numbers from the Poisson distribution $n_1, \ldots, n_m$. Finally, the system state is updated using (3.24).

$$\mathbf{X}(\tau + \delta\tau) = \mathbf{X}(\tau) + \sum_{j=1}^{M} n_j \mathbf{v}_j \tag{3.24}$$

Although, it seems very trivial to perform the simulation using this simple procedure, the selection of $\delta\tau$ is tricky [Gil01]. In addition to satisfying the leaping condition, the stochastic step-size needs to take into consideration the following two problems:

- $\delta\tau$ needs to be large enough such that we get noticeable simulation speed-up,
- the selected $\delta\tau$ should not result in negative values in the species number of molecules.

These issues are discussed in more details in [GP03, GP03, CGP05, CGP06]. Selecting a very small value for $\delta\tau$ such that $\delta\tau = \frac{1}{a_0(\mathbf{x})}$, will result in trajectories which are equivalent to the SSA ones. In this case only one reaction will fire at each time step, and it will fire only once [Gil01]. Indeed, $\delta\tau$ plays a crucial role in controlling the speed and accuracy of the tau leaping method.

Stiffness in Biochemical Reactions

Stiffness in systems of ODEs occurs in problems where the independent variables have more than one time-scales [PTVF02]. Using explicit ODE methods (e.g., explicit Euler or explicit Rung-Kutta), the ODE integrator takes steps that are excessively small. Therefore, explicit methods either take very long time to solve the problem or fail completely to produce a solution. The latter case occurs when the ODE solvers require a step size which is below a certain accuracy threshold.

Similarly, in the discrete approach (e.g., stochastic simulation), stiffness can occur due to the existence of reactions with more than one time scales [PWC11]. For an example see the reaction set in (3.25), assuming mass-action kinetics with $c_1 = c_2 = 10^5$ and $c_3 = 0.0005$ and the initial state $x(0) = (10000, 10000, 100)$, reaction R_3 is much slower than R_1 and R_2.

$$\begin{aligned} R_1 &: S_1 \xrightarrow{c_1} S_2 \\ R_2 &: S_2 \xrightarrow{c_2} S_1 \\ R_3 &: S_1 + S_2 \xrightarrow{c_3} S_3 \end{aligned} \tag{3.25}$$

Thus neither the SSA nor the explicit tau-leaping algorithm can efficiently solve the problem. Moreover, the advantage of leaping multiple reactions together will be lost.

One option to deal with the stiffness problem in the discrete case is to emulate the ODE methods and use an implicit updating formula (discussed in next subsection). While another one is to seek the hybrid approach (discussed in Chapter 4).

Implicit Tau-leaping

If the underlying reaction system is stiff, the explicit τ-leaping method will be of low performance. Inspired by the idea of dealing with stiffness in ODEs, an implicit tau leaping method was proposed in [RPCG03]. The proposal is to replace the explicit updating formula in 3.24 with a new implicit formula given by (3.26).

$$\mathbf{X}(\tau + \delta\tau) = \mathbf{X}(\tau) + \sum_{j=1}^{M} [P_j(a_j(\mathbf{x})\delta\tau) - a_j(\mathbf{x})\delta\tau + a_j(\mathbf{X}(\tau + \delta\tau))\delta\tau]\mathbf{v}_j \tag{3.26}$$

In fact the major difficulty of developing an implicit method for the tau leaping simulation is that the tau leaping is based on the Markov process theory which is memoryless when considering updating the system states [Gil07]. For this reason a partial implicitisation is used in (3.26).

The trapezoidal tau-leaping method [CP05] and the adaptive explicit-implicit tau-leaping procedure [CGP07] are two other extension of the updating formula in (3.26).

The detailed steps are beyond the scope of this thesis. Nevertheless, these methods are not easy to be implemented contrary to the simple SSA procedures. Therefore, they require the existence of a library of solvers to be used by systems biologists. Stocksim [LCPG08] is a library that offers an implementation of various algorithms.

3.4.6 Others

Many other extensions of the simple SSA procedure have been reported in the literature. On the one hand, the Optimised Direct Method [CLP04] reorders the reactions such that those of large propensities have lower indexes. Similarly, the Sorting Direct Method [MPC$^+$06] uses a sorting algorithm to dynamically reorder the reactions. The Kinetic Monte Carlo method [Sch02] reuses intermediate data and performs the search (step 6 in algorithm 3.1) in O(log M). On the other hand, some procedures are based on the Quasi-steady state approximation [Gou05] to reduce the reaction set and then apply the SSA algorithm on the reduced system.

Besides, the standard SSA algorithms support only one event type (reaction type), namely a stochastic event. Therefore, they preserve the Markovian property. There are other extensions that do not take into account the Markovian property and thus they support other event types such as immediate and deterministic time delay (see

Section 3.6.1) [HLGM09]. Such additional events are necessary to implement a certain model logic in typical biological cases (see Chapter 6).

3.5 Hybrid Approach

Hybrid simulation of biochemical reaction networks using both stochastic and deterministic kinetics was studied in [HR02, KMS04, SK05, ACT$^+$05, GCPS06, HL07]. The main idea of this approach is to partition the entire set of reactions into two different subsets: slow and fast [Pah09]. Slow reactions occur infrequently and they might be responsible for unexpected model behaviours (e.g., noise, volume variation, molecule fluctuations). Therefore, they are better to be stochastically simulated. Contrary, fast reactions occur frequently and it is better, from the simulation efficiency point of view, to simulate them continuously.

Stochastic simulation can be done using one of the algorithms discussed in Section 3.4, while continuous simulation can be carried out using either ODEs or the chemical Langevin equation. However, the most important and challenging questions of the hybrid simulation are the partitioning of reactions and the synchronisation of the stochastic and continuous regimes.

Partitioning the reactions into slow and fast ones might seem to be an easy task. However, it is critical for a successful hybrid simulation algorithm to have an efficient partitioning scheme. For instance, inefficient partitioning might result in a hybrid simulation which is slower than the stochastic one. Moreover, in order to simulate fast reactions continuously, their reactants have to satisfy the thermodynamic condition (i.e., the number of molecules and the volume of the system approach infinity) [Gil07]. Nevertheless, partitioning of reactions can be done either off-line (static) before the simulation starts, or on-line (dynamic) while the simulation is running.

Similarly, synchronisation of the two simulation regimes is vital to the accuracy of the simulation result. The continuous and stochastic simulators are not isolated from each other, instead there is a mutual influence between them. Indeed, continuously simulated reactions depend on the state of the stochastic simulator and the propensities of stochastic reactions are changing with time when the continuous simulator advances [HR02].

In the sequel, we discuss reaction partitioning and the simulator synchronisation in more detail.

3.5.1 Reaction Partitioning

An initial step to use hybrid simulation algorithms is to tell the simulation engine which reactions to simulate continuously and which ones are simulated stochastically. This problem can be solved using either an off-line (static) approach (i.e., perform the partitioning independently from the simulation), or using an on-line (dynamic) method

(i.e., re-check and repartition reactions during the simulation). In the following, we discuss the static and dynamic approaches independently.

Static Partitioning

Using a static partitioning scheme, reactions are partitioned off-line before the simulation starts. Partitioning can be specified by the user or by providing threshold values and to let the simulator performs the partitioning during the initialisation. Usually, users determine such partitioning using some knowledge of previously executed stochastic or continuous simulation (single run only).

A major advantage of this approach is that it does not require any additional computational overhead during the simulation to perform the partitioning. Moreover, it is easy to be understood by the modeller, since reactions can easily be enforced to be simulated stochastically or continuously.

Nevertheless, rates of biochemical reactions are often not constant over time. They may dynamically change with respect to time, since they depend on the current state of the reactants. Slow reactions can change themselves into fast ones during the simulation and vice versa. For instance, simulating an oscillating model may result in reaction rates that oscillate during the simulation between slow and fast. Thus dynamic partitioning is useful in this case.

Dynamic Partitioning

To deal with the drawback of static partitioning, dynamic partitioning checks repeatedly the current rates of transitions during the simulation. It repartitions them into stochastic and continuous ones if they have been substantially changed. Such partitioning scheme can correctly deal with dynamically changing rates and therefore increases the simulation efficiency. Reactions can be partitioned according to the current rates, current number of molecules of reactants, or using combinations of these two criteria.

Using the current reaction rates to perform the partitioning, transitions are partitioned into slow and fast through pre-defined thresholds. High threshold values produce accurate results, while low values speed-up the simulation. However, partitioning using reaction rates only does not guarantee that the thermodynamic condition will be fulfilled. Thus, other partitioning criteria are required.

Dynamic partitioning using a threshold of the species' number of molecules amends the drawback of partitioning using only reaction rates. In this method, species are partitioned according to their current number of molecules into two subsets: those which are below a certain threshold (group 1) and those which are above the threshold value (group 2). A reaction is simulated continuously only if all of its reactants belong to group 2.

A combination of those two methods is frequently used (e.g., in [ACT$^+$05, GCPS06]).

That is the simulator obtains two thresholds to dynamically perform the partitioning. One threshold is used for the place markings while another one is used for the transition rates. A reaction is simulated continuously if the following two conditions are satisfied:

- the current reaction rate is above a certain threshold
- all of the reaction's reactants are above a certain (other) threshold

Although the dynamic partitioning approach can efficiently deal with the problem of dynamically changing reaction rates, it introduces new computational challenges to the simulation. This additional overhead is due to the many rechecking and repartitioning of reactions. Therefore, it is recommended that the user selects in advance the appropriate partitioning scheme for the model under study.

In Chapter 4, we introduce the dynamic partitioning for hybrid Petri nets and provide two additional thresholds to help to efficiently simulate and represent biochemical networks.

3.5.2 Simulator Synchronisation

The hybrid simulation of biochemical networks consists basically of the coordination of continuous and stochastic simulation modules. Discrete solvers are usually asynchronous (i.e., the time steps are random), while continuous solvers are often synchronous (i.e., the time steps are deterministic) [KMS04]. Early developed hybrid algorithms are based on heuristic ideas, where the occurrence times of stochastic events are heuristically captured. On the other hand, mathematically founded algorithms are based on the mathematical relationship between stochastic events and the continuous simulation. Examples of these approaches are given below.

Heuristic Approach

As one example of the hybrid algorithms that are based on the heuristic approach, we consider here the one by Kiehl et al. [KMS04]. In this algorithm, the set of reactions are partitioned into two regimes: discrete and continuous. Reactions in the continuous regime are implemented using the ODEs integrators. The Runge-Kutta algorithm with fixed or variable step size is used. The partitioning leaves some species being represented in both regimes due to their participation in reactions which are simulated using continuous and discrete simulators. These species are called bridging species. They have two representations: one is a molecular representation (non-negative integer values) and the other one is concentrations (non-negative real values).

The algorithm starts by calculating the putative time of the stochastically simulated reactions. Then, the ODEs integrator is advanced to $\tau + \Delta\tau$, where $\Delta\tau$ is the minimum of the integrator step size (h) or the stochastic step ($\delta\tau$). At the end of each ODEs

step, the time-varying propensities are calculated and a putative time for the next stochastic event is calculated (τ_1). The system is updated using either the time and the state values of the ODE integrator if ($\tau_1 > t + \Delta\tau$), or using the stochastic time and the states of the continuous variables in addition to the firing of the occurred stochastic event.

Another example of algorithms that also fit in this category is the one presented in [TKHT04], where an embedded meta-algorithm is used. They combine multiple simulation algorithms to solve a multi-time scale model. The method consists of: (1) data structure, which manages the model specification and execution, (2) driver algorithm, which describes the synchronisation between the different solvers, and (3) integration algorithm which specifies the procedure by which the state variables are updated.

Although this approach can combine different simulation algorithms, it can not guarantee that events are accurately detected by stochastic and continuous simulators. With other words, the proposed interrupt function of each algorithm is left to the user to define it. Moreover, the process of how to assign state variables to each algorithm has not been discussed.

Exact and Approximate Approach

Due to the combination of both deterministic and stochastic reactions in the hybrid simulation approach, the propensities of the stochastic reactions depend on the state change of deterministically simulated reactions [HR02, SK05, Pah09]. Gillespie [Gil91] derived the correct reaction probability density function for this case as

$$P(\delta\tau, \mu|\mathbf{X}(\tau), \tau) = a_\mu(\mathbf{X}(\tau + \delta\tau)) \exp\left(-\int_\tau^{\tau+\delta\tau} a_0(\mathbf{X}(\tau))d\tau\right). \tag{3.27}$$

In [HR02], fast reactions are represented by a continuous Markov process being coupled with a Markov jump process for slow reactions where the continuous-time Markov chain is approximated by ODEs. However, they do not consider time varying propensities for slow reactions; instead a probability is introduced that no reaction occurs to decrease the approximation error [Pah09]. Other hybrid methods, for example in [ACT+05, GCPS06], consider time-varying propensities of slow reactions using (3.28).

$$g(\mathbf{x}) = \int_\tau^{\tau+\delta\tau} a_0^s(\mathbf{x})d\tau - \xi = 0\,, \tag{3.28}$$

where ξ is a random number exponentially distributed with a unit mean, and $a_0^s(\mathbf{x})$ is the cumulative propensity of slow reactions.

Using (3.28), the hybrid simulation algorithm can switch between deterministic and stochastic simulation by integrating the set of ODEs representing fast reactions along

with the cumulative propensity, $a_0^s(\mathbf{x})$, till (3.28) is satisfied, which means that a stochastic event has to occur. Then, a stochastic reaction R_μ is selected such that

$$\sum_{j=1}^{\mu-1} a_j^s(\mathbf{x}) < r_2 a_0^s(\mathbf{x}) \leq \sum_{j=1}^{\mu} a_j^s(\mathbf{x})\,, \tag{3.29}$$

where $a_i^s(\mathbf{x})$ is the propensity of the *i-th* slow reaction.

Later in Chapter 4, we not only have to detect stochastic events, but also other event types such as immediate and deterministic events. Immediate events represent the firing of an immediate transition while deterministic events represent the firing of a deterministically delayed transition and/or scheduled transition (see Section 3.6).

The algorithms discussed so far focus only on the simulation aspects. There are however other approaches which model the biochemical networks using a different primitives. Petri nets are one of these approaches.

3.6 Petri Nets

Petri nets are weighted, directed, bipartite graphs. They consist of two types of nodes: places, and transitions. Arcs are used to connect different node types. They are directed (i.e., from places to transitions or vice versa). Places are usually used to represent passive system components (e.g., conditions, resources, etc.), while transitions are used to represent active system components (e.g., events, processes, etc.). Places can carry non-negative integer values called place markings or tokens. Arcs are associated with positive integer values which are called arc weights.

The original ideas of Petri nets was introduced in Carl Adam Petri's dissertation [Pet62] and initially called place/transition nets. They are an excellent mathematical and graphical modelling formalism for describing and studying systems that are characterized by being concurrent, asynchronous, distributed, parallel or non-deterministic.

Since the ideas of Carl Adam Petri, many extensions have been proposed. Among the most prominent extensions are deterministically timed, stochastic, continuous and hybrid Petri nets. They have been applied in many discipline, e.g., industry, academia, circuit design, communication protocols, distributed computing, production systems, manufacturing, transportations, ecosystems, and *systems biology*.

In the biochemical network context, places may represent species (e.g., genes, proteins, mRNA, etc.), while transitions represent reactions (e.g., degradation, translation, transcription, association, disassociation, phosphorylation, etc.). The place markings represent the number of molecules of certain species, arc weights correspond to the stochiometry of the chemical reactions, pre-places are the reactants, and post-places the products. Before continuing with formally introducing Petri nets, it is useful to

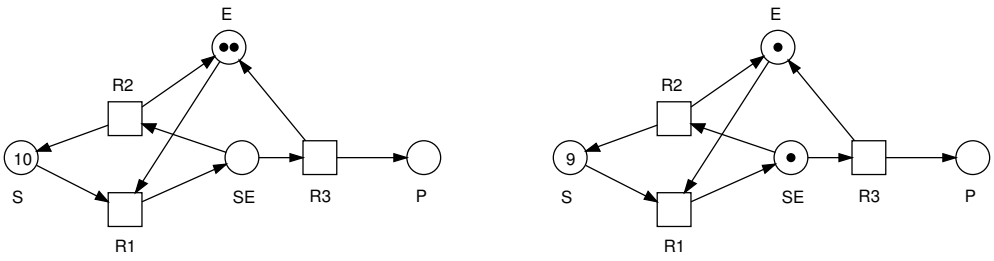


Figure 3.1: Petri net representation of the reaction set in (3.5). Pre-places represent reactants, post-places represent products, transitions represent reactions, tokens represent molecules. (b) single firing of the transition R_1 in Figure 3.1a. One token is removed from each A, E and added to AE.

define the following notations.

Notation 1 *The current marking of a place p_i is denoted by $m(p_i)$, ${}^\bullet t_j$ and $t_j^\bullet$ denote the pre-places and post-places of a tradition t_j, respectively, ${}^\bullet p_i$ and $p_i^\bullet$ denote the pre- and post-transitions of a place p_i, respectively.*

As an illustrative example consider again the enzyme-catalysed reaction in equation (3.5). Figure 3.1 gives the Petri nets representation of the reaction set in (3.5). Notice how the Petri net notions are intuitive to graphically represent the biochemical reaction network. Notice that $m(p_i)$ is equivalent to x_i which have been introduced in Section 3.2.

In addition to the static syntax of Petri nets, they have also a dynamic semantics. The dynamics of Petri nets are defined by enabling and firing of transitions.

A transition t is called enabled if each of its pre-places contains at least the number of tokens specified by the weight of the corresponding arc. For example, the transition R_1 in Figure 3.1a is enabled, while transition R_2 in the same figure is not enabled.

If a transition is enabled, it may fire and the net reaches a new marking state. The firing of a transition t involves removing some tokens from its pre-places (${}^\bullet t$) and adding some tokens to its post-places ($t^\bullet$). The number of added (removed) tokens is equal to the arc weight that connect t with the post-place (pre-place) p. For example, the Petri net in Figure 3.1b is the same one as in Figure 3.1a, but after a single firing of transition R_1. Firing of a transition corresponds to the occurrence of a reaction in the biochemical context.

Additionally, Petri nets can be classified into different classes based on one or more properties. For instance, a Petri net is called pure, if it has no self-loop, and it said to be ordinary if all of its arc weights are 1's.

Moreover, Petri nets are characterised by some interesting behavioural and structural properties. The former ones may be marking-dependent while the latter depend only on the topological structure of the Petri nets. Examples of behavioural properties are: reachability, boundedness, liveness, coverability, persistence and reversibility. Examples of structural properties are: conservativeness, and repetitiveness. Detailed discussion of these properties can be found in [Mur89].

In the sequel, we formally define Petri nets and some of the well known extensions which have been published in the context of *systems biology*. In Chapter 4, we define a new class of Petri nets which combines all features from the classes that are discussed in this chapter.

Definition 3.1 (Petri nets) *Petri nets are 5-tuple* $N = < P, T, A, F, m_0 >$ *where:*

- P *is a finite, non-empty, and disjoint set of places.*
- T *is a finite, non-empty set of transitions.*
- $A \subseteq (P \times T) \cup (T \times P)$ *is a finite set of directed arcs.*
- $F : A \to \mathbb{N}$ *is a function that assigns a positive integer number to each arc* $a \in A$.
- $m_0 : P \to \mathbb{N}_0$, *is a function that assign a non-negative integer number to each place as the initial marking.*

Here $\mathbb{N}$ *and* $\mathbb{N}_0$ *denote the sets of positive and non-negative integer numbers, respectively.*

□

Definition 3.2 (Enabling condition of Petri nets) *Let* $N = < P, T, A, F, m_0 >$ *be a Petri net and* m *the current marking of* N. *A transition* $t_j \in T$ *is enabled in the marking* m, *denoted by* $m[t_j\rangle$, *iff* $\forall p_i \in {}^\bullet t_j$, $m(p_i) \geq F(p_i, t_j)$.

□

Definition 3.3 (Firing rule of Petri nets) *Let* $N = < P, T, A, F, m_0 >$ *be a Petri net,* m *the current marking of* N, *and* $t_j \in T$ *a transition enabled in the marking* m. *A transition* t_j *can fire and reach a new marking* m', *denoted by* $m[t_j\rangle m'$, *with:*

- $\forall p_i \in {}^\bullet t_j$

$$m'(p_i) = m(p_i) - F(p_i, t_j)$$

- $\forall p_i \in t_j^\bullet$

$$m'(p_i) = m(p_i) + F(t_j, p_i)$$

□

3.6.1 Stochastic Petri Nets

Starting with the standard Petri nets, which contain only one type of place; transition; and arc, many extensions were proposed. In these discrete classes of Petri nets, the discrete state space description is preserved (e.g., see [KBD+94, GHL07, HLGM09]). One popular extension is the introduction of non-standard arcs (extended arcs) which yields extended Petri nets [HLGM09]. Another extension is the introduction of time as a delay to either places or transitions [KBD+94, GHL07].

The most popular extensions of net arcs are the read and inhibitor arcs. They are used to simplify the modelling process. A read arc is used to replace two reciprocal standard arcs with the same weight, while an inhibitor arc is introduced to inhibit the enabling of a transition when the marking of a place is greater than or equal to a certain threshold. Figure 3.2 illustrates the semantics of read and inhibitor arcs.

Unlike extending the standard Petri net definition by defining new arcs, some extensions are focused on associating time to transitions or places [KBD+94]. Assigning time to places is not frequently used. Time can be assigned to transitions, but the interpretation of the delay may have different meaning. Nevertheless, the time delay could be deterministic or stochastic. In the former, transitions are fired after a deterministically delayed time, while in the latter, the delay time is a random variable exponentially distributed.

Due to space limit, it is not viable to delineate separately all of the variants of extended and stochastic Petri nets, instead; we will consider one extension which combines most often used features in the context of *systems biology*: extended stochastic Petri nets ($\mathcal{XSPN}_{bio}$) [MRH12].

$\mathcal{XSPN}_{bio}$ extend the standard Petri nets by including: immediate, deterministically timed, scheduled and stochastic transitions. Moreover, they provide different arc types: standard, read, inhibitor, reset, and modifier arcs.

Definition 3.4 (Extended stochastic Petri nets) *$\mathcal{XSPN}_{bio}$ are a 6-tuple $\mathcal{XSPN}_{bio} = [P, T, A, F, V, m_0]$ where:*

- *P is a finite, non-empty, and disjoint sets of places.*
- *$T = T_{stoch} \cup T_{im} \cup T_{timed} \cup T_{scheduled}$ with:*
  1. *T_{stoch} is the set of stochastic transitions, which fire stochastically after an exponentially distributed waiting time.*
  2. *T_{im} is the set of immediate transitions, which fire with waiting time zero; they have higher priority compared with other transition types.*
  3. *T_{timed} is the set of deterministically delayed transitions, which fire after a deterministic time delay.*

4. *$T_{scheduled}$ is the set of scheduled transitions, which fire at predefined time points.*

- *$A = A_{standard} \cup A_{inhibit} \cup A_{read} \cup A_{equal} \cup A_{reset} \cup A_{modifier}$ is the set of directed arcs, with:*

  1. *$A_{standard} \subseteq ((P \times T) \cup (T \times P))$ defines the set of standard arcs.*
  2. *$A_{read} \subseteq (P \times T)$ defines the set of read arcs.*
  3. *$A_{inhibit} \subseteq (P \times T)$ defines the set of inhibitor arcs.*
  4. *$A_{equal} \subseteq (P \times T)$ defines the set of equal arcs.*
  5. *$A_{reset} \subseteq (P \times T)$ defines the set of reset arcs.*
  6. *$A_{modifier} \subseteq (P \times T)$ defines the set of modifier arcs.*

- *$F : A \to \mathbb{N}$ is a function which assigns a positive integer number to each arc as the arc weight.*

- *V is a set of functions $V = \{g, d, w\}$ where :*

  1. *$g : T_{stoch} \to H_s$ is a function which assigns a stochastic hazard function h_{s_t} to each transition $t_j \in T_{stoch}$, whereby $H_s = \{h_{s_t} | h_{s_t} : \mathbb{R}_0^{|^\bullet t_j|} \to \mathbb{R}_0^+, t_j \in T_{stoch}\}$ is the set of all stochastic hazard functions, and $g(t_j) = h_{s_t}, \forall t_j \in T_{stoch}$.*
  2. *$w : T_{im} \to H_w$ is a function which assigns a weight function h_w to each immediate transition $t_j \in T_{im}$, such that $H_w = \{h_{w_t} | h_{w_t} : \mathbb{R}_0^{|^\bullet t_j|} \to \mathbb{R}_0^+, t_j \in T_{im}\}$ is the set of all weight functions, and $w(t_j) = h_{w_t}, \forall t_j \in T_{im}$.*
  3. *$d : T_{timed} \cup T_{scheduled} \to \mathbb{R}_0^+$, is a function which assigns a constant time to each deterministically delayed and scheduled transition representing the (relative or absolute) waiting time.*

- *$m_0 : P \to \mathbb{N}_0$, is a function which assign a non-negative integer number to each place as the initial marking.*

Here $\mathbb{R}_0^+$ denotes the set of non-negative real numbers.

□

Please note that the arc weights of reset and modifier arcs are always set to be one which is a special case of the function F.

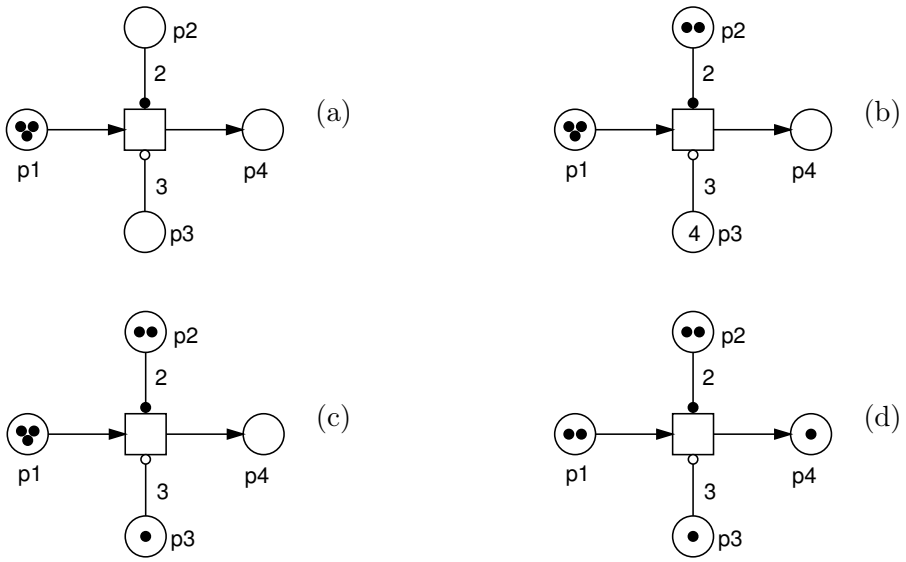


Figure 3.2: Examples of extended arcs. p_1, p_2, p_3 are connected with standard, read, and inhibitor arcs, respectively. (a) the net is not enabled, because $p_2 < 2$, (b) the net is also not enabled, because $p_3 > 3$, (c) the net is enabled, (d) the net of (c) after firing of the transition, the markings of p_2 and p_3 do not change when the transition fires.

Semantics of $\mathcal{SPN}$

The semantics of an $\mathcal{SPN}$ model is the continuous time Markov chains (CTMC), therefore it can be simulated using one of the SSA algorithms presented in Section 3.4. However, The extensions of $\mathcal{SPN}$ discussed in this section destroy the Markovian property. Nevertheless, a few extensions can be added to SSA algorithms to produce the semantics of $\mathcal{XSPN}$ on the simulation level.

Figure 3.3 presents the simulation results of the Petri nets in Figure 3.1, when it is read as a stochastic Petri nets. The transition rate are mass-action kinetics with a constant rate of 0.1.

3.6.2 Continuous Petri Nets

To model systems with large number of states, it is extremely hard or even impossible to use the discrete Petri net extensions. Continuous Petri nets were introduced in [DA87] to overcome such problems. In continuous Petri nets, the discrete token values of places are replaced with continuous values (also called fluid marking). Transitions fire con-

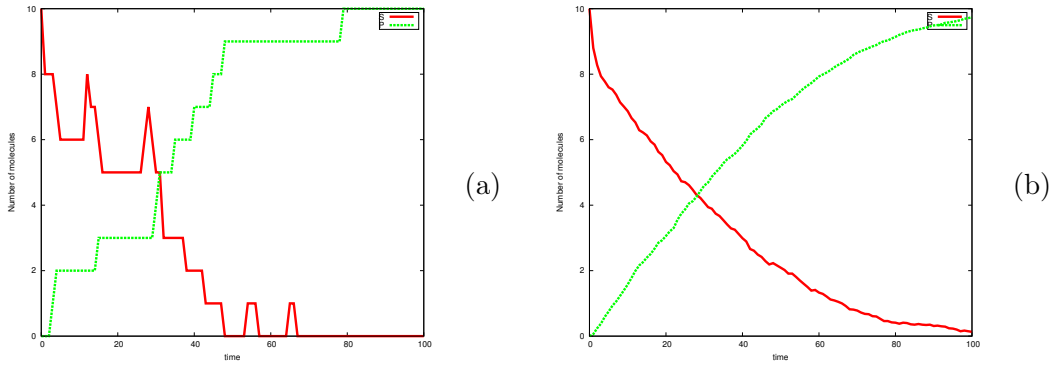


Figure 3.3: Stochastic simulation results of the Petri net in Figure 3.1. (a) single run, and (b) average of 100 runs .

tinuously with time. Similar to the discrete Petri nets, many extensions were proposed to the original idea of continuous Petri nets. A recent survey of the different classes of continuous Petri nets can be found in [DA10].

For our purpose of simulating biochemical networks, we are interested in a certain class of Petri nets: continuous transitions with maximum firing speed depending on time [DA10].

Definition 3.5 (Continuous Petri nets) *Continuous Petri nets are 6-tuple* $N = < P, T, A, F, v, m_0 >$ *where:*

- P *is a finite, non-empty, and disjoint sets of continuous places.*
- T *is a finite, non-empty set of continuous transitions.*
- $A \subseteq (P \times T) \cup (T \times P)$ *is a finite set of directed arcs.*
- $F : A \to \mathbb{Q}^+$ *is a function which assigns a positive real number to each arc* $a \in A$.
- $v : T \to H$ *is a function which assigns a firing rate function* h_t *to each transition* $t_j \in T$, *whereby* $H_c = \{h_t | h_t : \mathbb{R}_0^{|{}^\bullet t_j|} \to \mathbb{R}_0^+, t_j \in T\}$ *is the set of all firing rate functions, and* $v(t_j) = h_t, \forall t_j \in T$.
- $m_0 : P \to \mathbb{R}_0^+$, *is a function which assigns a non-negative real number to each place as the initial marking.*

Here $\mathbb{Q}^+$ *denote the set of positive rational numbers.*

□

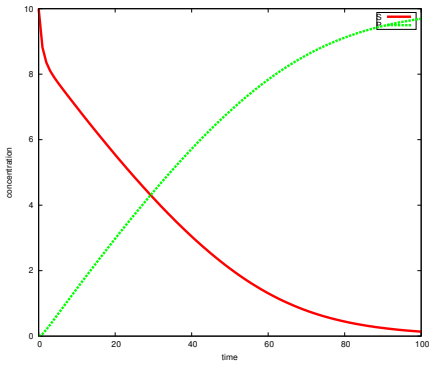


Figure 3.4: Continuous simulation results of the Petri net in Figure 3.1.

Semantics of $\mathcal{CPN}$

The specific continuos Petri nets (continuous Petri nets with transitions of maximum firing speeds depending on time) have been introduced in *systems biology* as biochemically interpreted continuous Petri nets ($\mathcal{CPN}_{bio}$) [GH06]. This net can provide a convenient means of describing ODEs in a structure-oriented manner. Each transition t_j is associated with a rate function $v_j(\tau)$ which defines its generally state-dependent kinetic rate (i.e., reaction propensity in the biochemical context). The corresponding ODE which describes the change of the concentration of the species p_i is generated by (3.30), see e.g., [GH06],

$$\frac{dm(p_i)}{d\tau} = \sum_{t_j \in {}^\bullet p_i} F(t_j, p_i) v_j(\tau) - \sum_{t_j \in p_i^\bullet} F(p_i, t_j) v_j(\tau) \tag{3.30}$$

Note that place names are here read as real-valued variables. Equation (3.30) corresponds to (3.1).

Figure 3.4 presents the simulation results of the Petri nets in Figure 3.1, when it is read as a continuous Petri net. The same kinetics are also used as in simulating the same example stochastically. In this example, we can notice the similarity between the single simulation run result of the continuous simulation and the average runs of the stochastic one in Figure 3.3. In fact, the stochastic simulation results will be identical to the average stochastic results if many runs are executed for this example.

3.6.3 Hybrid Petri Nets

Hybrid Petri nets [AD98, DA10] incorporate both discrete and continuous capabilities and can be used to model systems which contain both discrete and continuous elements. The classical example used to demonstrate the idea of HPN is the water tank model.

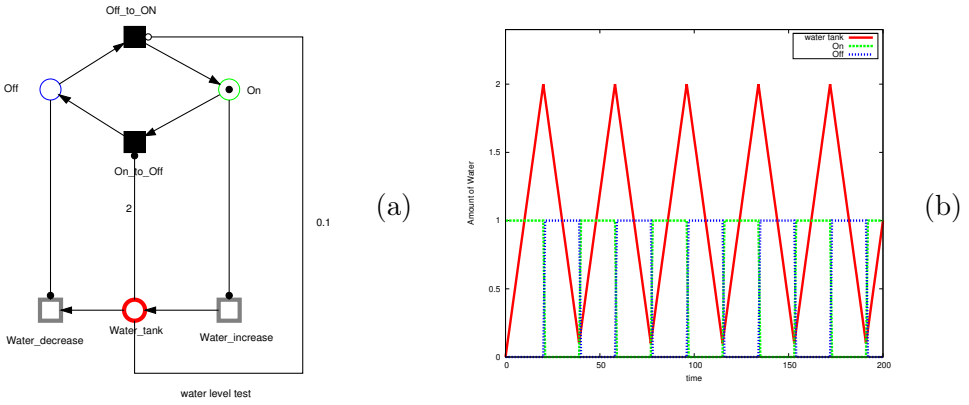


Figure 3.5: An example of Hybrid Petri Nets: the water tank model (a) HPN representation, (b) simulation result. Two discrete places are used to represent the discrete system states (on/off). A continuous place is used to model the current amount of water in the tank. The minimum and maximum amount of water in the tank are 0.1, and 2, respectively. The continuous transition *water_ increase* increases the amount of water by a constant rate (0.1) when the system is in the *on* state, while the transition *water_ decrease* decreases the amount of water by a constant rate 0.1 when the system is in the *off* state. The discrete transitions, *on_ to_ off* and *off_ to_ on*, read the current amount of water and switch the system state respectively from *on* to *off*, or *off* to *on*. Notice that how read arcs are used to read the system state, while inhibitor arc is used to inhibit the transition *off_ to_ on* based on the amount of water in the continuous place *water_ tank*

Figure 3.5 represents the water tank model using HPN semantics. Obviously, this model requires discrete and continuous places. The former are used to represent discrete states (on/off), while the latter are used to model water flow.

Many variations of hybrid Petri nets have been introduced during the last two decades, with different modelling goals. In the following we briefly discuss some of the HPN classes which are used in the literature.

An interesting class of hybrid Petri nets are the **Differential Petri Nets (DPNs)** [DK98]. DPNs contain differential places (equivalent to continuous places), differential transitions (equivalent to continuous transitions), discrete places and discrete transitions. They have been introduced for design and performance analysis of industrial supervisory control systems. Continuous transitions fire continuously with time and they are associated with a rate and delay. The delay is equivalent of the ODE integrator (fixed or variable) time step size. The main difference between DPNs and other HPNs is that differential places and arc weights could contain negative values. This

assumption is useful in some applications where ODE variables can contain negative values. Moreover, DPNs are conflict free (see Section 4.2.6).

Hybrid Dynamic Nets (HDN) [Dra98] allow any function for defining state-dependent transition rates, without structural restrictions. Contrary, in our net class, we restrict the domain of rate functions to the transitions' pre-places (see Definition 4.2.2). This constraint is very useful in biological context and crucial for the efficiency of our tools, since the reactions' rates are calculated in terms of the reactions' substrates. We provide a special arc type called modifier to allow any place in the transitions' rate functions.

Hybrid Functional Petri Nets (HFPN) were introduced in [MTA+03] to allow any function to be assigned as an arc weight or as transition delay. Hybrid Functional Petri Nets with extension (HFPNe) [NDMM04] extend HFPN by generic entities and generic data types. However, dynamic partitioning of transitions into discrete and continuous ones is not considered. In [YLL09], transitions can be simulated in an adaptive way, but distinction between discrete and continuous places is not supported. Other transition types (e.g., immediate transitions) are not supported neither.

Contrary, **Fluid Stochastic Petri Nets** (FSPNs) [TK93] combine both stochastic and continuous net parts into one net class. However, they suffer from unclear and inconsistent graphical representations [HK99] which make them inappropriate for our purpose of representing and simulating biochemical networks (for an example see Section 4.7.1). More importantly, they do not support the full range of deterministically delayed transitions as we do.

Finally, another example of HPN classes are the **First-Order Hybrid Petri Nets** (FOHPNs) [BGM00]. FOHPN are able to model systems whose first-order continuous behaviour can be studied by linear algebraic tools. A distinguished feature of FOHPNs is the use of linear programming to compute transition instantaneous firing speeds (rates). They consider the problem of transition speed calculation as an optimisation problem and try to find at each time step the appropriate rates of transitions.

In this thesis we are mainly interested in this specific class of hybrid Petri nets. In the next chapter a new hybrid Petri net class in introduced which combines both of the features of extended stochastic Petri nets and continuous Petri nets, introduced in subsections 3.6.1 and 3.6.2, respectively.

3.6.4 High-Level Petri Nets

With the increasing size of biological models, low-level Petri nets do not scale [Liu12]. Therefore, there is a need to find a better way to easily manage big models with repetitive components. Two scenarios can be used to deal with such models. Hierarchical Petri nets [Feh93] and coloured Petri nets [Jen95]. High-level Petri nets can not be considered independently from the low-level one. For instance, the majority of analysis and simulation techniques of coloured Petri nets are mainly based on the uncoloured

one by unfolding.

Hierarchical Petri nets facilitate the reuse of model building blocks to design larger ones. Although hierarchy does not add any power to the Petri net, it is important to design large non-trivial models using Petri nets [Feh93]. Subnets can be represented as places or transitions. Moreover, hierarchical Petri nets provide a systematic scenario to view the model under different levels of details.

Coloured Petri nets can represent a group of similar components by one colour component, each component is represented and defined in terms of a colour. An interesting advantage of coloured Petri nets is that the model size can be easily increased [Liu12]. Coloured extensions of all the Petri net class discussed earlier in this chapter have been presented in the literature (e.g., see [Liu12]). Our computational steering framework, which will be presented in Chapter 5, also supports coloured Petri nets.

3.7 Closing Remarks

In this chapter we have surveyed previous work in the field of biochemical modelling and Petri nets. From our discussion we can draw and re-consider the following remarks:

- With the progress of *systems biology*, it becomes vital to consider substantially larger computational models. These models can combine subnets of different scales. Therefore, a continuous or a stochastic approach becomes inefficient to produce the dynamics of these systems. Thus, it is evident that hybrid simulation is of paramount importance for the simulation and analysis of biochemical networks.

- For each biochemical modelling approach (deterministic, stochastic, or hybrid), there is a corresponding Petri net class which can solve the same problem but in a more elegant and intuitive way. A Petri net model has the advantage of structural and graphical representation compared with the traditional approaches (e.g., using only the ODE solvers to simulate the biochemical model). It has a dual role: graphical and mathematical.

- Although many hybrid Petri net classes have been introduced in the literature, none of them provides the full interplay between stochastic and continuous transitions, as required particularly in the context of biochemical network modelling.

Therefore, we are going to introduce Generalised Hybrid Petri Nets in the next chapter. $\mathcal{GHPN}_{bio}$ provide a tight coupling of the stochastic and deterministic modelling approaches. The hybrid modelling capabilities of $\mathcal{GHPN}_{bio}$ are not only on the simulation level but also on the representation one. $\mathcal{GHPN}_{bio}$ cover almost all of the open issues that have been discussed in this chapter.

4 Generalised Hybrid Petri Nets

4.1 Introduction

Motivated by previous work that has been presented in the literature and briefly surveyed in Chapters 1 and 3, we introduce in this chapter a new Petri nets class, Generalised Hybrid Petri Nets ($\mathcal{GHPN}_{bio}$) [HH12a] tailored to the specific needs of modelling and simulation of biochemical networks. It provides rich modelling and simulation functionalities by combining all features of Continuous Petri Nets and Extended Stochastic Petri Nets ($\mathcal{XSPN}$). Herein, we focus on modelling and simulation of stiff biochemical networks, in which some reactions are represented and simulated stochastically, while others are carried out deterministically. Additionally, two related simulation algorithms are presented, supporting static (off-line) partitioning and dynamic (on-line) partitioning. This chapter comes with a full-fledged implementation, supporting the introduced net class as well as the discussed simulation algorithms. The specific contributions of this part are:

- The introduction of $\mathcal{GHPN}_{bio}$ which are particularly well suited for the specific needs of systems biologists to represent and simulate different reaction types with convenient and rich modelling capabilities (different transition types and different arc types). Moreover, $\mathcal{GHPN}_{bio}$, compared with other hybrid Petri nets discussed in Chapter 3, provide the full interplay of stochastic and continuous transitions.
- The provision of two related algorithms to simulate the dynamics of $\mathcal{GHPN}_{bio}$ using either static partitioning in which the partitioning is done off-line before the simulation starts, or using dynamic partitioning in which the partitioning is done on-line during the simulation.
- Unlike previous work of studying stiff biochemical networks which focused on the simulation aspect only, this chapter considers representation as well as simulation features for a better understanding of such networks.
- All features discussed in this chapter are implemented in the platform-independent tool Snoopy [RMH10, HHL+12] which can be downloaded from [Sno12].

In simulating $\mathcal{GHPN}_{bio}$ we are more interested in a general simulation algorithms that allows the combination of any SSA and ODE solvers.

This chapter is organised as follows: using the related work which has been presented in Chapter 3, we start off with introducing $\mathcal{GHPN}_{bio}$ and show how they can be simulated using static or dynamic partitioning. More specifically, $\mathcal{GHPN}_{bio}$ are defined by formally specifying their syntax, enabling and firing rules, conflict resolution, and the generation of the corresponding ODEs. In Section 4.4 we discuss three implementation aspects related to the simulation of $\mathcal{GHPN}_{bio}$. Next, we discuss the relationship between $\mathcal{SPN}$, $\mathcal{CPN}$, and $\mathcal{GHPN}_{bio}$. To help to better position our contribution, we compare between $\mathcal{GHPN}_{bio}$ and four other similar hybrid Petri net classes. For the sake of completeness two simple examples are discussed in this chapter. More realistic biological case studies are postponed until Chapter 6. Finally, we sum up by some conclusions and closing remarks.

4.2 Generalised Hybrid Petri Nets

In this section, we discuss in more detail the different aspects of the Generalised Hybrid Petri Nets class. We start with its modelling capabilities of biological systems, specifically in simulating stiff biochemical networks, and explain how $\mathcal{GHPN}_{bio}$ models can be simulated.

4.2.1 Modelling

To model stiff biochemical networks, $\mathcal{GHPN}_{bio}$ combine both stochastic and continuous elements in one and the same model. Indeed, continuous and stochastic Petri nets complement each other. The fluctuation and discreteness can be conveniently modelled using stochastic simulation, and at the same time the computationally expensive parts can be simulated deterministically using ODE solvers. Modelling and simulation of stiff biochemical networks are outstanding functionalities that $\mathcal{GHPN}_{bio}$ provide for systems biology.

Generally speaking, biochemical systems can involve reactions from more than one type of biological network, for example gene regulation, metabolic pathways, or transduction pathways. Incorporating reactions that belong to distinct (biological) networks tends to result in stiff systems. This follows from the fact that gene regulation networks' species may contain small numbers of molecules, while metabolic networks' species may contain large numbers of molecules [KMS04].

In the rest of this section, we will discuss in more detail the newly introduced net class in terms of the graphical representation of its elements as well as the firing rules and connectivity between the continuous and stochastic net parts.

Elements

The $\mathcal{GHPN}_{bio}$ elements are classified into three categories: places, transitions and arcs. Figure 4.1 provides a graphical illustration of those elements.

$\mathcal{GHPN}_{bio}$ offer two types of place: discrete and continuous. Discrete places (single line circle) hold non-negative integer numbers which may represent the number of molecules of a given species (tokens in Petri net notions). On the other hand, continuous places - which are represented by the shaded line circle - hold non-negative real numbers which represent the concentration of a certain species. Please note that, except when otherwise mentioned, the number which a place p_i holds, also called its marking, is referred to by $m(p_i)$.

Furthermore, $\mathcal{GHPN}_{bio}$ offer five transition types: stochastic, immediate, deterministically delayed, scheduled, and continuous transitions [HGD08]. Stochastic transitions, which are drawn in Snoopy as a single line square, fire randomly with an exponentially distributed random delay. The user can specify a set of firing rate functions that determine the random firing delay. The transitions' pre-places can be used to define the firing rate functions of stochastic transitions. Immediate transitions (black bar) fire with zero delay, and have always highest priority in the case of conflicts with other transitions. They may carry weights (which can also be defined by a state-dependent functions) that specify the relative firing frequency in the case of conflicts between immediate transitions. Deterministically delayed transitions (represented as black squares) fire after a specified constant time delay. Scheduled transitions (grey squares) fire at user-specified absolute time points. Continuous transitions (shaded line square) fire continuously in the same way as in continuous Petri nets. Their semantics are governed by ODEs which define the changes in the transitions' pre- and post-places. More details about the biochemical interpretation of deterministically delayed, scheduled, and immediate transitions can be found in [HLGM09]. To simplify the presentation, we occasionally refer to stochastic, immediate, deterministically delayed or scheduled transitions as discrete transitions.

The connection between those two types of node (places and transitions) takes place using a set of different arcs. $\mathcal{GHPN}_{bio}$ offer six types of arc: standard, inhibitor, read, equal, reset, and modifier arcs. Standard arcs connect transitions with places or vice versa. They can be discrete, i.e., carry non-negative integer-valued weights (stoichiometry in the biochemical context), or continuous, i.e., carry non-negative real-valued weights. In addition to their influence on the enabling of transitions, they also affect the place marking when a transition fires by adding (removing) tokens from the transition's post-places (pre-places). For more details, see Section 4.2.2.

Extended arcs such as inhibitor, read, equal, reset, and modifier arcs can only be used to connect places to transitions, and not vice versa. A transition connected with an inhibitor arc is enabled (with respect to the corresponding pre-place) if the marking of the pre-place is less than the arc weight. In contrast, a transition connected with a

read arc is enabled if the marking of the pre-place is greater than or equal to the arc weight. Similarly, a transition connected using an equal arc is enabled if the marking of the pre-place is equal to the arc weight.

The other two remaining arcs do not affect the enabling of transitions. A reset arc is used to reset a place marking to zero when the corresponding transition fires. Modifier arcs permit one to include any place in the transitions' rate functions and simultaneously preserve the net structure restriction. Besides, the markings of places connected using read, inhibitor, equal, or modifier arcs does not change when the corresponding transition fires.

The connection rules and their underlying formal semantics are discussed in more detail below. Figure 4.1 provides a graphical illustration of all elements. Although this graphical notation is the default one, they can be customised easily using our Petri nets editing tool, Snoopy.

As a simple example for the above discussion, consider again the stiff biochemical network in (3.25) which is now illustrated in Figure 4.2. Using $\mathcal{GHPN}_{bio}$, the slow reaction R_3 can be modelled using a stochastic transition, while the other two fast reactions, R_1 and R_2, can be represented as continuous transitions. Places are partitioned into discrete and continuous ones based on the connection rules which will be discussed next.

Connection Rules

A critical question arises when considering the combination of discrete and continuous elements: how are these two different parts connected with each other? Figure 4.3 provides a graphical illustration of how the connection between different elements of $\mathcal{GHPN}_{bio}$ takes place.

First, we will consider the connection between continuous transitions and the other elements of $\mathcal{GHPN}_{bio}$. Continuous transitions can be connected with continuous places in both directions using continuous arcs (i.e., arcs with real-valued weights). This means that continuous places can be pre- or post-places of continuous transitions. These connections typically represent deterministic biological interactions.

A Continuous transitions can also be connected with discrete places, but only by one of the extended arcs (inhibitor, read, equal, and modifier). Read arcs allow to specify positive side conditions, while inhibitor arcs allow to specify negative side conditions. This type of connection permits a link between discrete and continuous parts of the biochemical model.

Discrete places are not allowed to be connected with continuous transitions using standard arcs, because the firing of continuous transitions is governed by ODEs which require real values in the pre- and post-places. Hence, this cannot take place in the discrete world. It is worth mentioning that some authors allow such connections, e.g., see [DK98, BGM00, DA10]. However, they impose additional conditions to ensure that

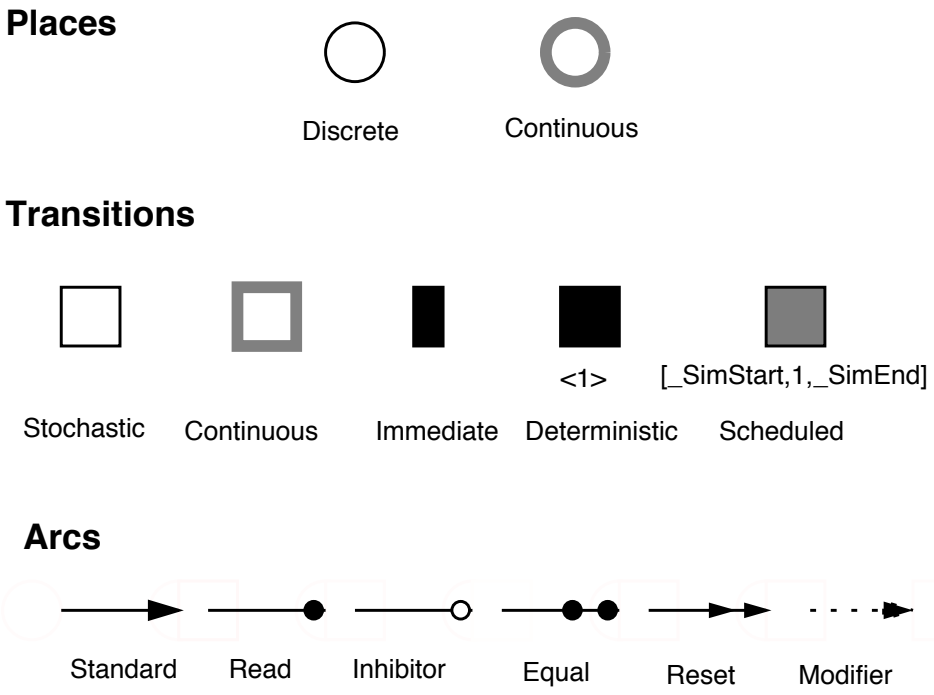


Figure 4.1: Graphical representation of the $\mathcal{GHPN}_{bio}$ elements [HH12a]. Places are classified as discrete and continuous; transitions as continuous, stochastic, immediate, deterministically delayed and scheduled; and arcs as standard, inhibitor, read, equal, reset, and modifier.

real values do not occur. For instance in [DA10], if an arc between a discrete place and a continuous transition exists, another reciprocal arc must exist with the same weight value. Second, discrete transitions can be connected with discrete or continuous places in both directions using standard arcs. However, the arc weights need to be considered. The connection between discrete transitions and discrete places takes place using arcs with non-negative integer numbers (i.e., discrete values), while the connection between continuous places and discrete transitions is weighted by non-negative real numbers (i.e., continuous values). The general rule to determine the weight type of arcs is the type of the connected place.

Figure 4.4 illustrates graphically the semantics of the different extended arcs that are included in $\mathcal{GHPN}_{bio}$. Notice how these arcs can simplify the modelling process. For instance, to model the situation where the current marking is reset when transition t fires, would require three transitions and several standard arcs. However, using a reset arc, these can be intuitively done, as shown in Figure 4.4.d. Note that the equivalence between a read arc and two reciprocal standard arcs, as shown in Figure 4.4.a, is valid

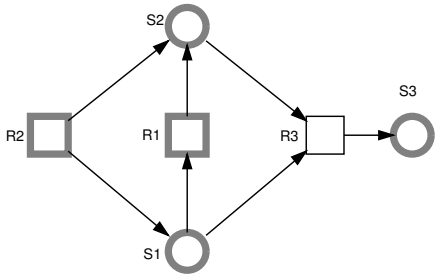


Figure 4.2: An example of a stiff biochemical network: $\mathcal{GHPN}_{bio}$ representation of the reactions in (3.25) Assuming mass-action kinetics with $c_1 = c_2 = 10^5$ and $c_3 = 0.0005$ and the initial state $x(0) = (10000, 10000, 100)$, reaction R_3 is much slower than R_1 and R_2.

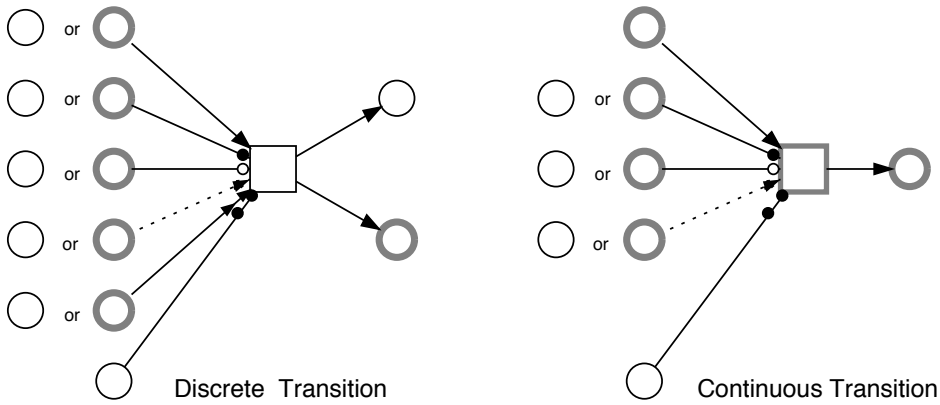


Figure 4.3: Possible connections between $\mathcal{GHPN}_{bio}$ elements. The restrictions are as follows. Discrete places cannot be connected with continuous transitions using standard arcs, continuous places cannot be tested with equal arcs, and continuous transitions cannot use reset arcs.

only under the interleaving semantics of Petri nets.

Connecting continuous places and discrete transitions will result in a model like in [TK93], in which changes in continuous places are governed by the firing of stochastic transitions. Discrete transitions can also have discrete or continuous pre-places using extended arcs.

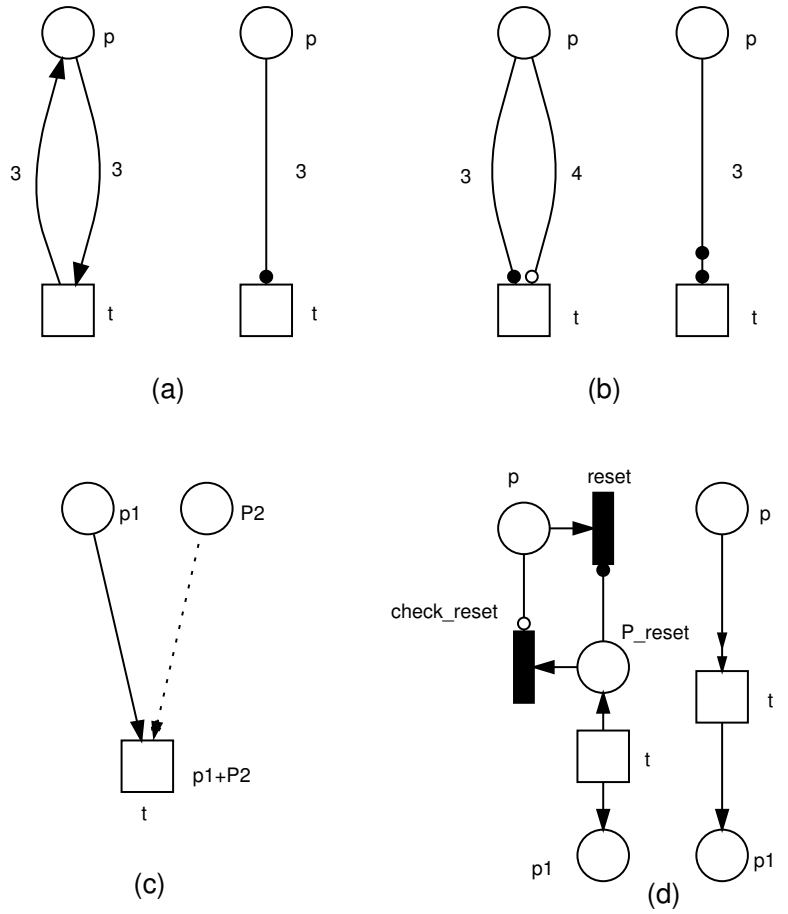


Figure 4.4: Arc semantics in $\mathcal{GHPN}_{bio}$: (a) a **read arc** replaces two standard arcs to check $m(p) \geq 3$, (b) an **equal arc** replaces two extended arcs (read and inhibitor arcs) to check if $m(p) = 3$, (c) a **modifier arc** is used to render the use of place p_2 in transition rate without any effect on the enabling of transition t (assuming that $p_1 + p_2$ is a rate function for the transition t), and (d) a **reset arc** replaces two immediate transitions, one place and four arcs to set $m(p) = 0$ when transition t fires.

4.2.2 Formal Definition

In this section, the syntax and semantics of $\mathcal{GHPN}_{bio}$ are formally defined. In Section 4.3, two algorithms are presented to simulate the semantics of $\mathcal{GHPN}_{bio}$.

Definition 4.1 (Generalised Hybrid Petri Nets) *Generalised Hybrid Petri Nets are a 6-tuple* $GHPN_{bio} = [P, T, A, F, V, m_0]$*, where* P*,* T *are finite, non-empty and disjoint sets.* P *is the set of places, and* T *is the set of transitions with:*

- $P = P_{disc} \cup P_{cont}$*, whereby* P_{disc} *is the set of discrete places to which non-negative integer values are assigned, and* P_{cont} *is the set of continuous places to which non-negative real values are assigned.*
- $T = T_{stoch} \cup T_{im} \cup T_{timed} \cup T_{scheduled} \cup T_{cont}$ *with:*
  1. T_{stoch} *is the set of stochastic transitions, which fire stochastically after an exponentially distributed waiting time.*
  2. T_{im} *is the set of immediate transitions, which fire with waiting time zero; they have higher priority compared with other transitions.*
  3. T_{timed} *is the set of deterministically delayed transitions, which fire after a deterministic time delay.*
  4. $T_{scheduled}$ *is the set of scheduled transitions, which fire at predefined time points.*
  5. T_{cont} *is the set of continuous transitions, which fire continuously over time.*
- $A = A_{cont} \cup A_{disc} \cup A_{inhibit} \cup A_{read} \cup A_{equal} \cup A_{reset} \cup A_{modifier}$ *is the set of directed arcs, with:*
  1. $A_{disc} \subseteq ((P \times T) \cup (T \times P))$ *defines the set of discrete arcs,*
  2. $A_{cont} \subseteq ((P_{cont} \times T) \cup (T \times P_{cont}))$ *defines the set of continuous arcs,*
  3. $A_{read} \subseteq (P \times T)$ *defines the set of read arcs,*
  4. $A_{inhibit} \subseteq (P \times T)$ *defines the set of inhibits arcs,*
  5. $A_{equal} \subseteq (P_{disc} \times T)$ *defines the set of equal arcs,*
  6. $A_{reset} \subseteq (P \times T^D)$ *defines the set of reset arcs,*
  7. $A_{modifier} \subseteq (P \times T)$ *defines the set of modifier arcs,*

  where $T^D = T_{stoch} \cup T_{im} \cup T_{timed} \cup T_{scheduled}$ *is the set of discrete transitions.*
- F *is a function*

$$F : \begin{cases} A_{cont} \to \mathbb{Q}^+, \\ A_{disc} \to \mathbb{N}, \\ A_{read} \to \mathbb{Q}^+, \\ A_{inhibit} \to \mathbb{Q}^+, \\ A_{equal} \to \mathbb{N}, \\ A_{reset} \to \{1\}, \\ A_{modifier} \to \{1\}. \end{cases}$$

which assigns a positive integer value or positive rational value as a weight to each arc depending on the arc type. If an arc is not explicitly weighted, traditionally, we assume a weight of 1.

- V *is a set of functions* $V = \{g, d, w, f\}$ *where :*

  1. $g : T_{stoch} \rightarrow H_s$ *is a function which assigns a stochastic hazard function* h_{s_t} *to each transition* $t_j \in T_{stoch}$*, whereby* $H_s = \{h_{s_t} | h_{s_t} : \mathbb{R}_0^{|^\bullet t_j|} \rightarrow \mathbb{R}_0^+, t_j \in T_{stoch}\}$ *is the set of all stochastic hazard functions, and* $g(t_j) = h_{s_t}, \forall t_j \in T_{stoch}$*.*
  2. $w : T_{im} \rightarrow H_w$ *is a function which assigns a weight function* h_w *to each immediate transition* $t_j \in T_{im}$*, such that* $H_w = \{h_{w_t} | h_{w_t} : \mathbb{R}_0^{|^\bullet t_j|} \rightarrow \mathbb{R}_0^+, t_j \in T_{im}\}$ *is the set of all weight functions, and* $w(t_j) = h_{w_t}, \forall t_j \in T_{im}$*.*
  3. $d : T_{timed} \cup T_{scheduled} \rightarrow \mathbb{R}_0^+$*, is a function which assigns a constant time to each deterministically delayed and scheduled transition representing the (relative or absolute) waiting time.*
  4. $f : T_{cont} \rightarrow H_c$ *is a function which assigns a rate function* h_c *to each continuous transition* $t_j \in T_{cont}$*, such that* $H_c = \{h_{c_t} | h_{c_t} : \mathbb{R}_0^{|^\bullet t_j|} \rightarrow \mathbb{R}_0^+, t_j \in T_{cont}\}$ *is the set of all rates functions and* $f(t_j) = h_{c_t}, \forall t_j \in T_{cont}$*.*

- $m_0 = m_{cont} \cup m_{disc}$ *is the initial marking for both the continuous and discrete places, whereby* $m_{cont} \in \mathbb{R}_0^{|P_{cont}|}$*,* $m_{disc} \in \mathbb{N}_0^{|P_{disc}|}$*.*

□

4.2.3 Semantics

The semantics of $\mathcal{GHPN}_{bio}$ is given in terms of the discrete and continuous transitions. To harmonise the mathematical notations, let $T^C = T - T^D = T_{cont}$ denote the set of continuous transitions.

Definition 4.2 (Enabling condition) *Let* $N = [P, T, A, F, V, m_0]$ *be a generalised hybrid Petri net and* m *be the marking of* N *at time* τ*. A transition* $t_j \in T$ *is enabled in the marking* m*, denoted by* $m[t_j\rangle$*, iff* $\forall p_i \in {}^\bullet t_j$*:*

- $m(p_i) \geq F(p_i, t_j)$*, if* $(p_i, t_j) \in A_{cont} \cup A_{disc} \wedge t_j \in T^D$*,*
- $m(p_i) > 0$*, if* $(p_i, t_j) \in A_{cont} \wedge t_j \in T^C$*,*
- $m(p_i) \geq F(p_i, t_j)$*, if* $(p_i, t_j) \in A_{read}$*,*
- $m(p_i) < F(p_i, t_j)$*, if* $(p_i, t_j) \in A_{inhibit}$*,*

- $m(p_i) = F(p_i, t_j)$, *if* $(p_i, t_j) \in A_{equal}$.

□

Definition 4.3 (Firing rule of discrete transitions) *Let* $N = [P, T, A, F, V, m_0]$ *be a generalised hybrid Petri net,* m *a marking of* N*, and* $t_j \in T^D$ *a transition enabled in the marking* m*,* $m[t_j\rangle$*, at time* τ*. The transition* t_j *can fire and reach a new marking* m'*, denoted by* $m[t_j\rangle m'$*, at time* $\tau + d_j$ *if it is still enabled at that new time, with:*

- $\forall p_i \in {}^\bullet t_j$

$$m'(p_i) = \begin{cases} m(p_i) - F(p_i, t_j) & \text{if } (p_i, t_j) \in A_{cont} \cup A_{disc} \\ 0 & \text{if } (p_i, t_j) \in A_{reset} \\ m(p_i) & else \end{cases}$$

- $\forall p_i \in t_j^\bullet$

$$m'(p_i) = m(p_i) + F(t_j, p_i)$$

where

$$d_j = \begin{cases} d(t_j) & \text{if } t_j \in T_{timed} \\ \tau + d(t_j) & \text{if } t_j \in T_{scheduled} \\ d_{stoch}(t_j) & \text{if } t_j \in T_{stoch} \\ 0 & \text{if } t_j \in T_{im} \end{cases}$$

is a delay which is associated to the discrete transition t_j *and* $d_{stoch}(t_j)$ *is random firing delays with negative exponential probability density function calculated for each stochastic transition* t_j *using its rate* $g(t_j)$*.*

□

According to the above enabling and firing definitions, discrete transitions follow a policy which is called an enabling memory policy [KBD+94].

Firing of continuous transitions The semantics of continuous transitions are analogue to the ones in continuous Petri nets with maximal firing speeds depending on time as introduced in [DA10] and tailored to the specific needs is systems biology [GH06]. The transitions' current firing rates (instantaneous firing speeds) depend on the current marking of their pre-places (i.e., species concentrations). In what follows, the firing semantics of continuous transitions are formally given.

We introduce the following notation. Let $v_j(\tau)$ represent the current firing rate of a transition $t_j \in T^C$ at time τ, $m_i(\tau) = m(p_i)$ denote the current marking of a place p_i at time τ, and $f_j(\tau)$=$f(t_j)$ denote the maximal firing rate of a transition t_j at time τ, then:

$$v_j(\tau) = \begin{cases} f_j(\tau) & \text{if } t_j \text{ is enabled} \\ 0 & \text{else} \end{cases} \tag{4.1}$$

Equation (4.1) implies that a continuous transition can fire with its maximal rate if it is enabled or its rate will be zero otherwise.

When a continuous transition is enabled, it fires as soon as possible and its effect on the connected places can be given by the following definition.

Definition 4.4 (Firing of continuous transitions) *Let $N = [P, T, A, F, V, m_0]$ be a generalised hybrid Petri net, m a marking of N, $t_j \in T^C$ a transition enabled in the marking m, $m[t_j\rangle$, at time τ, and $v_j(\tau)$ denotes the current firing rate of the transitions t_j. The transition t_j fires with:*

- $\forall p_i \in {}^\bullet t_j$

$$m_i(\tau + d\tau) = m_i(\tau) - F(p_i, t_j) \cdot v_j(\tau) d\tau \tag{4.2}$$

- $\forall p_i \in t_j^\bullet$

$$m_i(\tau + d\tau) = m_i(\tau) + F(t_j, p_i) \cdot v_j(\tau) d\tau \tag{4.3}$$

□

Equations (4.2) and (4.3) are called outflow and inflow of a place p_i, respectively, due to the firing of a transition t_j [DA10]. Summing up all inflow and outflow of a certain place will result in (3.30) (see page 53).

4.2.4 Generation of the Corresponding ODEs

For a given transition $t_j \in T^C$, the functions $read(w, p_i)$, $inhibit(w, p_i)$ are defined as follows:

$$read(w, m(p_i)) = \begin{cases} 1 & \text{if } m(p_i) \geq w \\ 0 & \text{else} \end{cases}$$

with $w = F(p_i, t_j) \wedge (p_i, t_j) \in A_{read}$, and

$$inhibit(w, m(p_i)) = \begin{cases} 1 & \text{if } m(p_i) < w \\ 0 & \text{else} \end{cases}$$

with $w = F(p_i, t_j) \wedge (p_i, t_j) \in A_{inhibit}$

Then the ODE corresponding to each continuous place in $GHPN_{bio}$ can be generated using (4.4)

$$\begin{aligned}\frac{dm\,(p_i)}{d\tau} = &\sum_{t_j \in {}^\bullet p_i} F\,(t_j, p_i) \cdot v_j\,(\tau) \cdot read(w, m(p_i)) \cdot inhibit(w, m(p_i)) - \\ &\sum_{t_j \in p_i {}^\bullet} F\,(p_i, t_j) \cdot v_j\,(\tau) \cdot read(w, m(p_i)) \cdot inhibit(w, m(p_i))\end{aligned} \tag{4.4}$$

Notice also that (4.4) is obtained from (3.30) by introducing read and inhibitor arcs. We will illustrate by examples the effects of extended arcs on the ODEs. Figure 4.5 is a $\mathcal{GHPN}_{bio}$ without any extended arc. Each place has an ODE, because each one is connected with a transition by a standard arc. The continuous transition rate is abbreviated by a pattern from the biochemical context, $MassAction(k)$ (see Section 3.3).

In Figure 4.6, p_4 is connected with the transition t using a read arc. Its effect is reflected in the ODEs by multiplying the transition's rate function by the aforementioned boolean function $read(w, m(p_i))$, while it has no ODE because it is a discrete place; meaning the read arc has no effect on the place itself.

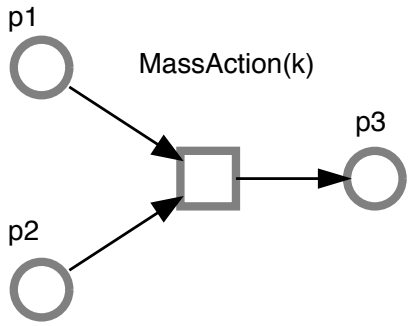
Figure 4.7 contains two different special arcs: read and inhibitor arc. The read arc plays the same role as in Figure 4.6, and the inhibitor arc makes its effect by the $inhibit(w, m(p_i))$ function. The weight of the former arc is 5 while the later one has weight of 2. In this case, the rate function of transition t is switched on when $m(p_4) \geq 5$ and $m(p_5) < 2$. The places p_4, p_5 have zero change rates (p_4 is a discrete place, hence it has no ODE).

Figure 4.8 illustrates the idea behind modifier arcs. The place p_4 is connected with transition t by a modifier arc. The modifier arc allows us to use the pre-place in the transition's rate function. Thus the current marking of the pre-place has an influence on the transition rate without being changed itself.

The functions $read(w, m(p_i))$ and $inhibit(w, m(p_i))$ define full activation and full inhabitation of the connected transition. If the modeller does not require such semantics of read and inhibitor arcs and prefers an explicit representation of the inhibition and activation via appropriate kinetic equations, then a modifier are can be used to model such cases.

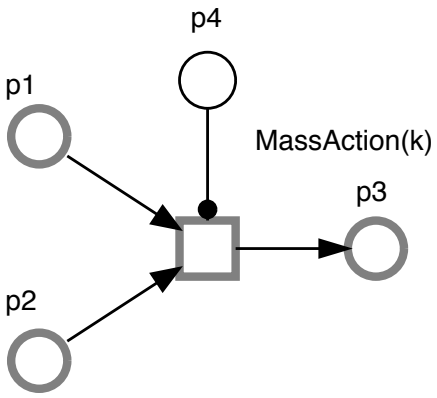
4.2.5 Marking-dependent Arc Weights

To support the special modelling requirements of some biological models (e.g., cell cycle model), arc weights are allowed to be defined by pre-place of a transition [Val78] or even a function which is defined in terms of the transition's pre-places [MTA+03].



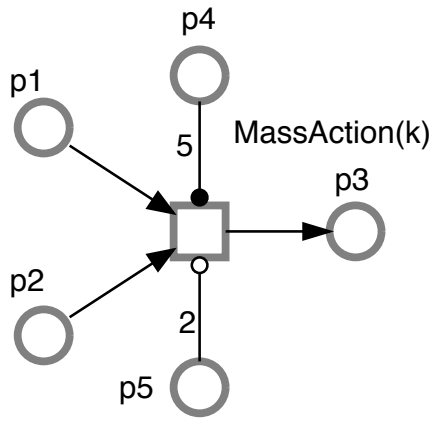
$$
\begin{aligned}
dm(p1)/dt &= -k \cdot m(p1) \cdot m(p2) \\
dm(p2)/dt &= -k \cdot m(p1) \cdot m(p2) \\
dm(p3)/dt &= \quad k \cdot m(p1) \cdot m(p2)
\end{aligned}
$$

Figure 4.5: Example of ODEs generation of $\mathcal{GHPN}_{bio}$ without extended arcs.



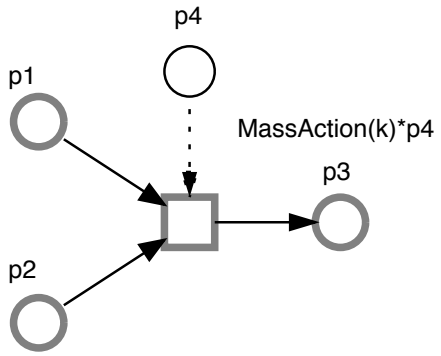
$$
\begin{aligned}
dm(p1)/dt &= -k \cdot m(p1) \cdot m(p2) \cdot m(p4) \cdot read(1, m(p4)) \\
dm(p2)/dt &= -k \cdot m(p1) \cdot m(p2) \cdot m(p4) \cdot read(1, m(p4)) \\
dm(p3)/dt &= \quad k \cdot m(p1) \cdot m(p2) \cdot m(p4) \cdot read(1, m(p4))
\end{aligned}
$$

Figure 4.6: Example of ODEs generation of $\mathcal{GHPN}_{bio}$ with a read arc.



$$
\begin{aligned}
dm(p1)/dt &= -k \cdot m(p1) \cdot m(p2) \cdot (m(p4))^5 \cdot read(5, m(p4)) \cdot inhibit(2, m(p5)) \\
dm(p2)/dt &= -k \cdot m(p1) \cdot m(p2) \cdot (m(p4))^5 \cdot read(5, m(p4)) \cdot inhibit(2, m(p5)) \\
dm(p3)/dt &= \quad k \cdot m(p1) \cdot m(p2) \cdot (m(p4))^5 \cdot read(5, m(p4)) \cdot inhibit(2, m(p5)) \\
dm(p5)/dt &= \quad 0
\end{aligned}
$$

Figure 4.7: Example of ODEs generation of $\mathcal{GHPN}_{bio}$ with read and inhibitor arcs.



$$
\begin{aligned}
dm(p1)/dt &= -k \cdot m(p1) \cdot m(p2) \cdot m(p4) \\
dm(p2)/dt &= -k \cdot m(p1) \cdot m(p2) \cdot m(p4) \\
dm(p3)/dt &= \quad k \cdot m(p1) \cdot m(p2) \cdot m(p4)
\end{aligned}
$$

Figure 4.8: Example of ODEs generation of $\mathcal{GHPN}_{bio}$ with a modifier arc.

Consider the following biological example. When a cell divides the mass between two daughter cells, each daughter takes approximately half of the mass. This situation cannot easily be modelled using standard Petri nets as shown in Figure 4.9a. In Figure 4.9b, using the marking-dependent arc weight; the ingoing arc of the transition t has a weight equal to the marking of the place p_1, while each of the two outgoing arcs has a weight equal to half of the marking of place p_1.

Motivated by the case study discussed in more details in Section 6.2, marking-dependent weights are introduced to the majority of arc types supported by $\mathcal{GHPN}_{bio}$ (standard, read, inhibitor, and equal arc). For more detail see Chapter 6.

The aforementioned definition of $\mathcal{GHPN}_{bio}$ does not account for such arc weights. However, few extensions are required on the syntactic level.

Definition 4.5 (Marking-dependent arc weight) *Let D_n and D_q be sets of functions defined as follows:*

$$D_n = \{d_n | d_n : \mathbb{N}_0^{|{}^\bullet t_j|} \to \mathbb{N}, t_j \in T\}, and$$
$$D_q = \{d_q | d_q : \mathbb{R}_0^{|{}^\bullet t_j|} \to \mathbb{Q}^+, t_j \in T\}$$

then the function F (compare Section 4.2.2) assigns a marking-dependent function to each arc depending on its type and is defined as follows:

$$F : \begin{cases} A_{cont} \to D_q, \\ A_{disc} \to D_n, \\ A_{read} \to D_q, \\ A_{inhibit} \to D_q, \\ A_{equal} \to D_n, \\ A_{reset} \to \{1\}, \\ A_{modifier} \to \{1\}. \end{cases}$$

This definition of arc weights permits the functions to be defined in terms of the marking of the destination transition's preplaces and therefore preserves the structure of the Petri net.

4.2.6 Conflict Resolution

A structural conflict [KBD+94, DA10] is given if a place has two post-transitions. A structural conflict turns into an effective one if the current marking enables both transitions, but only one of them can fire. To resolve the effective conflicts between transitions which belong to different types, we assume implicitly a priority level assigned to each transition type. The priority order from high to low is:

Figure 4.9: Marking-dependent weight illustrated by a simple biological example. (a) cell division cannot be modelled, (b) cell division can intuitively be modelled.

- immediate transitions,
- stochastic transitions,
- deterministically delayed (including scheduled) transitions, and
- continuous transitions.

For example, in the case of conflict between an immediate transition and a stochastic one, the immediate transition will fire first. However, if there is a conflict between continuous and stochastic transitions, stochastic transitions will be given a priority to fire. Nevertheless, a conflict might also exist between transitions of the same type.

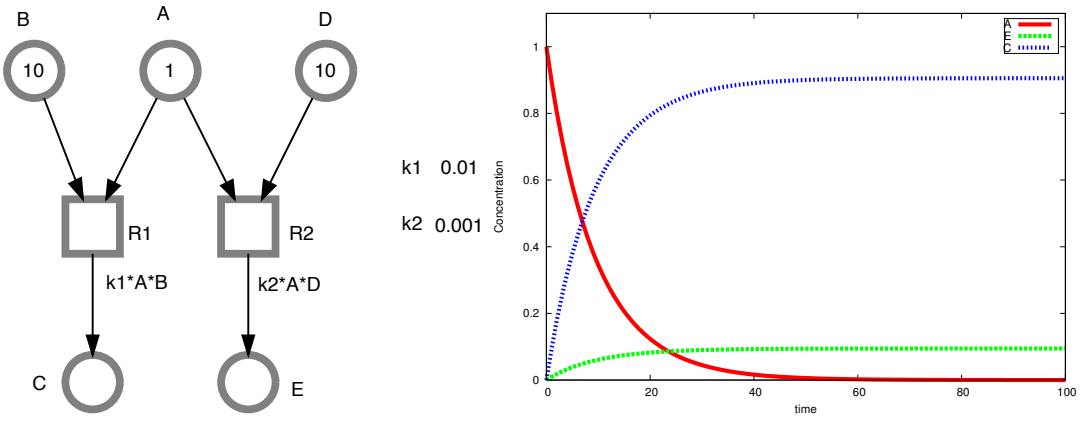
Conflicts between immediate transitions will be resolved by computing the relative firing frequencies of each enabled immediate transition. More precisely, if an immediate transition t_j is enabled in the current marking m, then it fires with the probability given by (4.5).

$$\frac{w(t_j)(m)}{\sum_{t_k \in T_{im} \wedge isEnabled(t_k,m)} w(t_k)(m)} \tag{4.5}$$

where $w(t_j)(m)$ is the weight assigned to an immediate transition t_j in the current marking m, and $isEnabled(t_k, m)$ is a function that checks if a transition t_k is enabled on the marking m.

Stochastic transitions will never be in a conflict with each other due to the exponential distribution of the random waiting time [KBD+94].

Conflicts between deterministically delayed transitions are solved by selecting one of them randomly. Indeed, they enjoy the original Petri net property of non-determinism.



(a) (b)

Figure 4.10: Conflict between continuous transitions. (a) Petri net representation, (b) time course result of species A. Structural conflict might exist, however there will be no effective conflicts.

Finally, structural conflicts [DA10] might exist between continuous transitions; however, effective conflicts will not occur. To clarify this point, consider the two simple reactions in (4.6) and their Petri net representation in Figure 4.10.

$$\begin{aligned} A + B &\xrightarrow{k_1} C \\ A + D &\xrightarrow{k_2} E \end{aligned} \tag{4.6}$$

According to the mass-action kinetic law, the rates of the two reactions are: $k1 \cdot A \cdot B$ and $k2 \cdot A \cdot D$, where $k1$ and $k2$ are the kinetic rate constants.

In this case, the reaction rates may vary with time; however, the same ODEs will be used to calculate the current rates as depicted in the simulation result in Figure 4.10. Other models similar to the this one, where a structural conflict exists while there is no effective conflict, can be found in [DK98, DA10].

Biological example To motivate the aforementioned discussion of the interconnection between discrete and continuous parts, consider the generalised hybrid Petri net in Figure 4.11 which is a direct translation of the reaction set in Table 4.1. It models the T7 phage viral kinetics using generalised hybrid Petri nets. Two different time scales can be distinguished in this model. One represents fast reactions and contains R_5 and R_6, and the other one comprises the slow reactions R_1, R_2, R_3, and R_4. Slow reactions are modelled using stochastic transitions to preserve the accuracy, while fast reactions

No.	Reaction	Propensity	Rate constants
R_1	$gen \rightarrow tem$	$c_1 \cdot gen$	$c_1 = 0.0025$
R_2	$tem \rightarrow \phi$	$c_2 \cdot tem$	$c_2 = 0.25$
R_3	$tem \rightarrow tem + gen$	$c_3 \cdot tem$	$c_3 = 1.0$
R_4	$gen + struct \rightarrow$ "$virus$"	$c_4 \cdot gen \cdot struct$	$c_4 = 7.5 \times 10E-6$
R_5	$tem \rightarrow tem + struct$	$c_5 \cdot tem$	$c_5 = 1000$
R_6	$struct \rightarrow \phi$	$c_6 \cdot struct$	$c_6 = 1.99$

Table 4.1: T7 phage viral kinetics reaction set [SYSY02]. The net representation is given in Figure 4.11. For detailed discussion of this model see Section 6.1

are modelled using continuous reactions to speed up the model simulation. Notice how the read arc that connects transition R_5 and place *tem* is used to link discrete and continuous parts. If there are tokens in place *tem*, the rate of reaction R_5 will be non-zero, and it will change depending on the number of tokens in that place. In contrast, if there is no token in place *tem*, the reaction rate of R_5 will be zero. The simulation of this model will be discussed later in Section 6.1 (page 116).

4.3 Simulation of GHPN

Having presented the modelling aspects of $\mathcal{GHPN}_{bio}$ in the previous section, we discuss here the approach which is used to simulate $\mathcal{GHPN}_{bio}$. The key simulation idea is to numerically solve the set of ODEs generated by the continuous transitions until a discrete event occurs. The event type is dispatched, and afterwards the continuous simulation is resumed. We start with discussing the simulation of statically partitioned $\mathcal{GHPN}_{bio}$, before presenting the dynamic partitioning of $\mathcal{GHPN}_{bio}$, which substantially simplifies the modelling of biochemical networks using $\mathcal{GHPN}_{bio}$.

4.3.1 Simulation of Statically Partitioned $\mathcal{GHPN}_{bio}$

In the following, we illustrate how $\mathcal{GHPN}_{bio}$ can be simulated using an extended version of the algorithms which were discussed in [HR02, ACT+05, GCPS06]. To map the mathematical notations introduced in Chapter 3 with the ones that are introduced in this chapter, let

$$a(t_j) = a_j(\mathbf{x}) = \begin{cases} v_j(\tau) & \text{if } t_j \in T_{cont} \\ g_j(\tau) & \text{if } t_j \in T_{stoch} \end{cases}$$

be the propensity (rate) of a continuous or a stochastic transition.

Algorithm 4.1 summarises the steps that are needed to simulate $\mathcal{GHPN}_{bio}$. Starting from an initial marking which corresponds to the initial state of a biochemical system,

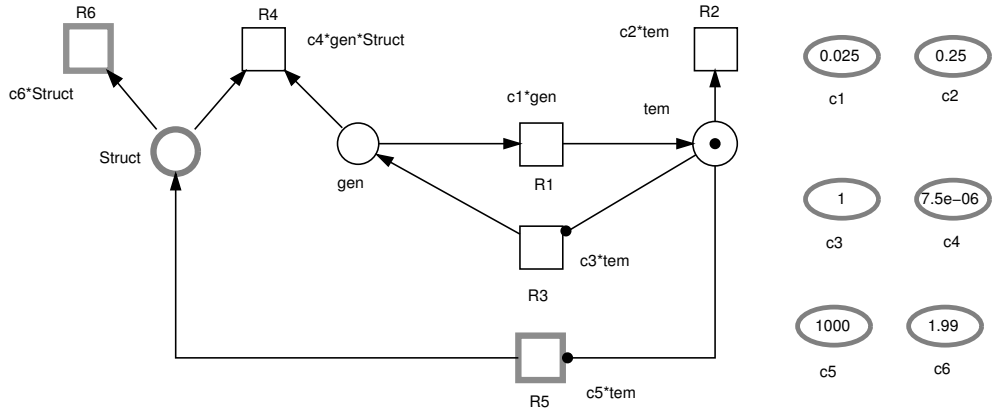


Figure 4.11: A $GHPN_{bio}$ representation of the T7 phage model; compare Table 4.1. Values in eclipses are read as parameters.

the algorithm computes state changes over time, which is represented by the current marking $m(\tau)$. Initially, the current marking is set to the initial marking, and the individual propensities $a(t_j)$ are calculated for both stochastic and continuous transitions (lines 3-4); afterwards, the cumulative propensity of stochastic transitions is computed in line 5. Deterministically delayed transitions do not have propensities, since they fire after a pre-defined time delay at absolute points of time. Note that in the algorithm we consider scheduled transitions as deterministically delayed transitions since they can be considered as a special case of them. Note that immediate transitions also do not have propensities associated with them (see Section 4.1.1).

If there is a non-stochastic transition in the model, then the algorithm determines the next stochastic transition to fire by integrating the set of ODEs as well as the cumulative propensities until (3.28) is satisfied. Please note, the ODEs can contain only (3.28) if there is no continuous transition.

The numerical integrator stops when an event E occurs. The event may be an enabling of an immediate or deterministically delayed transition, a deterministically delayed transition has finished its delay, a stochastic event occurred, or the end of simulation time has been reached. Then, the appropriate action will be taken.

Line 11 updates all of the transitions' propensities that share a pre-place with a continuous transition. The function $IsEnabled(t_j)$ checks for enabling of a transition t_j, while $Fire(t_j)$ fires an enabled transition. The details of these functions are easy to be implemented, therefore they are not further considered here.

$CheckImmediateTransitions()$ checks if there is any immediate transition enabled.

Algorithm 4.1: Simulating Statically Partitioned $GHPN_{bio}$

1: $\tau \leftarrow 0$;
2: $\xi \leftarrow exp(1)$ {Generate a random number exponentially distributed with a unit mean}
3: $m(\tau) \leftarrow m(0)$; {current marking=initial marking}
4: $\forall t_j \in T_{cont} \cup T_{stoch}$ calculate $a(t_j)$;
5: $a_0 \leftarrow \sum a(t_j), \forall t_j \in T_{stoch}$;
6: **while** $\tau < \tau_{end}$ **do**
7: **if** There are non-stochastic transitions **then**
8: Initialise the ODE solver by $m(\tau)$;
9: Simultaneously integrate the system of ODEs generated using (3.30) and $g(m(\tau))$ until an event E occurs;
10: $\tau \leftarrow$ the current integrator time;
11: Update$(a(t_i), a_0)$, $\forall t_i : {}^\bullet t_i \cap \{{}^\bullet t_j \cup t_j^\bullet\} \neq \phi, \forall t_j \in T_{cont}$;
12: **if** E is: $\exists t_j \in T_{im} \wedge$ IsEnabled(t_j) **then**
13: CheckImmediateTransitions();
14: **else if** E is: $\exists t_j \in T_{deter} \wedge$ IsEnabled(t_j) **then**
15: CheckDeterministicTransitions();
16: **else if** E is: $\exists t_j \in T_{deter} \wedge$ FireTime$(t_j) = \tau$ **then**
17: CheckDeterministicTransitions();
18: **else if** E is: $g(m(\tau)) - \xi \geq 0$ **then**
19: $g(m(\tau)) \leftarrow 0$;
20: $\xi \leftarrow exp(1)$;
21: $t_{chosen} \leftarrow$ a transition index i satisfying (3.29);
22: Fire(t_{chosen});
23: Update$(a(t_i))$, $\forall t_i : {}^\bullet t_i \cap \{t_{chosen}^\bullet \cup {}^\bullet t_{chosen}\} \neq \phi$
24: **else if** E is: $\tau \geq \tau_{end}$ **then**
25: terminate;
26: **end if**
27: **else**
28: $\tau \leftarrow \tau + \exp(a_0)$ {See (3.18).}
29: **if** $(\tau < \tau_{end}) \wedge (a_0 > 0)$ **then**
30: $t_{chosen} \leftarrow$ a transition index i satisfying (3.19);
31: Fire(t_{chosen});
32: Update$(a(t_i))$, $\forall t_i : {}^\bullet t_i \cap \{t_{chosen}^\bullet \cup {}^\bullet t_{chosen}\} \neq \phi$
33: **end if**
34: **end if**
35: **end while**

If such a transition is found, it will be fired. If there are several immediate transitions enabled, then the first one to fire is selected based on their weights as given by (4.5).

The purpose of $CheckDeterministicTransitions()$ is twofold. First, it checks if there are any enabled deterministically delayed transitions; if so, it puts them in the delay list along with their tentative time to fire. Secondly, if there are transitions in the delay list which have finished their delay, then it fires them.

Lines 19-23 and lines 30-32 perform the same task, but for different conditions. In the former case, a stochastic transition is selected to be fired when the ODE integrator determines that a stochastic event has occurred. The stochastic transition is selected based on equation (3.29). In the latter case, the model contains only stochastic transitions. Thus the next reaction time is computed based on equation (3.18), and the next transition to fire is selected based on (3.19).

When a transition fires, the propensity of this transition as well as the propensities of any other transitions that are affected by this firing are recomputed and the cumulative propensity is updated. The simulation ends when the current simulation time exceeds the simulation's end time which is specified by the user.

While this algorithm can simulate any $\mathcal{GHPN}_{bio}$, it requires the user to specify the partitioning in advance. Sometimes it is not easy to do the partitioning off-line. It is also possible that a good partitioning changes dynamically over time. Therefore, we present in the next section an algorithm which supports on-line partitioning. In some cases, the price of this dynamic partitioning is a higher computational overhead [Pah09].

4.3.2 Transition Partitioning

Static partitioning of Petri nets into stochastic and continuous net parts is not always appropriate. During the simulation, transitions' rates can drastically vary between low and high. Furthermore, off-line partitioning is not user friendly, since it is not easy for naive users to determine which transitions should be considered stochastically and which one continuously [Pah09]. The latter problem could be overcome by running stochastic simulation for only one or a few trajectories in order to determine the partitioning automatically [ACT+05]. Another solution is to use stochastic analysis techniques.

The partitioning of the reactions into slow and fast can be done through the use of two thresholds: λ for the transitions' rates and Λ for the places' marking [ACT+05, GCPS06].

However one important question remains: when do we need to consider repartitioning? One solution to this problem is to reconsider repartitioning after a specific time period (for example every one or two seconds). However this will not correctly solve the problem since there may be significant changes in some species' populations during this period. Moreover it results in useless computational overhead when there is no need to repartition. In our partitioning approach we solve this problem by specifying

two other thresholds: $a_{0_{max}}$, $a_{0_{min}}$.

Consider equation (3.18), which determines the next time point a stochastic event will occur. Larger values of a_0 will result in smaller time steps in the stochastic simulation. On the other hand, smaller values of a_0 will result in larger time steps. In fact this also affects equation (3.28) which determines when we switch from deterministic to stochastic simulation. The same arguments hold for (3.28). The main idea here is that we can control the speed and accuracy of hybrid simulation by specifying a lower and upper bound of a_0. Then the algorithm will realise that it needs to repartition the net when a_0 drops below $a_{0_{min}}$ or exceeds $a_{0_{max}}$.

Algorithm 4.2 summarises the steps which are needed to carry out on-line partitioning of the network. It considers repartitioning if equation (4.7), (4.8) or both are violated.

$$a_{0_{min}} \leq a_0 \leq a_{0_{max}} \tag{4.7}$$

$$m(p_i) \geq \Lambda, \forall p_i \in \{{}^\bullet t_j \cup t_j{}^\bullet\}\ \forall t_j \in T_{cont} \tag{4.8}$$

where Λ is a threshold for the marking of a place p_i.

An inappropriate choice of the thresholds can result in unsuitable partitioning which may turn out to be more computationally expensive than static partitioning.

The algorithm takes as input the stochastic and continuous transitions, a_{min}, a_{max} – the upper and lower bounds of the cumulative propensity, respectively, the transitions' rate threshold λ, and the places' marking threshold Λ. Note that the other transition types (the discrete transition) are not repartitioned. At the end of the partitioning, the algorithm returns T'_{stoch} and T'_{cont} as the new partitioning.

The idea of repartitioning is then easy. If one of the transitions violates the partitioning criterion, it will be added to the stochastic transitions, otherwise it will be added to the continuous one.

A transition type conversion from stochastic to continuous or vice versa might also involve a conversion of the corresponding transition rate functions. Such conversion is automatically done only if the user deploys a kinetic pattern transition rate function (e.g., MassAction). In this case, the simulation algorithm automatically uses the appropriate representation of this pattern (continuous or stochastic) based on the current transition type.

This algorithmic idea, together with the one which is presented in Section 4.3.1, provides a dynamic simulation of the $\mathcal{GHPN}_{bio}$ which have been introduced in this thesis.

Algorithm 4.2: Dynamic Partitioning of $GHPN_{bio}$

1 Input: a_{min}, a_{max}, λ, Λ, T_{stoch}, T_{cont}
2 Output: T'_{stoch} , T'_{cont}

$T'_{stoch} \leftarrow \phi$, $T'_{cont} \leftarrow \phi$
if $a_0 < a_{min} \vee a_0 > a_{max} \vee \forall p_i \in {}^\bullet T_{cont} \cup T_{cont}^\bullet \; \exists m(p_i) < \Lambda$ **then**
  for all $t_j \in T_{stoch} \cup T_{cont}$ **do**
    if $a(t_j) > \lambda \wedge \forall p_i \in \{{}^\bullet t_j \cup t_j^\bullet\}, m(p_i) \geq \Lambda$ **then**
      if $t_j \in T_{stoch}$ **then**
        $a_0 \leftarrow a_0 - a(t)$
        $T'_{cont} \leftarrow T'_{cont} \cup \{t_j\}$
        $T_{stoch} \leftarrow T_{stoch} - \{t_j\}$
      end if
    else
      if $t_j \in T_{cont}$ **then**
        $a_0 \leftarrow a_0 + a(t_j)$
        $T_{cont} \leftarrow T_{cont} - \{t_j\}$
        $T'_{stoch} \leftarrow T'_{stoch} \cup \{t_j\}$
      end if
    end if
  end for
  return T'_{stoch}, T'_{cont}
else
  return T_{stoch}, T_{cont} {No partitioning is needed}
end if

4.4 Implementation Aspects

The presented Petri net class and its simulation algorithms are implemented in Snoopy [HHL+12]. Snoopy is available free of charge for non-commercial use and it can be downloaded from [Sno12]. It is platform-independent and runs under Mac OS X, Windows and Linux (selected distributions). We implemented Gillespie's direct method [Gil76] to simulate stochastic transitions, while SUNDIALS CVODE [HBG+05] and other single step solvers are used to integrate the ODEs induced by the continuous transitions (see below). Snoopy supports also a dedicated net class to simulate Petri nets which contain only continuous elements (Continuous Petri Nets) and provides 14 different ODEs integrators. The addition of further stochastic simulators is easy due to the generic design of Snoopy (see future work). Snoopy supports also many other useful modelling features like hierarchy and logical nodes which are specifically helpful when considering modelling and simulation of larger biochemical networks. Furthermore, models devel-

oped with Snoopy can in principle be exported to a variety of analysis tools. However, there is no comparable tool for our $\mathcal{GHPN}_{bio}$ models. Therefore, a $\mathcal{GHPN}_{bio}$ model could be exported with some loss of information only. Nevertheless, $\mathcal{GHPN}_{bio}$ models can be exported to other Petri net classes supported by Snoopy whereby any loss of information is announced. This means that $\mathcal{GHPN}_{bio}$ models can be indirectly exported to SBML, ODEs in LaTeX notation, etc.

Finally, using $\mathcal{GHPN}_{bio}$ the same model can be simulated continuously or stochastically independently of its original modelling method, thanks to dynamic partitioning.

In the following subsections, we discuss briefly some important implementation issues of the simulation algorithm for $\mathcal{GHPN}_{bio}$.

4.4.1 Stochastic Simulation Algorithm

Obviously, the hybrid simulation algorithm presented in Section 4.3 integrates a stochastic simulation algorithm (SSA) as well as an ODE solver. Theoretically, all of the stochastic algorithms discussed in Chapter 3 can be used to accomplish this task. However, not all of them provide an acceptable performance. For instance, although the next reaction method outperforms the direct method in a stand alone implementation of the SSA algorithm, it is empirically inappropriate for the hybrid case. The special data structure required by the next reaction method renders the hybrid simulation algorithm more complex. Similarly, the first reaction method requires the generation of random numbers equal to the number of stochastically simulated transitions. Thus, this method might be useful if the number of stochastic transitions is limited. Finally, the direct method seems to be the best one among its peer to fit the purpose of hybrid simulation.

4.4.2 Selecting an Appropriate ODE Solver

Using an appropriate ODE solver is crucial to obtain an acceptable performance and accuracy index of the problem under study. For instance, using unsuitable ODE solvers might result in a simulation engine that is slower than a pure stochastic simulation.

In Section 3.3.1 the different types of ODE solvers have been discussed. What is important to our discussion of ODE solvers, in the context of hybrid simulation, is the comparison between single- and multi-step methods.

In the case where the ODE system does not contain discontinuities (i.e., the solution does not jump from one branch to the other), multi-step methods do not require additional work to compute the history points, since these solutions already exist. However, if such discontinuities exist, the memory feature (i.e., bookkeeping and reusing history solution points) is lost and the solver becomes slow. Unfortunately, many systems of interest experience significant discontinuities during transients [MP02a]. Particularly in hybrid simulation algorithms, discontinuities occur frequently due to the switching

between stochastic and continuous simulation. This makes multistep solver very slow for this purpose.

To overcome this problem, we provide different ODE solvers such that the user can select among them. If the model contains many discontinuities (i.e., many stochastic transitions or few stochastic transitions with high rates), the explicit or implicit Rung-Kutta solver will be the best choice. Contrary, if the problem under study does not exhibit any discontinuities or contains only few of them, Adams or BDF [HBG+05] solvers will be the appropriate choice to simulate the model. We reconsider later this issue after the presentation of some case studies in Chapter 6.

4.4.3 Detecting Discrete Events

The hybrid simulation algorithm requires switching between the continuous and the discrete regimes whenever a discrete event occurs. Unfortunately, such events can not be located only using the current simulation time. For instance, in order to determine when (3.28) is satisfied, ODE solver needs to repeatedly check it during the numerical integration. Such a feature is called event detection or root finding [MP02a]. As an example of the root finding problem, consider a function of the form $f(m)$. Finding the root of this function means to find the value of m satisfying $f(m)=0$. However, we would like to do that simultaneously while integrating the set of ODEs. In this context, the time where the root is found is also required (i.e., the location of the root). Few ODE solvers provide such an option. In our simulation problem, we have two different root functions: the enabling of immediate or deterministic transitions and the detection of the time point where (3.28) is satisfied. Nevertheless, ODE solvers accomplish this job by monitoring the signs of the event functions. Therefore, the event location can be missed if the event function has an even number of roots. Fortunately, our two event function types have an odd number of events. They are negative before the occurrence of the root and positive thereafter.

4.5 SPN, CPN and GHPN: the Big Picture

Gilbert et al. propose in [GHL07] a unified framework of modelling biochemical reaction networks. The framework integrates qualitative, stochastic and continuous approaches to simulate biological networks. This chapter extends this framework to include the hybrid approach as combined approximation of stochastic and continuous methods.

Figure 4.12 graphically illustrates the extended framework. Qualitative Petri nets ($\mathcal{QPN}$) are an abstract representation of reaction networks. They do not contain any quantitative information. Although this view of the representation and simulation of biochemical reactions does not enable the prediction of species' populations over time, the rich qualitative analysis techniques, available for this category of Petri nets, permit the analysis of the system behaviour under any time constraints.

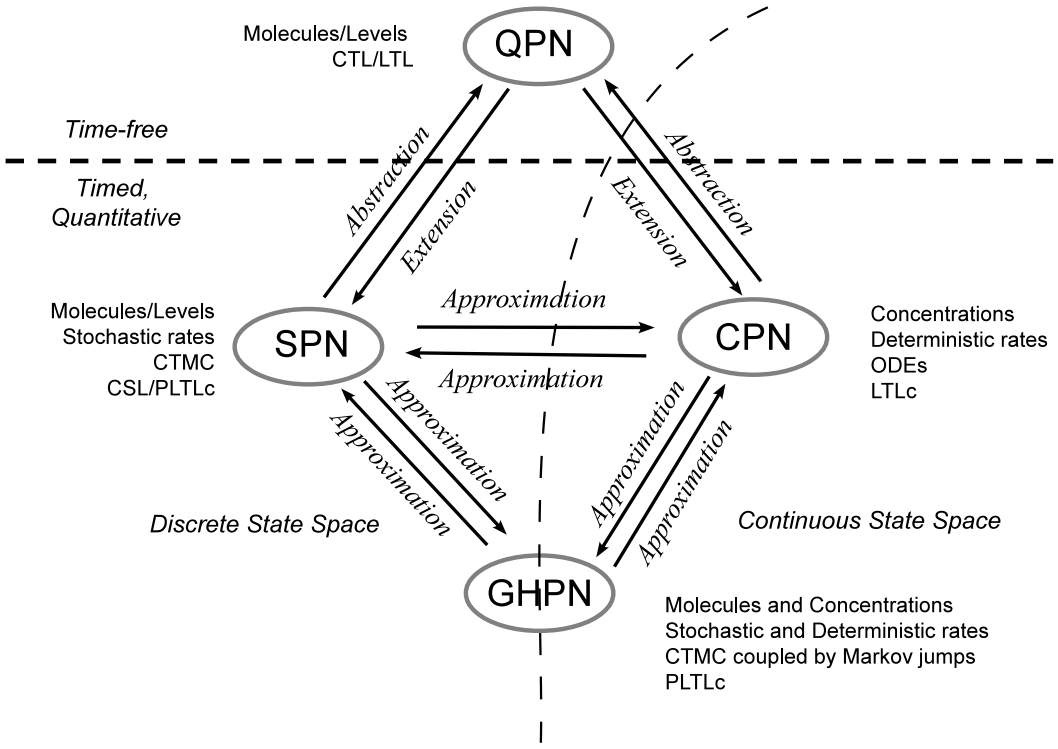


Figure 4.12: Relationship between $\mathcal{SPN}$, $\mathcal{CPN}$, and $\mathcal{GHPN}_{bio}$. An extension of the conceptual framework in [GHL07].

Contrary, timed information can be added to the qualitative model using Stochastic Petri Nets ($\mathcal{SPN}$) (see Chapter 3), while preserving the discrete state descriptions. The simulation results produced using $\mathcal{SPN}$ can be considered as the most accurate ones among the available quantitative Petri net classes, since they execute one reaction per simulation step. Here we mean by accuracy the ability of a simulator to reproduce the biological model behaviour. The $\mathcal{QPN}$ model can be converted to an $\mathcal{SPN}$ one by extending the transitions with probabilistically distributed firing rates (waiting times). Moreover, $\mathcal{SPN}$ can be converted back into $\mathcal{QPN}$ by abstraction (i.e., removing the time information).

Similarly, $\mathcal{CPN}$ is an extension of the $\mathcal{QPN}$, whereby the time as well as the markings are continuous and deterministic quantities. The transformation between $\mathcal{CPN}$ and $\mathcal{QPN}$ are similar to the one between $\mathcal{SPN}$ and $\mathcal{QPN}$.

$\mathcal{SPN}$ and $\mathcal{CPN}$ can be approximated with each other. The stochastic firing rate of $\mathcal{SPN}$ can be approximated by a deterministic firing, then the discrete tokens are replaced by continuous concentrations. This direction of approximation might involve

loss of information, particularly, if the number of tokens are small. However, it will result in a more efficient simulation.

Obviously, the approximation between $\mathcal{SPN}$ and $\mathcal{CPN}$ is extreme. We have either to sacrifice efficiency to retain the accuracy or sacrifice accuracy to run the simulation in a reasonable time. Therefore, there is a need to have a tool with an intermediate approximation between $\mathcal{SPN}$ and $\mathcal{CPN}$. $\mathcal{GHPN}_{bio}$ is such a tool. The level of approximation can easily be controlled using $\mathcal{GHPN}_{bio}$ by controlling the number of transitions and places that belong to each category. A $\mathcal{GHPN}_{bio}$ model will become equivalent to $\mathcal{SPN}$ if all model components are discrete. On the other hand, it will be equivalent to a $\mathcal{CPN}$ model if all the elements are continuous.

4.6 Comparison with Other Hybrid Petri Net Tools

In this section we discuss our contributions in comparison to other hybrid Petri nets classes. Here, we select four HPNs from the literature which support similar features as $\mathcal{GHPN}_{bio}$. These HPNs are: Fluid Stochastic Petri Nets (FSPN), Hybrid Functional Petri Nets with extensions (HFPNe), First Order Hybrid Petri Nets (FOHPN), and Differential Petri nets (DPN). For a short discussion of this classes see Chapter 3.

Table 4.2 summarises the features of each net class. From this table, we can notice that $\mathcal{GHPN}_{bio}$ support important features which do not exist in the other classes.

For example, although almost all these classes provide stochastic and continuous transitions, the simulation results are not accurate if the model contains both of them. In other words, they do not support the full interplay (as shown in Table 4.2) between stochastic and continuous parts. $\mathcal{GHPN}_{bio}$ do that by considering time-varying stochastic rates, while solving the resulting ODEs generated by the continuous transitions.

4.7 Examples

In this section, we present two simple examples to illustrate the hybrid modelling and simulation of $\mathcal{GHPN}_{bio}$ to keep this chapter self-contained. The two examples are: break-repair, and the Goutsias model. With the first example we aim to demonstrate the modelling power of generalised hybrid Petri nets compared to fluid stochastic Petri nets (FSPN). In the second one, we compare between the continuous, stochastic, and hybrid results when the set of reactions are separable into fast and slow ones. As we will see, stochasticity can be preserved when $\mathcal{GHPN}_{bio}$ is used.

Table 4.2: Comparison between selected HPN classes

	FSPN	**HFPNe**	**FOHPN**	**DPN**	**GHPN**
Application Area	Performance optimisation	Systems Biology	Manufacturing Systems	Performance optimisation	Systems Biology
Continuous Transitions		X	X	X	X
Stochastic Transitions	X		X	X	X
Full Stochastic - Continuous Interplay					X
Deterministic Transitions		X	X	X	X
Scheduled Transitions					X
Immediate Transitions	X	X			X
Standard Arcs	X	X	X	X	X
Read Arcs		X			X
Inhibitor Arcs		X			X
Modifier Arcs					X
Reset Arcs					X
Marking-dependent arcs		X			X
Software Tool		Cell Illustrator			Snoopy

4.7.1 Break-Repair Model

To emphasis the modelling power of $\mathcal{GHPN}_{bio}$, we compare the modelling of a break-repair system using $\mathcal{GHPN}_{bio}$ and FSPN which supports the interplay between stochastic and continuous parts as we do. Although this case study does not have a biological context, it reveals some important $\mathcal{GHPN}_{bio}$ features. The example models a system of N statistically identical and independent components. Each of them can undergo a failure and a repair process.

The system was originally modelled in [TK93] as FSPN as given in Figure 4.13a. The amount of work currently in the system is approximated by the continuous place *load*, as it might accumulate a large number of tokens. The number of currently operating machines is represented by the number of tokens in p_1 and they can undergo a failure with rate λ. This failure process is modelled using the stochastic transition *break_down*. The number of non-working machines is modelled using the number of tokens in p_2; they can be repaired with rate μ which is achieved by the firing of the transition *repair*. Work arrives continuously at a constant rate r when the stochastic transition *incoming_work* is enabled (hence not when it fires, according to the rules of FSPN). Similarly, the work is dispatched continuously with a constant rate d per machine when the stochastic transition *break_down* is enabled. Thus stochastic transitions perform two functions: when they are enabled, they continuously consume (produce) flow with a rate specified by the arc weights, and when they fire, they add (consume) tokens to (from) a post-place (pre-place). Moreover, the continuous flow rates are assigned to the arc weights. Such semantics is graphically unclear and inconsistent with the usual Petri nets semantics as it was asserted in [HK99].

Figure 4.13b represents the modelling of the same system using $\mathcal{GHPN}_{bio}$. The process of incoming and dispatching work is modelled using the continuous transitions, *incoming_work* and *outgoing_work*, respectively. The latter transition is controlled by the number of functioning machines in p_1 using a read arc. Figure 4.14 illustrates the simulation result of the $\mathcal{GHPN}_{bio}$ model in Figure 4.13b.

The continuous simulation results in Figure 4.14 suggest that the system enters a steady state with these settings, however stochastic and hybrid simulation results reveal that it does not enter a steady state in the average behaviour (10^4 runs). Such different interpretations come from the discrete firing of stochastic transitions in the stochastic and hybrid simulations that take into consideration the perturbation of small numbers of tokens in the discrete places p_1 and p_2.

4.7.2 Goutsias Model

This model has been used by Goutsias in [Gou05] as an example for systems that can be effectively partitioned into two distinct subsystems, one that comprises slow reactions and one that comprises fast reactions. It has been studied in [WGMH10]

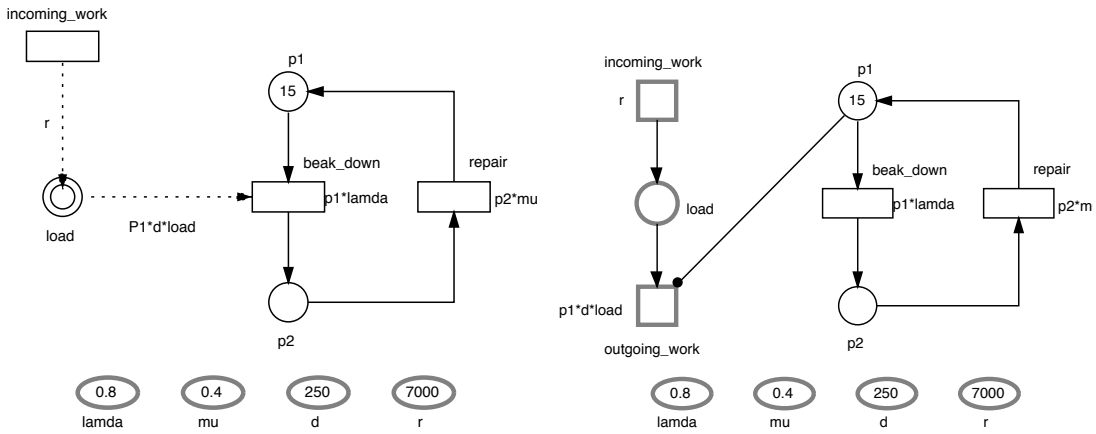


Figure 4.13: Generalised hybrid Petri net of the break-repair model: (a) FSPN representation, (b) $\mathcal{GHPN}_{bio}$ representation. Compared to $GHPN_{bio}$, FSPN notations are unclear and inconsistent with standard Petri net conventions. Numbers in ovals are read as parameters.

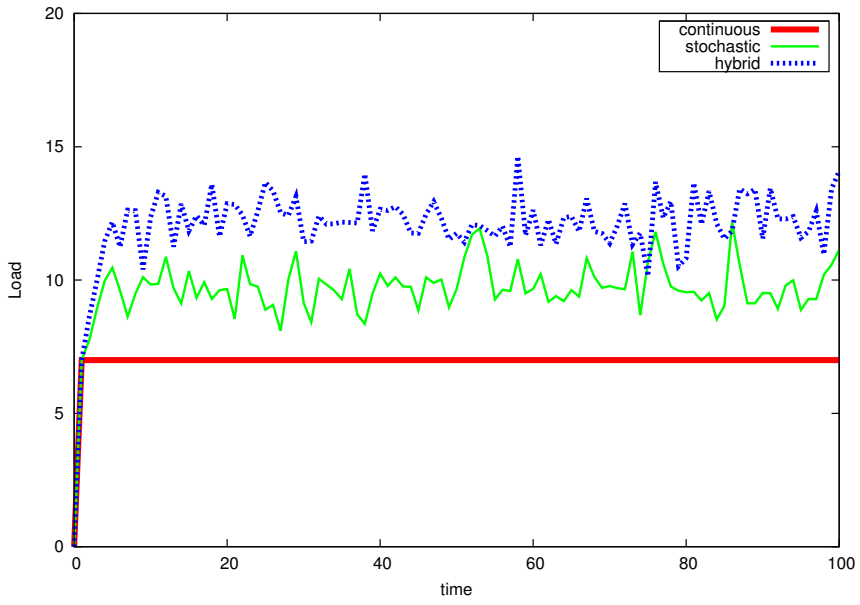


Figure 4.14: Simulation results of the break-repair model produced with Snoopy's simulation engine.

No.	Reaction	Propensity	Rate constants
R_1	$RNA \rightarrow RNA + M$	$k_1 \cdot RNA$	$k_1 = 0.043$
R_2	$M \rightarrow \phi$	$k_2 \cdot M$	$k_2 = 0.0007$
R_3	$DNA\_D \rightarrow RNA + DNA\_D$	$k_3 \cdot DNA\_D$	$k_3 = 71.5$
R_4	$RNA \rightarrow \phi$	$k_4 \cdot RNA$	$k_4 = 3.9 \times 10E - 6$
R_5	$DNA + D \rightarrow DNA\_D$	$k_5 \cdot DNA \cdot D$	$k_5 = 0.02$
R_6	$DNA\_D \rightarrow DNA + D$	$k_6 \cdot DNA\_D$	$k_6 = 0.48$
R_7	$DNA\_D + D \rightarrow DNA\_2D$	$k_7 \cdot D \cdot DNA\_D$	$k_7 = 0.0002$
R_8	$DNA\_2D \rightarrow DNA\_D + D$	$k_8 \cdot DNA\_2D$	$k_8 = 9 \times 10E - 12$
R_9	$M + M \rightarrow D$	$k_9 \cdot M \cdot M$	$k_9 = 0.08$
R_{10}	$D \rightarrow M + M$	$k_{10} \cdot D$	$k_{10} = 0.5$

Table 4.3: Goutsias model reaction set

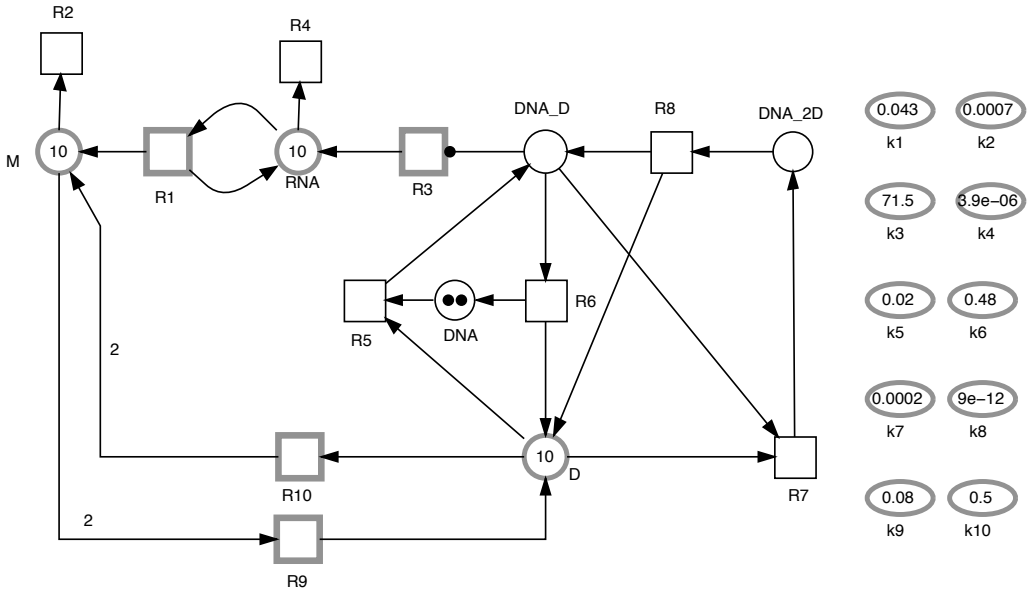


Figure 4.15: A $GHPN_{bio}$ representation of the Goutsias model. Transitions with high rates are represented as continuous transitions, while transitions with low rates are represented as stochastic transitions. Places are classified as continuous or discrete so that they satisfy the connection rules.

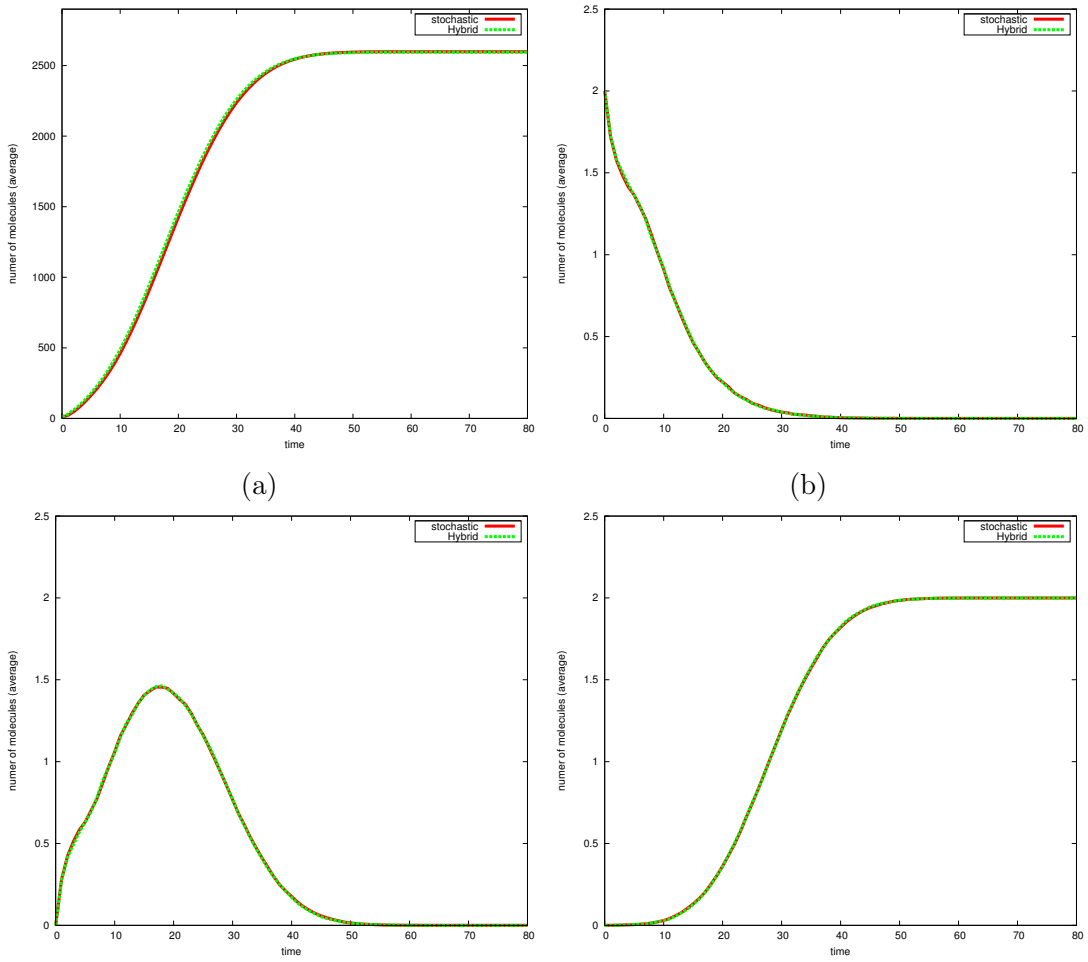


Figure 4.16: Stochastic, and hybrid simulation results of Goutsias model: (a) RNA, (b) DNA, (c) DNA_D, and (d) DNA_2D

and [HMMW10] as example for hybrid numerical solutions of the chemical master equation. We use the same reactions which have been originally proposed by [Gou05], and the more challenging parameters which have been used in [HMMW10]. The reactions set is given in Table 4.3.

Figure 4.15 is a hybrid Petri net representation of Goutsias' model. The partitioning of transitions and places into discrete and continuous ones is based on running one trajectory of a fully stochastic simulation. R_1, R_3, R_9, and R_{10} are reactions with high rates compared to the other reactions. Thus this set of reactions is represented

by continuous transitions which in turn are simulated by ODEs integrator. Note that places are partitioned into continuous and discrete ones according to the type of pre- and post-transitions. Places are considered as discrete ones if the adjacent arcs do not preclude this interpretation. Figure 4.16 is a time course result of the places *RNA, DNA, DNA.D*, and *DNA.2D*. The result shows that the hybrid simulation result coincides with the stochastic one for the four species. This is because the reactions in Figure 4.15 are completely separable into stochastic and continuous ones. Therefore, stochastically simulated reactions are executed as in the pure Monte Carlo simulation.

4.8 Conclusions

In this chapter, we have presented a new class of Petri nets which combines Extended stochastic Petri nets ($\mathcal{XSPN}$) and continuous Petri nets into one net class, the Generalised Hybrid Petri Nets. The introduced net class has several functionalities which help biologists to model and simulate their biochemical networks through an easy-to-use visual language. $\mathcal{GHPN}_{bio}$ models can be simulated by off-line partitioning or by on-line partitioning.

The two examples discussed in this chapter are simple and intended to explain some of the elements of $\mathcal{GHPN}_{bio}$. In Chapter 6, three other case studies are presented. Moreover, the performance of the $\mathcal{GHPN}_{bio}$ simulation engine will be discussed there in terms of runtime behaviour.

5 A Computational Steering Framework for Collaborative, Distributed, and Interactive Simulation of Biochemical Networks

5.1 Introduction

Although there are many problem solving environments in the context of computational modelling, which employ the computational steering technique, there is no dedicated computational steering tool for the scenario of kinetic modelling of biochemical networks. Early applications of computational steering were focused on classical computational problems, e.g., Computational fluid Dynamics, for examples see [JPH+99, Ani02, ABF+10]. However, few researches adapt general purpose computational steering environments for applications in systems biology, e.g., in [CFH+05, LGS+07]. Moreover, software tools that make use of Petri nets as a modelling language focus mainly on providing the user with a convenient graphical environment to draw the Petri net model, while they pay little attention to simulate the model conveniently. However, the quantitative simulation of Petri nets is not less important than the graphical representation of the reaction networks. It is an important step towards the understanding of the model under study.

In addition to the classical advantages of computational steering technique which have been discussed in Chapters 1 and 2, it has interesting potential outcomes when it is applied to systems biology problems.

As the main concern of systems biologists is to understand biological models at the system level, computational steering can increase such understanding by helping them to set-up "if-conditions" during the conduction of dry-lab experiments. Additionally, computational steering helps systems biologists to speed up the discovery process. Finally, computational steering could provide systems biologists with a comfortable and flexible simulation environment.

In this chapter, the second contribution of this thesis is discussed. Here, we propose, design and implement a computational steering and Petri nets framework [HH12b]. The framework combines computational steering and Petri nets to permit users to interact with the quantitative analysis of Petri net models. 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 extendibility to include further simulators provided by the users. Besides, the framework supports coloured and uncoloured Petri nets (see Chapter 3).

This chapter is organised as follows: first, we outline the general requirements of a computational steering environment which is tailored to the scenario of kinetic modelling of biochemical networks. Afterwards, we present our computational steering framework by discussing its interdependent individual components. In Section 5.5, we discuss in more details the integration of the simulation algorithms and computational steering. After that we discuss some implementation issues which face the simulation of multi-scale models using computational steering. A comparison between our framework and other software frameworks, which are used in similar application areas, is provided in Section 5.7. Finally, we sum up by conclusions and closing remarks.

5.2 Requirements and Characteristics

To be used by systems biologists, a computational steering based kinetic modelling software should fulfil some requirements and imperative features to facilitate the user's job. In this section, we will quickly pinpoint these requirements in the context of our proposed framework. Some of the significant requirements are: user friendliness, the ability to be executed in a simple or over distributed autonomous computational entities, extendibility and versatility, portable implementation, intuitive representation of the reaction networks, low latency, and collaborative and interactive simulation.

User friendliness is a crucial factor for users. For a computational steering technique to be successfully used by systems biologists, many technical issues should be hidden from the user (e.g., multi-threading, socket communication, and portability issues). Furthermore, no technical assistant should be needed by naive users to set up and run the resulting software in their own computers and share their models and experiences with other users. Additionally, the software application needs to be deployed with minimal, yet very rich set of simulators that could be used without additional external simulation code. However, in the case of existent legacy code, a facility should be provided to include it into the provided biochemical kinetic modelling software.

Lightweight communication and steering is another mandatory factor of applying computational steering to biochemical kinetic modelling applications. Since the final goal is to decrease the overall experiment duration, the communication between the user interface clients and the simulator should not extensively affect the simulation efficiency. One way to accomplish this goal is to send the user the results only upon request. Moreover, the simulator should not wait for the user input, instead the values steered by the user could be scheduled for the simulator to be processed at some

appropriate time points. For more details see Section 5.5.

Nevertheless, the resulting framework should be able to run on a single computer or several distributed computers. As a typical scenario, the steering server could be run on a powerful computer and therefore benefits from its huge computing resources, while the graphical user interface runs at the user's personal computer or even at a mobile device.

Extendibility is another important property of a versatile computational steering based kinetic modelling software. Advanced users need to easily integrate their own simulation code with an interactive framework. The instrumentation process – the process by which the simulation code is injected by a set of API calls – should also be kept user friendly and it should not require previous knowledge of socket communication or other technical terms. The user wants only to call some tiny APIs to carry out all of the steering and monitoring tasks. Moreover, it would be helpful if the graphical user interface can be rewritten by means of this APIs to meet the special user requirements.

Portability is another important issue. The (probably separable) components should have the ability to run on different computer hardware a long with different operating systems. The actual portability and synchronisation issues should completely be hidden from the user. Our implementation fulfils all of of these requirements. Moreover, It can run on Windows, Mac OS and Linux. Section 5.6 gives more detail about the different implementation issues.

Finally, using such organisation of components, users can collaborate together in solving a single problem and share their results upon their needs. Users can navigate from one model to another and get insights about the latest progress of the current problem under study. Manual intervention by the user with the simulators is important too. The interactive intervention enables users to correct simulation parameters without interrupting the simulator.

5.3 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.

5.3.1 Overview

Figure 5.1 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 (APIs), and the internal and external simulators. These

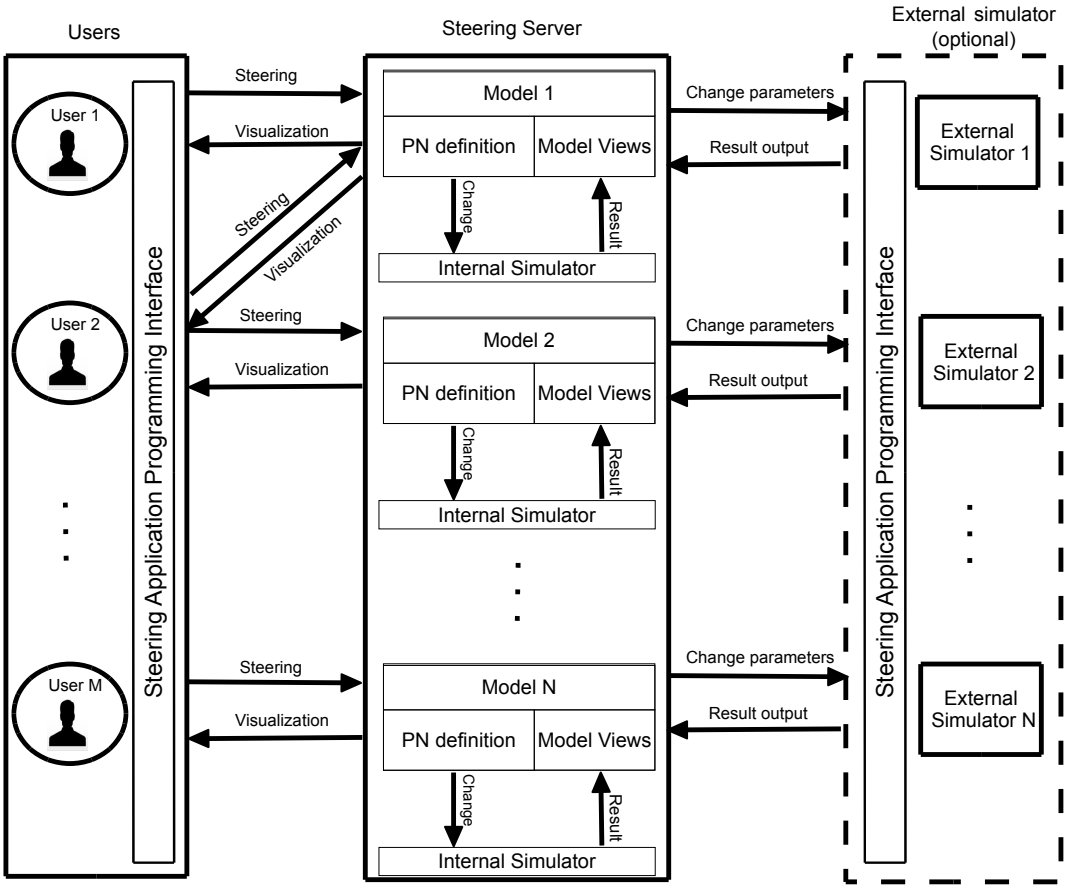


Figure 5.1: Petri nets and computational steering framework. The framework consists of four components: steering server, steering graphical user interface (GUI), steering application programming interface (Steering API), and simulators (internal and external). The flow of information goes in two opposite 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 (See Section 5.3.4).

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 early 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 5.3.2 gives more information about the operations and functionalities of the steering server.

The steering graphical user interface is the 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 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 these 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.

5.3.2 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-user 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. Indeed, computational steering could promote knowledge sharing between users of different backgrounds [KGM11]. Furthermore, multi-model and multi-threaded features coupled by multi-simulator 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 results, 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 [HGD08]. 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 running simultaneously on the server.

Furthermore, different users can dynamically 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 (see Section 5.6.1). 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 that 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 (i.e., collaborating users). The steering commands can be: altering model parameters, altering place marking, restarting, pausing, stopping the simulator, etc. The reason behind such organisation is that we cannot 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 are 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 afterwards other users are informed of the decision. For more details about including computational steering in a specific simulation algorithm, see Section 5.5.

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 the 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 a mediator role between the user interface and the external data source.

5.3.3 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 starts.

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 (e.g., reaction rate constants) 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 (e.g., start and end interval of the simulation). Figure 5.2 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 join the simulation remotely and change model parameters and the current marking. Figure 5.3 illustrates graphically the application scenario of Snoopy's computational steering framework.

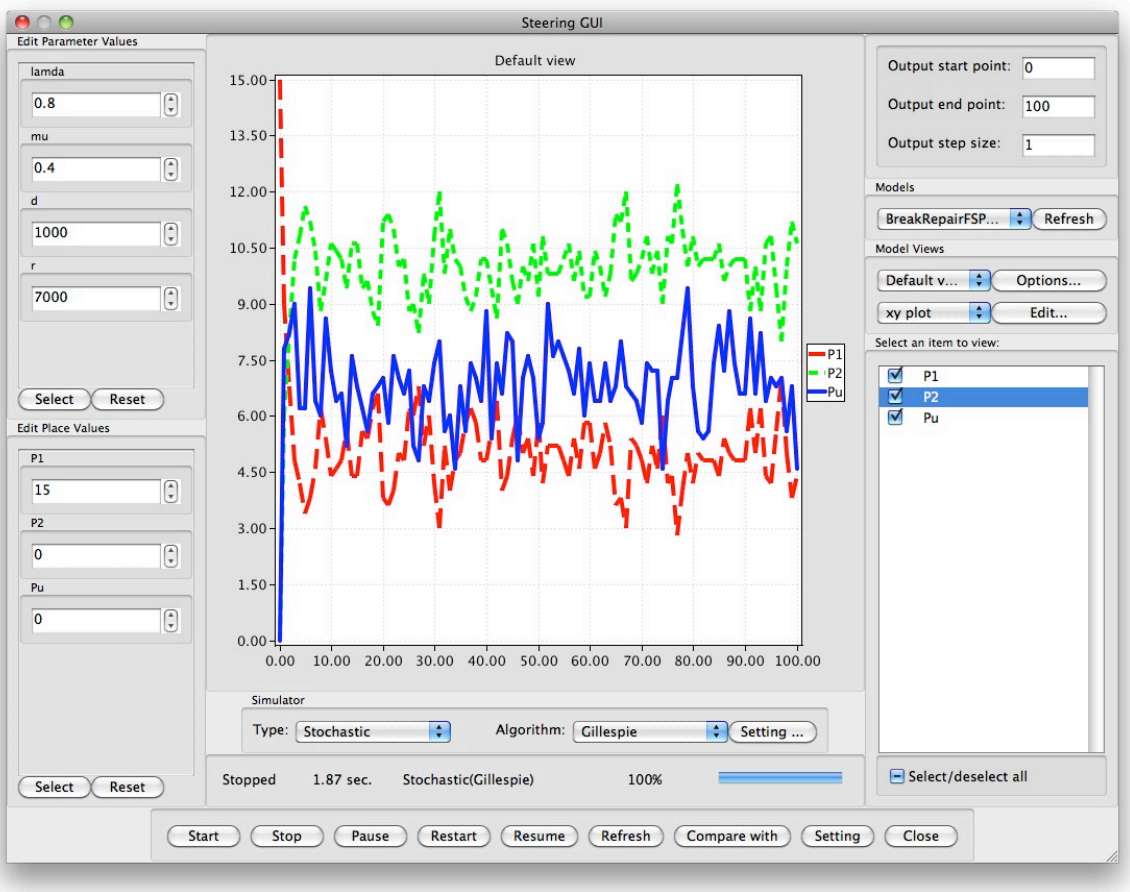


Figure 5.2: Snoopy's steering GUI: steering panel (left), output panel (middle), control panel (bottom) and manipulation panel (right). The user can select a subset of the model parameters to change their values during the simulation. The output can be viewed as table, xy plot, or histogram plot. The model results could be exported in csv or image format.

5.3.4 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

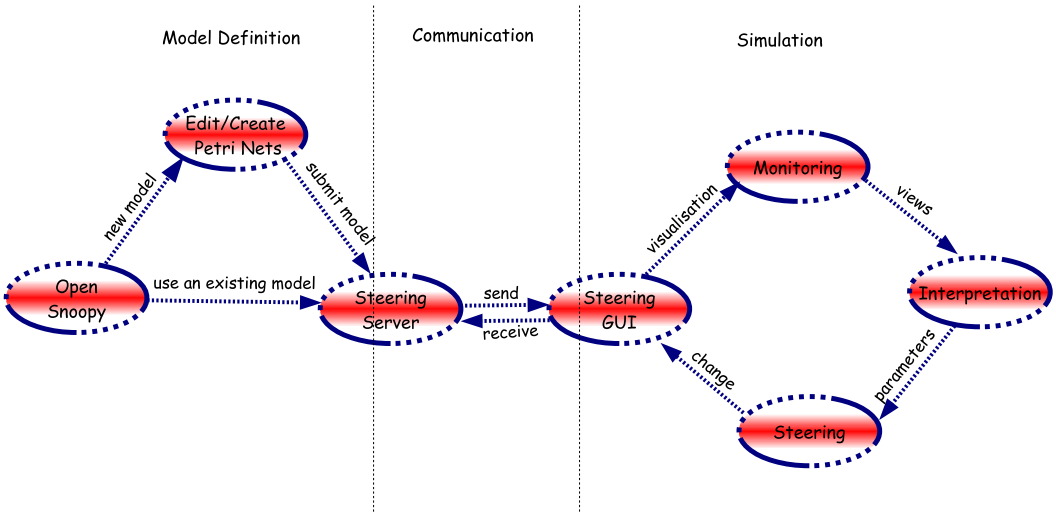


Figure 5.3: Graphical illustration of a typical application scenario of Snoopy's steering framework. The user has two options at the beginning: either reading one of the models already existing in the server or submitting a new one. In the latter case the Petri net model can be created using Snoopy Petri net editing tools. Afterwards, Snoopy's steering GUI can be used to perform the monitoring and steering.

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 5.4 illustrates the different classes of our implementation of the steering API library.

While our framework comes with a set of full-fledged simulators (see Section 5.3.5), 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.

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 which is illustrated in Figure 5.2; instead, they could implement their own dialogues and use their favourite visualisation libraries. The availability 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 compo-

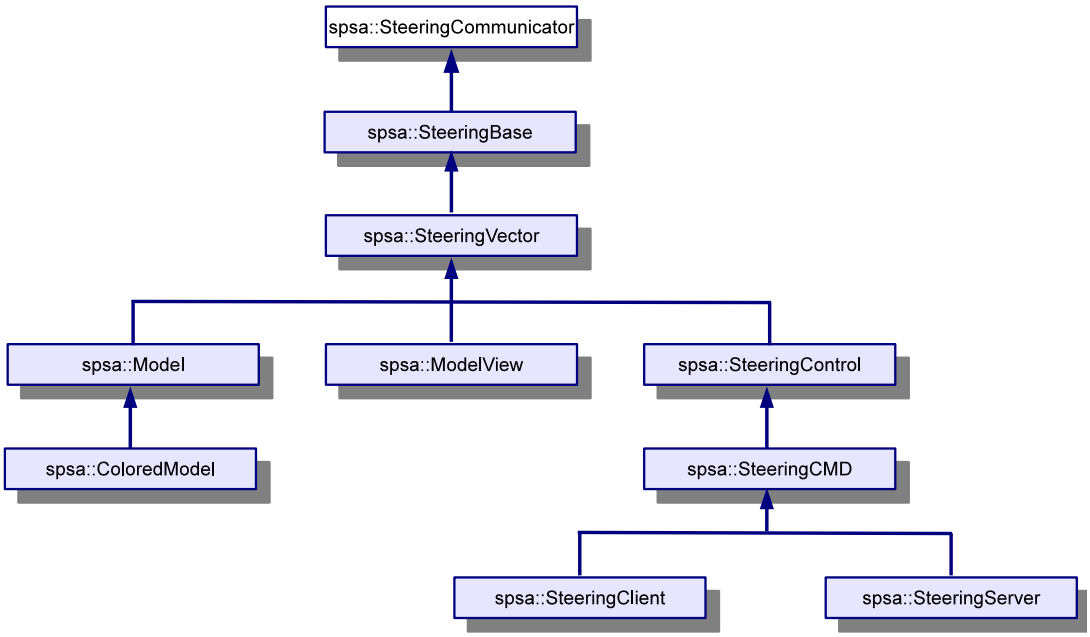


Figure 5.4: Inheritance diagram of Snoopy's steering APIs (SPSA). The Snoopy steering API classes can be classified into four main categories: communication, data structures, control commands, and end point components.

nents: 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 ones are used to encapsulate the Petri net models 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 5.4.

Model views are defined by the user through the steering GUI. They are saved on the server side and can further be downloaded whenever a user open a model. A view is defined by selecting a subset of the model places or transitions to monitor their values during the simulation. Each selected place (transition) is called a curve inside the view. A curve is defined by a set of attributes, e.g., colour, style, width and so on. In Snoopy,

when a model is executed for the first time, a default view is created.

Besides, each view is associated with a viewer. The viewer determines how the view's data is displayed. For instance a tabular viewer displays the simulation output in a table.

Views add many advantages to increase the user's experience of using the steering framework. They provide an easy way of exploring the model results. After the views are defined in a model, the user can explore the results by just turning the views. Additionally, views increase the collaboration among the users. Users on different computers are able to define different views independently from each other. After that they can share them by submitting their views to the server. Finally, views are defined "on the fly" while the simulation is running. Users does not need to interrupt the simulator to define a new view.

Moreover, the API library contains a number of control commands. 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 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 5.4).

5.3.5 Simulators

To combine both extendibility and ease of use, the proposed framework comprises two types of simulators: internal and external ones. The internal simulators are built into the steering server, while external simulators might be provided by the user as external simulation modules. In the following, we give a closer look to both of them.

Internal 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. Figure 5.5 provides a graphical representation of the relationships between the different simulation approaches discussed in Chapter 3 and which are now available in the Server. In each category, some specific algorithms are provided. For instance, for 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 [HBG+05]), or hybrid simulation with either static or dynamic partitioning (see Chapter 4). Snoopy provides steering commands to all of these algo-

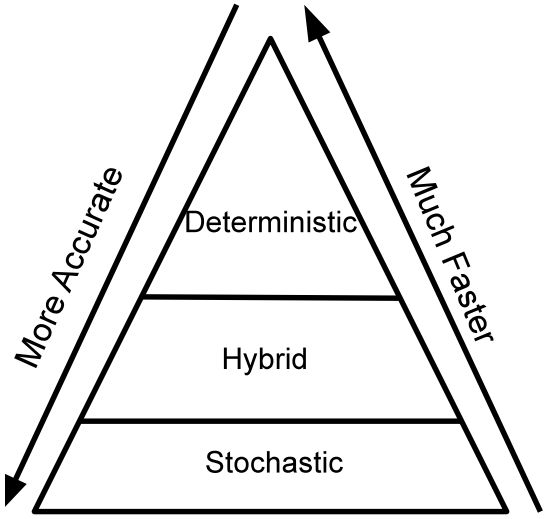


Figure 5.5: Summary of the different simulation approaches of biochemical reaction networks.

rithms. See [HHL+12], for more information about simulating continuous, stochastic and hybrid Petri nets using Snoopy. Figure 5.6 presents the inheritance diagram of the simulation algorithms that are currently supported in Snoopy.

The internal simulators are implemented using the object-oriented approach, and they are deployed as a stand-alone simulation library which can be called from other simulation tools. Previous versions of the simulation code of Snoopy [RMH10] are also integrated into the library. The resulting library is generic and extendable to support other simulation algorithms.

External Simulators

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 servers. The API library supports the registration of the simulation data at the servers which could later be used to accommodate the GUI requests.

To motivate the need of supporting external simulators in our framework, consider the following scenario. A scientific research group has its own simulation code that fulfils their need of implementing additional features, e.g., sensitivity analysis, steady state analysis, etc. However, this code works in batch mode only, i.e., it does not allow users to interact with the simulation. To add interactivity to their code, they might need a long time and consequently the entire focus might be shifted. Moreover, it requires them to be familiar with many computer science techniques, e.g., socket communication, multithreading, synchronisation, in order to use computational steering for their code. Using our framework, this aim could be achieved with modest efforts.

5.4 Backtracking

Backtracking is another important aspect to permit the user to rollback to a previous state of the simulation. This feature enhances the user interaction with the biochemical simulation and as a result gives the user a better understanding of the problem under study.

There are two main reasons (functionalities) of including backtracking to our framework: permitting the user to easily restart the simulation from a given history point, and documenting the manual interactions with the running simulator.

As the ultimate goal of the computational steering technique is to shorten the total experiment time by allowing the user to change the simulation parameters on the fly, it is helpful to let the user to restart the simulation from a certain history change point. The feature enables the exploration of different paths of the running model starting from the same time point.

Furthermore, for the simulation of a biological model to be reproducible, the user interactions with the running model need to be saved at each point the user changes any parameter of the running simulation.

The implementation of the backtracking function involves saving the user changes of the model parameter as well as the current marking whenever a manual intervention occurs. Later on, the user is presented with a set of history points to restart the simulator using one of them.

5.5 Steering Algorithms for Simulation of Biochemical Networks

In this section, we focus on the detailed algorithms which are used to perform the steering. However, before that we need to define which part of the model can be changed during the simulation. Common to all the simulation approaches is the way to perform the steering. It usually takes place before the numerical simulator performs a step. However, the simulation approaches differ in the level of granularity with respect to

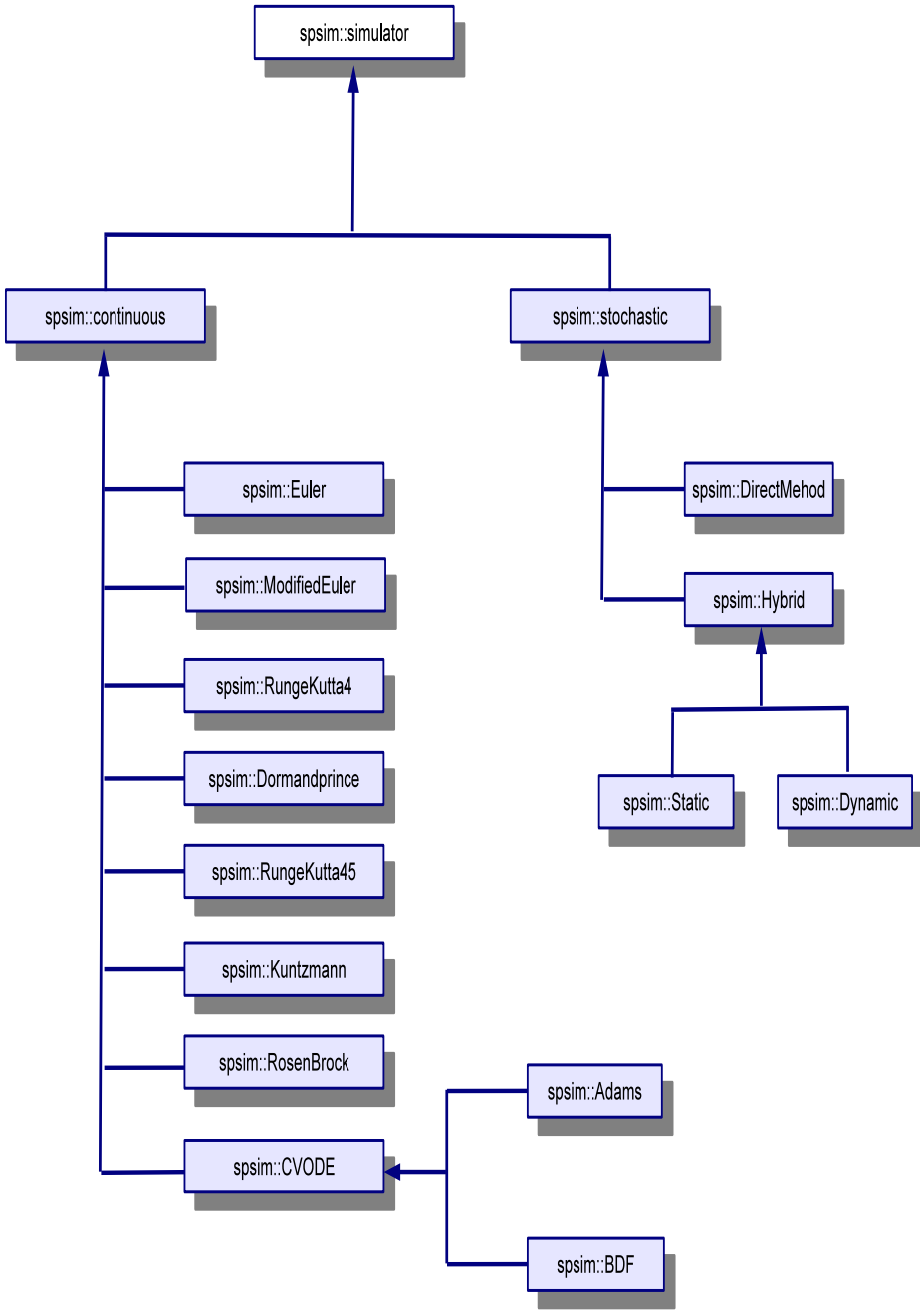


Figure 5.6: Inheritance diagram of the simulators supported in Snoopy. Snoopy's simulation library implements three categories of simulation approaches: continuous, stochastic and hybrid. In each category, there are some specific algorithms. The implementation of the simulation library is generic, extendable, and platform-independent.

performing parameter changes. For instance, the semantics of the stochastic simulation permits the user to change simulation parameters at each single firing of any transition. However, deterministic simulation allows changes to take place only at a defined grid resolution. In the following subsections, we discuss the steering of the deterministic, stochastic, and hybrid simulation algorithms.

5.5.1 What Could Be Steered

Before we discuss the details of introducing computational steering to a specific simulation approach, we consider in this section the question of what can be controlled by the user during a simulation. In other words, which aspects of the model definition can be changed while the simulation is running. Generally speaking, changes of a model specification can be classified into two main categories: adjusting the model parameters and adjusting the model structure.

Adjusting model parameters involves modifying place markings as well as kinetic constants which are used to define transition rates. Although modifications related to this category have a great impact on the understanding of the model understudy, they fortunately do not require much computational effort in order to respond to the user's changes (see next section), particularly if the implementation of the computational steering is implemented efficiently. Ideally, by the help of such technique, systems biologists can check many model properties without stopping the simulation. For instance, the model robustness to small perturbations of parameter values can be checked by changing key parameter values and monitor how the system responses. As a more specific example, consider a system that exhibits a steady state under certain parameter settings. While the system is in a steady state, the user can change some of the kinetic parameter values and checks if the steady state is affected by such modification or not.

Contrary, adjusting the model structure involves changing the number of places, the number of transitions, and/or the connection between places and transitions. The application of these types of changes renders the implementation of biological models in which their structure are changing with respect to time, e.g., self-modifying models. Contrary to adjusting model parameters, adjusting the model structure requires a substantial amount of computational overhead from the simulator's point of view. Indeed, the simulator requires – in this case – much time to adapt itself to the user's changes.

Hereinafter, we confine ourselves to the type of steering related to the former category, namely, adjusting model parameters, while we postpone the latter one to the outlook.

Algorithm 5.1: Collaborative steering algorithm of deterministic simulation

1: Initialise the ODE solver;
2: Record the initial state to the result matrix;
3: Set the simulator speed;
4: **while** current time $\leq$ end time AND isRunning() **do**
5: Respond to the current scheduled tasks;
6: Reinitialise the ODE solver;
7: Simulate one step;
8: Record the current state to the result matrix;
9: Update the current time;
10: **end while**

5.5.2 Deterministic Simulation

In this section, we consider the problem of changing transition rate constants as well as place markings while the deterministic simulation is in progress. From Section 3.3 we know that one of the methods of implementing the deterministic simulation is the numerical integration of the resulting ODEs by applying a certain kinetic rate law, e.g., mass-action. Additionally, Sections 3.6.2 and 4.2.4 discussed the generation of the corresponding ODEs from continuous and hybrid Petri nets respectively. Thus, the problem of quantitatively simulating continuous Petri nets is simplified to numerically integrating a set of coupled ODEs.

The algorithm presented here combines computational steering with the ODE solver without assuming any type of integration algorithms, e.g., stiff vs. unstiff or explicit vs. implicit methods. The only assumption here is that the ODE solver carries the numerical integration at certain time points (e.g., each 0.1 time unit). Such assumption does not restrict the use of standard ODE solvers, since almost all solvers permit recording their output results at certain time points according to the resolution specified by the user.

Algorithm 5.1 summarises the steps of integrating computational steering with the simulation of continuous Petri nets.

At step 1 the ODE integrator is initialised. The initialisation phase mainly depends on the type of ODE solver. For instance, simple single-step algorithms (e.g., Euler) require no initialisation compared to multi-steps implicit (e.g., BDF). However, we assume this step exists in our algorithm to keep it general.

Step 2 simply records the model state at τ_0 for output purposes, while line 3 sets the simulation speed (i.e., the how often the simulator can take a step) to the level selected by the user. The latter step does not exist in standard ODE integrators as the goal is to solve the problem as fast as possible. However, in our case the user might be interested in slowing down the simulator speed in order to have the chance to modify

model parameters or current marking. In our implementation, we support four levels of speeds: normal, medium, slow and very slow.

Afterwards, the algorithm enters a loop of repeating a number of steps until the simulation time reaches the specified end or the user manually stops it. It is worth mentioning here that the algorithm can be easily modified by the developer to loop forever until the user stops it to fulfil specific needs.

At line 5 the algorithm obtains the set of scheduled tasks as well as the steering commands that are defined by the collaborating users. Scheduled tasks are the users' changes of parameters or marking. The reason for such organisation will become clear in the subsequent discussion.

Next, in line 6 the ODE solver is reinitialised using the new parameter and marking values. This step is crucial for the ODE solver to account for discontinuity due to user changes, particularly for implicit ODE solvers that employ multi-steps algorithms (see Section 4.4.2).

After that (line 7), the ODE integrator takes one step forward. The step size here is equal to the output grid resolutions (i.e., the time points at which the simulator outputs the results). This could be equivalent to taking multiple steps using variable-step size solvers. Finally, the algorithm records the current model state and updates the current system time.

To explain the concept of scheduled tasks, which is introduced in this thesis, we compare our approach with other methods, e.g., [SWR+11]. In [SWR+11], the steering of model parameters is done by stopping the simulator at the appropriate time point and then notifying the user to take an action. However, what will happen if the user does not want to steer at this time point? One option is to instruct the simulator to continue or let the simulator automatically resume the computation after a certain time period expired.

Nevertheless, this approach is not computationally efficient. Additionally, it is not user friendly. Therefore, in our solution we let the user makes changes to the model at any time during the simulation. Such changes will be applied when we reach an appropriate time point (line 5 in Algorithm 5.1).

Moreover, this approach is well suited to eliminate conflicts between different users that are concurrently steering the same model.

5.5.3 Stochastic Simulation

Adding computational steering features to stochastic simulation is similar to the procedure of steering ODE models, however with two exceptions.

On the one hand, stochastic simulation algorithms are based on the Markovian property and therefore, there is no need to consider reinitialisation whenever a change is applied to parameters or place markings. Fortunately, this feature is useful when considering computational steering, because it keeps the computational efficiency as before

introducing the steering functionalities to the original simulation algorithm. Moreover, modification of model parameters can take place at each firing of a transition and not only at some output grid points. Particularly, these assumptions are valid for SSA algorithms (refer to Chapter 3 for more details).

On the other hand, stochastic simulation might require to perform more than a single run to produce the average model behaviour, while deterministic simulation requires only a single run to produce the complete results. This distinguishing feature of stochastic simulation necessitates the existence of two steering approaches. One approach is concerned with the steering of a single run while the other one is related to the steering of multiple runs. In both cases, the user will have the option to use either of the two methods.

In the former method, the algorithm of steering stochastic simulations will be similar to Algorithm 5.1. However, the step in line 7 will represent the firing of a single transition, while the user can monitor the model response only at the next output point. In the latter method, the user changes take place only between the individual runs. The reason for this is that the user will not be able to follow the model simulation on the basis of single transition firing. Instead, it is feasible to inject the modification of model parameters between the runs.

To sum up, the Markovian property of stochastic simulation facilitates the use of computational steering of biochemical models. Changes to the simulator can be applied in more fine-grained steps than in deterministic simulation.

5.5.4 Hybrid Simulation

Now we turn to hybrid simulation. In fact, the hybrid simulation is more similar (in its basic functionality) to stochastic simulation rather than to the deterministic one. For instance, to study model dynamics, multiple runs are typically required. This explains why the hybrid simulator is derived from the stochastic one, see Figure 5.6.

Nevertheless, hybrid simulation is not identical to the stochastic one. It internally combines the features of stochastic and deterministic simulation algorithms. As it has been shown in Chapter 4, the hybrid simulation algorithm consists of the firing of one or more discrete transitions followed by a step of the ODE solver that corresponds to the firing of continuous transitions. Therefore, the steering of a hybrid model can take place at the following points:

- before the firing of a discrete transition,
- before the ODE solver takes a step.

In the former case, there are no additional actions required to cope with user changes since the firing of a discrete transition depends only on the current making and the transition firing rates of stochastic transitions. Immediate and deterministically timed

transitions with zero delay might fire several times before simulating the continuous transitions.

In the latter one, the user changes can take place before the ODE solver takes a step, but this might be after a firing of a discrete transition. As in the purely deterministic steering algorithm, discussed in Section 5.5.2, the ODE solver requires reinitialisation before this step in order to account for any discontinuities due to the user interactions. However such reinitialisation is also required due to the firing of a discrete transition.

Obviously, if all transitions are stochastic ones, the hybrid steering algorithm will be equivalent to the stochastic algorithm, while if all transitions are continuous ones, the model can be steered using the deterministic steering algorithm.

5.6 Implementation Issues

In this section we discuss some of the implementation issues that rose during developing the computational steering framework. The discussed issues are: synchronising the model data strucure, the use of sockets and threads, communicating model specification and communicating the result matrix (i.e., simulation output).

5.6.1 Model Synchronisation

Due to the multi-user feature of our computational steering framework, concurrent writes or read/write of some running models are possible to occur. Therefore, a synchronisation mechanism is necessary to ensure the coherence of the model information at the server side.

In general this is a well known and classical problem of computer science, see e.g., [Ray86]. Hence, many solutions are available in the literature to deal with it. Here we identify three of them (from the implementation point of view). The first of these solutions is called "mutex". Mutex can coordinate mutually exclusive access to shared resources and allows only one thread at a time to own a mutex object (i.e., the synchronisation object). The second solution is to use a critical section to prevent the concurrent access to shared code. The difference between mutex and critical section is that under some implementations (e.g., MS Windows), the former can be visible for different processes, while the latter is visible only for one process. Finally, semaphores can be used to limit the number of threads concurrently accessing a shared resource.

In our specific scenario, we consider this issue as an optimisation problem: maximising user concurrency while preventing concurrent access to the model information. Therefore, we employ a synchronisation scheme at two different levels: global and local.

Global synchronisation is used to prevent concurrent access to information that is related to all users, models, or simulators. The synchronisation object is owned by the server itself and can be used whenever a user requests changes or a (new) user

joins/leaves the system. For instance, when a user submits a new model, all other users are prevented to access the model or user information.

Contrary, local synchronisation is performed only at the model level. Only users which share the same model could be interrupted when a user of the same group changes the model information. Each model has its synchronisation object to coordinate the operation of the users that are accessing this model as well as the model simulator. For example, consider again the concept of scheduled tasks which has been introduced in Section 5.3.2. While one user is appending a new task, the other users are not allowed to change the TO-DO task list. Similarly, while the simulator is dispatching a task, users are not permitted to add new ones.

5.6.2 Sockets and Threads

Clients and the servers are communicating with each other through network sockets. Network sockets are endpoints of the communication flow among two processes [SFR03]. Therefore, for our purpose of providing a platform-independent implementation of our framework, it is required to use a socket library which can communicate and run under different platforms. For this reason, we used the wxWidget socket library [wxW12]. However, we had to choose between different options such as: synchronous vs asynchronous communication, blocked/unblocked calls.

Furthermore, we extensively use threads to implement our framework. We can classify our use of threads into three categories: user threads, worker threads, and dispatcher threads.

Each user is represented inside the server as an independent thread. The user's thread is created as soon as the connection is successfully set up between the client and the server. The user thread is terminated when the user exits the system.

Furthermore, each model is associated with a worker thread to perform the simulation. The worker thread is created when the simulator is started and terminated when there is no active simulation for this model. Whatever the number of users that are connected to the same model, only one worker thread is created. This coincides with our design of running only one simulator and permits all other users to collaborate with each other. Worker threads are assigned a higher priority in comparison to user threads. This implies that computations have higher priorities over communications. Moreover, worker threads might spawn multiple different child threads in the case of using multi-threading to execute several simulation runs of stochastic or hybrid simulation.

Yet another thread type is the dispatcher thread. It is required to perform the initialisation of newly connected clients. This step involves sending/receiving the model specification. The dispatch thread releases the server main loop from the details to initialise the clients which is communication intensive tasks.

5.6.3 Communicating Model Specification

According to the envisaged scenario of implementing computational steering, as given in Figure 5.3, either the client or the server sends the model specification to the other side before the actual functionalities of computational steering can take place. In this step, the model specification has to be transmitted across a local area network (LAN) or over the internet. The latter communication type is very slow. The problem with this step is that the user can wait seconds or a few minutes until the actual simulation starts. However, they will not wait hours for such an initialisation step.

For a low level (i.e., uncoloured) Petri nets, such problem does not exist, because such models contain usually at most a few hundreds to thousands of places and transitions. However, the problem will apparently appear when considering coloured Petri net models, if they are unfolded.

For example, consider a coloured Petri net with three places, one transitions and three arcs, but with one thousand colours as it is depicted in Figure 5.7. The unfolded version of this model will contain three thousand places, one thousand transitions and three thousand arcs. Remember that for each model we have to send many information, e.g., place names, initial marking, transition names, etc. However this model is a simple one. An interesting property of coloured Petri nets is that they can scale very easily (see Chapter 3). Therefore, real coloured models might contain hundreds of thousands or millions of nodes. Thus, different strategies are required to overcome this problem.

One solution is to serialise place and transition names and issue one socket command. This will enforce the low-level library to send and receive packets of larger size. This solution will effectively reduce the communication time, particularly, when communicating using LAN. However, the packet size is often limited, especially over the internet.

Another solution is to compress the data before sending it and uncompress it on the other side. However, the question is how much compression ratio can we obtain. The answer will depend on the type of data, especially as we seek lossless data compression.

Nevertheless, this step needs to be done in a very short time to make the computational steering framework useable. If they have to wait a long time until the system has been initialised, they will not opt to use computational steering in exploring their case studies.

5.6.4 Communicating Output Matrix

Another related, yet interesting problem of communicating the model information is the transfer of the result matrix. That is how to communicate the simulator output to the user client. A simple calculation will be helpful to understand the problem.

Consider again our simple three-place coloured model. We assume that the result of the model is stored in an $M \times N$ matrix of type double, where M is the number of

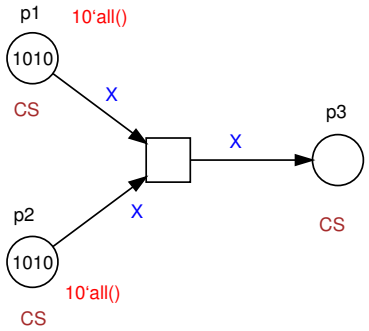


Figure 5.7: Example of a simple coloured Petri nets

output points and N is the number of nodes (places/transitions). The total amount of memory space required to store the result matrix can be calculated in terms of (5.1).

$$\text{Total matrix size} = M \times N \times sizeof(double) \tag{5.1}$$

For our simple example, to record the result of 100 time points, assuming the double data type is represented by 8 bytes, we need:

$$\text{Total matrix size} = 100 \times 3000 \times 8 = 2,400,000 \text{ bytes}$$

If we increase the number of colours to 100,000, we will need 240,000,000 bytes, and for 1,000,000, we need 2,400,000,000.

It is not feasible to transmit repeatedly 2,6 Gigabytes in a few seconds, to keep the system all the time responsive. Furthermore, the required space might be much more if the number of output points are increased (e.g., 1000 or 10,000 points).

However, this problem differs from the problem of communicating the model specification in two aspects:

- we have to communicate the result matrix each time the client needs to refresh (i.e., redraw the output);
- we have space for more optimisations.

One workaround is to apply the technique discussed in Section 5.6.3, namely, compressing the result matrix or using a bigger packet size. However, this will not solve the problem (imagine compressing 2.4 gigabytes, what will be the result!).

Another efficient and natural solution is not to send the entire matrix each time the user client needs a refresh, instead, we send only a part of the matrix. This is because users usually select a few places or transitions to view. Therefore, the problem of transmitting 2.4 gigabyte will be reduced to sending/receiving a few kilobytes or even

bytes. Moreover, in the case of exporting the entire matrix to a CSV (comma separated values) file, where all place values are needed, we can communicate the matrix over the HTTP protocol. In this case, it is acceptable for the user to wait for the export to take place. This solution has more potential optimisation. We can only send the matrix values which have been simulated so far. Usually, the simulator will not produce the results at one-go, instead, it takes time to produce the solution at each time step. This heuristics is mainly very useful when considering the simulation of a large number of output points.

Finally, we can perform the result visualisation at the server side. Therefore, we do not need to send the raw data to the client to carry out the visualisation. This step will keep the transmitted data size fixed, despite of the model size. However, this solution will increase the server load.

5.7 Comparison

In this section we compare our computational steering framework with other software frameworks. It is not feasible to conduct a comparison with all the computational steering based environments and Snoopy. Therefore, we select a few of the tools presented in Chapter 2.

Since the computational steering tools, which have been developed so far, are not dedicated to the problem of kinetic modelling of biochemical reactions as we do, we added three tools to our comparison list which explicitly support simulation of biochemical networks. These tools are: Cell Designer, Cell Illustrator, and VCells (see Chapter 1). However, none of them supports the computational steering technique. Indeed our work might bridge the gap between those two software type.

Table 5.1 summarises the different features of the Snoopy framework in the context of the other simulation environments according to some criteria. From this table we can conclude that Snoopy combines many useful features that are distributed across several problem solving environments. That is because Snoopy integrates computational steering and Petri nets in one framework.

Table 5.1: Comparing Snoopy's computational steering framework with other computational steering and biochemical modelling software

	Cell Designer	**Virtual Cell**	**Cell Illustrator**	**Snoopy**	**SCIRun**	**Discover**	**CUMULVS**
Application Area	kinetic modelling of biochemical reactions				bio-medical	general purpose	
Interactive Simulation				X	X	X	X
Model Representation	non-standard graphical notations		Petri nets		visual widgets		
Multi-user				X		X	X
Multi-model				X		X	
Multi-simulator				X		X	
Collaboration		X		X		X	X
Platform-independent	X	X	X	X	X	X	X
Distributed Components		X		X		X	X
Built-in Simulator	X	X	X	X			
External Simulator				X	X	X	X

5.8 Conclusions

In this chapter, 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 framework is provided as part of Snoopy. Moreover, this thesis proposes new features of biochemical kinetic modelling software to support interactivity. The proposed framework is also compared with other software architectures. In the Section 6.3.3 we present a case study that explains a typical application of computation steering in the context of modelling biochemical reaction networks.

6 Case Studies

In this chapter three case studies are discussed to illustrate the functionalities of $\mathcal{GHPN}_{bio}$ and the computational steering framework by biological examples. These are: the T7 phage model, the eukaryotic cell cycle, and the circadian oscillation model. The main advantages of using $\mathcal{GHPN}_{bio}$ to study these networks are the intuitive graphical representation of the system logic as well as the accurate simulation.

The T7 phage reaction network consists of three species and six reactions. $\mathcal{GHPN}_{bio}$ allow us to stochastically represent and simulate reactions which are important to accurately reproduce the model behaviour, while other reactions that do not influence the fluctuation of molecules are simulated continuously. Additionally, representing and simulating this model using $\mathcal{GHPN}_{bio}$ results in a substantial improvement in terms of the runtime over a pure stochastic simulation.

The eukaryotic cell cycle model is an ideal example to demonstrate most of the presented elements of $\mathcal{GHPN}_{bio}$. For instance, modelling the logic of making a decision to perform the division requires the adaptation of immediate transitions along with different types of extended arcs. Similarly, other features like marking dependent arc weights, logical nodes are also demonstrated using this example. An important requirement of this model is to capture the variability of the cellular volume to reproduce the "in vivo" experiment results. Such variability is due to either intrinsic or extrinsic noise. The former noise type can be accounted by simulating slow reactions using stochastic transitions.

The circadian oscillator case study presents another aspect which the hybrid simulation has to deal with. In this model, transition rates as well as species population change dramatically over time. Using a pre-determined partitioning of the transitions will not result in an improvement of the simulation runtime. Dynamic partitioning can better deal with such models. Moreover, we illustrate how computational steering can be used to study the effects of changing key parameter values on the period and amplitude of the resulting oscillation.

Finally, we conclude this chapter by comparing the runtime performance of simulating these models using different simulation methods as well as investigating different ODE solvers. Additionally, we propose some recommendations of the suitability of each simulation approach for specific case studies.

6.1 The T7 Phage Model

In this section we start demonstrating the operations of $\mathcal{GHPN}_{bio}$ through a typical biological example, the intracellular growth of bacteriophage T7. Our selection of this case study is motivated by three reasons: first, this is a simple and clear example to show how stochasticity can play a role. The continuous and stochastic simulation results suggest different conclusions. Second, the model reactions and the system dynamics can easily be understood which gives us the chance to concentrate on the syntax and semantics of the $\mathcal{GHPN}_{bio}$. Indeed, the T7 phage model contains only six reactions, three species, and ten arcs. Finally, the model's reactions can be spilt into fast and slow ones. Therefore, this model has often been used in the literature to discuss the principles of stochastic or hybrid simulation algorithms (e.g., see [KMS04, ACT$^+$05, YLL09]).

Table 4.1 (see page 74) presents the different reactions that are involved in this model, while Figure 4.11 (see page 75) shows the graphical representation of the reaction set as $\mathcal{GHPN}_{bio}$.

Generally speaking, the growth of a virus is determined, within the cell, by a complex interplay of transcription, translation, assembly and virus release processes. A virus infection may be initiated by a single virus particle that delivers its genome to its host. The Petri net representation of the T7 viral model consists of two components, which have been adapted from [SYSY02], where only stochastic and deterministic versions are presented: the viral nucleic acids and a viral structural protein (struct).

The viral nucleic acids are further classified as genomic (gen) or template (tem). The genome is the vehicle by which viral genetic information is transported. The genome can undergo one of two reactions. The first possibility is to be modified and form tem (R_1), and the second one is to be packaged within the structural proteins to form progeny virus (R_4). The standard sequence of viral replication events involves: (1) the amplification of the viral template after an infection, and (2) the production of progeny virus.

If we apply the mass-action kinetic law, which has been discussed in Chapter 3, to the reactions in Table 4.1, we will get the following coupled set of ODEs:

$$\begin{aligned}
\frac{d[tem]}{dt} &= c_1 \cdot [gen] - c_2 \cdot [tem] \\
\frac{d[gen]}{dt} &= c_3 \cdot [tem] - c_1[gen] - c_4[gen][struct] \\
\frac{d[struct]}{dt} &= c_5 \cdot [tem] - c_6 \cdot [struct] - c_4[gen][struct].
\end{aligned} \tag{6.1}$$

Using (6.1) the deterministic simulation approach can produce the model dynamics that is shown in Figure 6.1. Linear stability analysis of the deterministic model revealed the existence of two steady state nodes [SYSY02]. One unstable steady state occurs

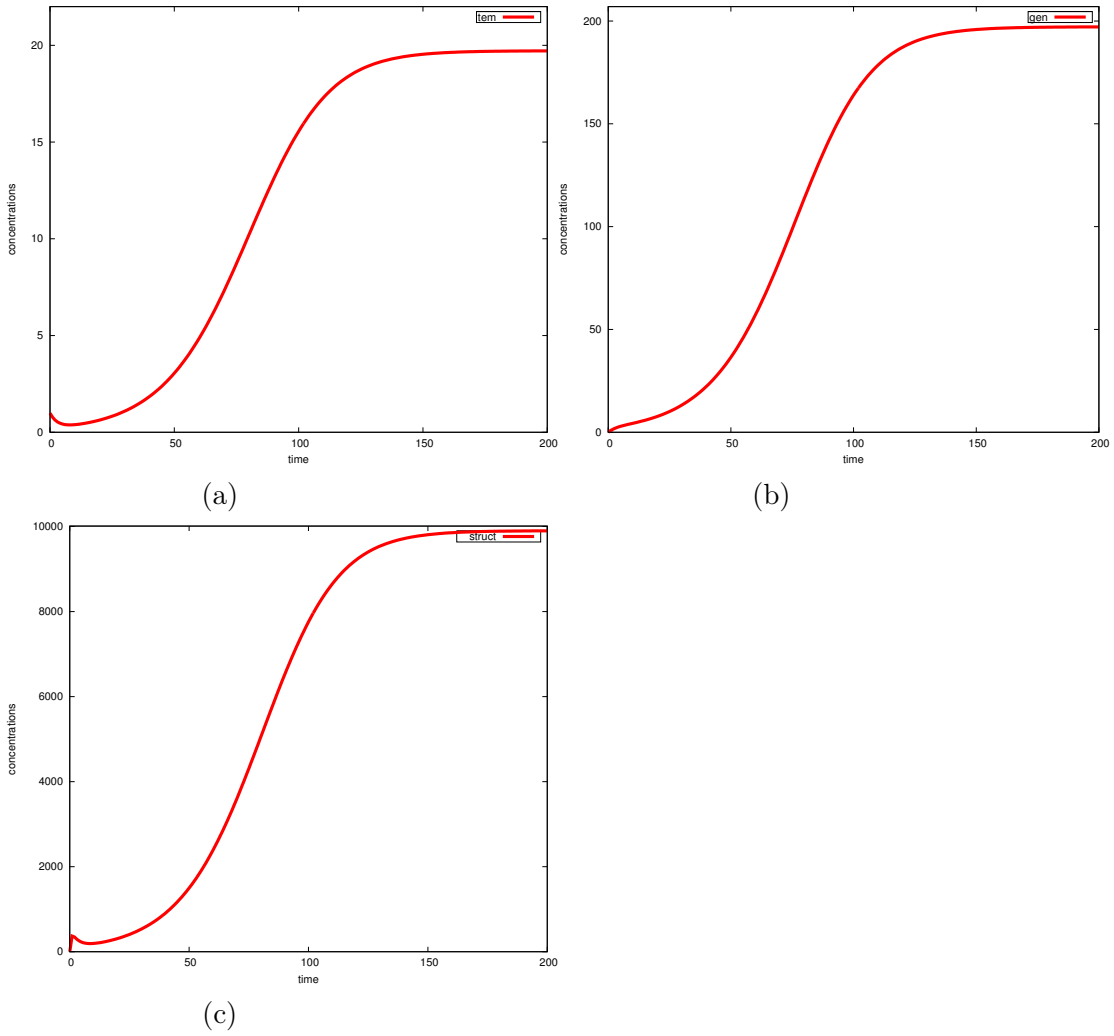
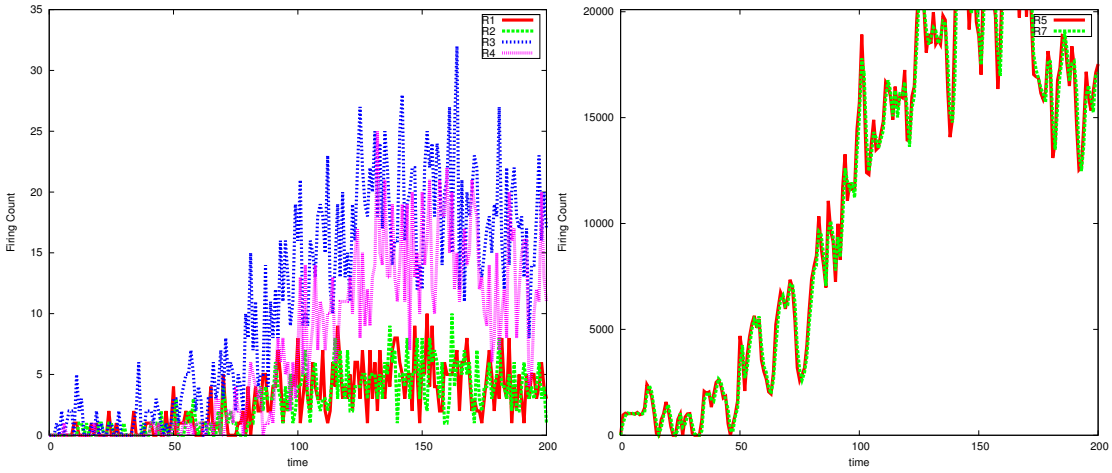


Figure 6.1: Continuous simulation results of the T7 phage model. (a) *tem*, (b) *gen*, and (c) *struct* concentrations. A stable steady state at *tem*=20, *gen*=200, and *struct*=10000 can be obtained.

when $tem = gen = struct = 0$. The other one is stable and occurs when *tem*=20, *gen*=200, and *struct*= 10,000. Srivastava et. al. assert in [SYSY02] that stochastic simulation does not always result in those steady states. In the following we discuss their conclusions and show that these conclusions are also valid when the hybrid approach is used. However, we firstly discuss how the reactions can be partitioned into those which are simulated continuously and those which are simulated stochastically.



(a) (b)

Figure 6.2: Comparison of the T7 Phage model's reaction rates. (a) reactions with low rates (R_1 - R_4), and (b) reactions with high rates (R_5 and R_6)

6.1.1 Slow and Fast Reactions

To illustrate the difference of rates among the reactions of the T7 viral model, we run a single stochastic simulation and analyse the results.

Figure 6.2 shows a time course simulation result of the number of firing of each transition of the T7 phage model.

The numbers of firing of transitions R_1 and R_2 are rather limited (Figure 6.2a). They do not fire more than seven times at each time step. Transitions R_3 and R_4 fire more frequently than R_1 and R_2. However, they exhibit also low firing frequency.

Contrary, transitions R_5 and R_6 (Figure 6.2b) have much higher firing frequencies than R_1 – R_4. Therefore, as shown in Figure 4.11, it is reasonable in terms of performance to represent and simulate reactions R_1 – R_4 using continuous transitions, while representing and simulating reactions R_5 and R_6 as stochastic ones.

6.1.2 Simulation Results

In this section we study the simulation result of the $\mathcal{GHPN}_{bio}$ model in Figure 4.11. We use the kinetic parameters in Table 4.1. The system is initiated using one molecule in *tem*.

While the model exhibits a stable steady state, using the deterministic simulation, stochastic and hybrid simulation results in Figure 6.3d show that the model enters a steady state at *tem*=13. To interpret such disagreement, we analyse single stochastic and hybrid simulation runs of the species *tem*.

Figure 6.3a shows a single run result of species *tem*. The model enters a steady state at *tem*$\simeq$20. This result coincides with the one produced via the deterministic simulation.

Now we consider the simulation result in Figure 6.3b. The model enters the non-stable steady state at *tem=gen=struct=0*. The reason for such behaviour is that reaction R_2 is fired and *tem* dies. Therefore, no further reactions will occur in the subsequent time points. This behaviour is a special one of stochastic and hybrid simulation and does not occur in the deterministic case. With other words, the dead state is hidden in the continuous setting.

Contrary, Figure 6.3c shows that the system enters the non-stable state for some time and then it recovers and enters the stable state at future time. Such die and recovery occurs when the *gen* is firstly synthesised using the current *tem* molecule (R_3) and then *tem* is lost (R_2). At some later time point, the *gen* is converted into a new *tem* allowing the virus to recover [SYSY02].

Now, we come to interpret the disagreements between hybrid and continuous simulation results in Figure 6.3d. Although stochastic results should be the same, in the average, to the deterministic one, we do not obtain such behaviour of the model under consideration. If we sum up the values of *tem* in the different runs, simulation results that exhibit steady state at *tem=0* will cancel some values at the stable steady state. Therefore, the average simulation result of the stochastic and hybrid simulations yields steady states at lower values than the continuous simulation. Srivastava et al. proved this explanation in [SYSY02] by eliminating the simulation runs where the model exhibits the non-stable steady states. They found out that stochastic and continuous simulation results are equivalent to each other after such post-processing.

Hybrid simulation can account for such stochasticity, since all the reactions related to *tem* are simulated stochastically. The major gain of using hybrid simulation over the stochastic one is the simulation runtime (see Section 6.4).

6.2 The Eukaryotic Cell Cycle

System level understanding of the repetitive cycle of cell growth and division is crucial for disclosing many unexpected principles of biological organisms. The deterministic or stochastic approach are alone not sufficient to study such cell regulation due to the complex reaction network and the existence of reactions with different time scales. Thus, integration of both approaches is advisable to study such biochemical networks.

The reproduction of eukaryotic cells is controlled by a complex regulatory network of reactions known as cell cycle [TN01, MCN08, SGCK$^+$08]. Through it, cells grow, replicate and divide into two daughter cells [KBPT09, SSJT11]. This regulation cycle consists of four phases: S phase (synthesis) and M phase (mitosis) separated by two gap phases: G1 and G2 [TN01]. During the synthesis phase, the cell replicates all of

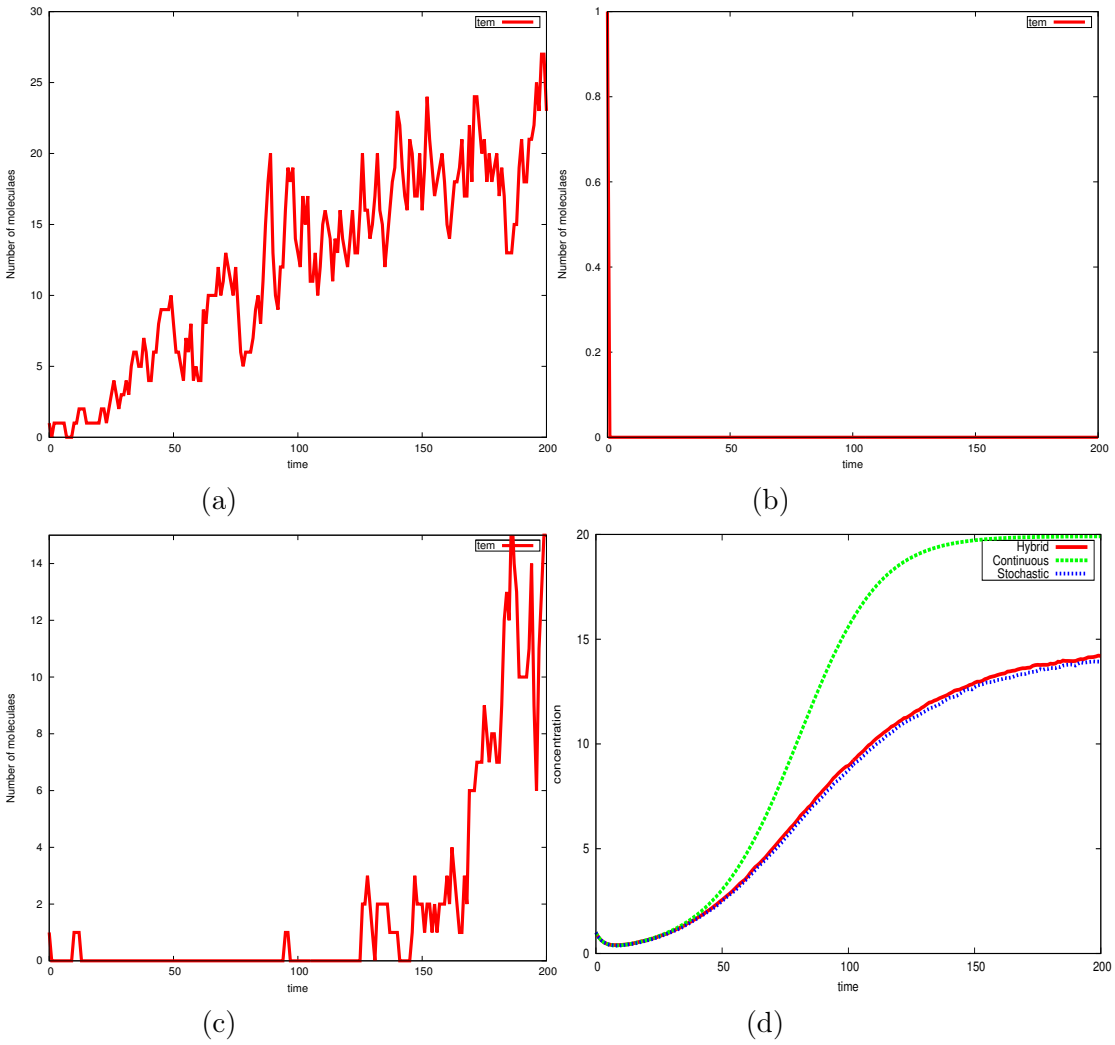


Figure 6.3: Continuous, stochastic, and hybrid simulation results of T7 Phage model.

its components, while it divides each component type more or less evenly between the two daughter cells at the end of the mitosis phase [KBPT09]. After the S phase, there is another gap (G2) where the cell ensures that the duplication of DNA has completed and prepares itself for mitosis. Newborn cells are not replicated and located at the G1 gap. Furthermore, the processes of synthesis and mitosis alternate with each other during the reproduction process. Understanding such control cycles is crucial for revealing defects in cell growth which underly many human diseases (e.g., cancer) [TN11]. Figure 6.4 provides a graphical overview of the cell cycle phases.

In the eukaryotic cell cycle, the alternation between the S and the M phase as

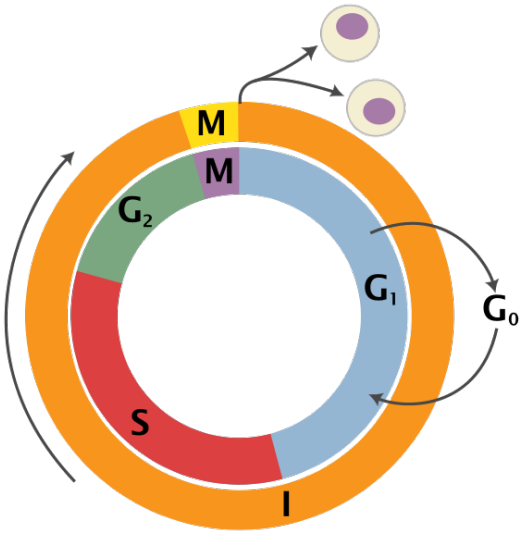


Figure 6.4: Graphical illustration of the cell cycle regulation [wik12]. The cell cycle consists of four distinct phases: G_1, synthesis (S), G_2, and mitosis (M). The first three phases are known as interphase (referred to by the outer ring). Cells that have stopped dividing are called entering the G_0 phase.

well as the balance of growth and division is governed by the activity of a family of cyclin-dependent protein kinases (CDK) [KBPT09]. Therefore, many computational models have been constructed to study the control system of CDK (e.g., in [TN01, CCCN+04, MCN08, SGCK+08, KBPT09]). Some of these models are based on the deterministic approach which represents changes of species concentrations as continuous variables that deterministically and continuously evolve with respect to time. However, such approach does not capture the variability of cell size due to the fluctuation of some species which usually exist in low numbers of molecules [Gil07]. Motivated by this argument, a number of stochastic models have been created and simulated using either a stochastic simulation algorithm (e.g., [KBPT09]) or by introducing noise to the model through Langevin equation [Ste04]. However, the stochastic approach is computationally expensive, particularly when the model under study contains reactions of high rates or species with large numbers of molecules.

In this section we present another argument to motivate the hybrid simulation of the cell cycle control system. The cell cycle model contains some components which would be better represented as continuous processes (e.g., volume growth), while other reactions of low rates are vital to be represented as stochastic processes. For instance, Mura and Csikasz-Nagy constructed in [MCN08] a stochastic version of the model in [CCCN+04] using stochastic Petri nets. However, they faced the problem of repre-

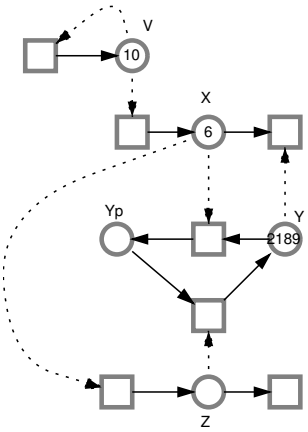


Figure 6.5: A continuous Petri net representation of the Tyson-Novak Model: X (*CycB-Cdk1*) phosphorylates Y (*Cdh1-APC*) and free Y catalyses the degradation of X. Z denotes the effects of *Cdc20* and *Cdc14*. High activity of X promotes the synthesis of *Cdc20* which activates *Cdc14*. The dephosphorylated *Cdc14* activates Y. This behaviour results in a bistable switch that is responsible for the transitions between *G1* and *S-G2-M* states.

senting cell growth processes which evolve continuously and exponentially with respect to time using stochastic Petri net primitives only. Indeed cell growth is a typical example where continuous transitions could be used. Moreover, the model which we propose is graphically and intuitively represented in terms of Petri nets.

6.2.1 Related Work

Mura and Csikasz-Nagy created in [MCN08] a stochastic model based on the work of [CCCN$^+$04] to study the effect of noise on cell cycle progression. However, their model is based on phenomenological rate laws (e.g., Michaelis-Menten) which do not work well with stochastic simulation algorithms [KBPT09] (see Section 3.4.2 to compare Gillespie's idea). Moreover, some components could not intuitively be represented using stochastic Petri net primitives only (e.g., cell growth). Sabouri-Ghomi et al. [SGCK$^+$08], and Kar et al. in [KBPT09], asserted that applying Gillespie's stochastic simulation algorithm [Gil76] directly to phenomenological rate laws might produce incorrect results. Therefore, they unpacked the deterministic model of Tyson-Novak [TN01] in terms of elementary mass-action kinetics. The Tyson-Novak model is based on a bistable switch between the complex CycB-Cdk1 (denoted by the variable X) and the complex Cdh1-APC (denoted by the variable Y). Figure 6.5 is a continuous Petri net representation of the Tyson-Novak. CycB-Cdk1 phosphorylates Cdh1-APC and free Cdh1-APC catalyses the degradation of CycB-Cdk1. To model a complete cell

cycle, Kar et al. [KBPT09] unpacked the effect of Cdc20 and Cdc14 which are lumped in the variable Z in the Tyson-Novak model. High activity of CycB-Cdk1 promotes the synthesis of Cdc20 which activates Cdc14. Finally the dephosphorylated Cdc14 activates Cdh1-APC. The Kar et al. model accounts for both intrinsic and extrinsic noises. Intrinsic noise is due to the fluctuation of species with low numbers of molecules, while extrinsic noise is due to the unequal division of the cell between the two daughter cells [KBPT09].

In [SSJT11], a hybrid model which combines ordinary differential equations (ODEs) and discrete Boolean networks has been constructed to adopt quantitative as well as qualitative parts in the same model. The Boolean networks approach requires less knowledge about realistic kinetic rate constants. Liu et al. [LPL+12] simulate the stochastic model of [KBPT09] using the Haseltine and Rawlings approach [HR02]. However, such models cannot structurally or graphically be represented which makes it unmanageable for further extensions.

In this section a hybrid Petri net model of the eukaryotic cell cycle is presented. The model is hybrid in the sense that it combines continuous, stochastic and immediate transitions to represent deterministic, stochastic and control components. Using Snoopy's simulator, it can be simulated either using the deterministic or the hybrid simulator.

6.2.2 The Model

Figure 6.6 shows the hybrid Petri net model based on the previous one introduced by Kar et al. [KBPT09]. The Petri net model is adopted from the reaction set in Table 6.1. It consists of 26 places, 51 transitions, and 164 arcs. Proteins, genes and mRNAs are represented by places. Transitions represent reactions. We use the same kinetic parameters and initial values as given in [KBPT09]. Moreover, we use Snoopy's logical node features to simplify connections between different nodes. For example, place X and Y are involved in many reactions which decreases the network's readability. We repeat them multiple times with the same names to keep the model understandable (logical places); likewise the immediate transition *divide* (logical transitions). Furthermore, the increase of cell volume size is intuitively represented by a continuous transition with a rate $\mu \cdot V$, where μ is the growth factor and V is the cellular volume.

In the model, continuous transitions simulate the corresponding reactions deterministically, while stochastic transitions carry them out stochastically. The latter transitions are responsible for molecular fluctuations. Immediate transitions monitor the model evolution and perform the division when the free number of molecules of Cdh1_APC reaches a certain threshold ($\hat{Y} = Y + YX + XY$). Please note that the variable $\hat{Y}$ is computed during the model simulation by the weight of the immediate transitions.

In the sequel we present in more detail some of the model's key components and the corresponding $\mathcal{GHPN}_{bio}$ representations.

No.	**Reaction**	**Propensity**	**Rate constant**
R_1	$mRNA\_x \rightarrow mRNA\_x + X$	$k_1 * mRNA\_x * V$	k_1=2.5
R_2	$X \rightarrow \phi$	$k_2 * X$	k_2=0.12
R_3	$Y + X \rightarrow Y\_X$	$k_3 * X * Y/V$	k_3=1.7976
R_4	$Y\_X \rightarrow Y + X$	$k_4 * Y\_X$	k_4=12.0
R_5	$Y\_X \rightarrow \phi + Y$	$k_5 * Y\_X$	k_5=3.0
R_6	$X + Y \rightarrow X\_Y$	$k_6 * X * Y/V$	k_6=7.875
R_7	$X\_Y \rightarrow X + Y$	$k_7 * X\_Y$	k_7=42.0
R_8	$X\_Y \rightarrow Yp + X$	$k_8 * X\_Y$	k_8=105.0
R_9	$Z + Yp \rightarrow Z\_Yp$	$k_9 * Z * Yp/V$	k_9=44.97
R_{10}	$Z\_Yp \rightarrow Z + Yp$	$k_{10} * Z\_Yp*$	k_{10}=12.0
R_{11}	$Z\_Yp \rightarrow Z + Y$	$k_{11} * Z\_Yp*$	k_{11}=30.0
R_{12}	$Yp + X \rightarrow Yp\_X$	$k_{12} * Yp * X/V$	k_{12}=22.497
R_{13}	$Yp\_X \rightarrow Yp + X$	$k_{13} * Yp\_X$	k_{13}=12.0
R_{14}	$Yp\_X \rightarrow \phi + Yp$	$k_{14} * Yp\_X$	k_{14}=0.003
R_{15}	$mRNA\_z \rightarrow mRNA + Z$	$k_{15} * mRNA\_z * V$	k_{15}=1.2883
R_{16}	$Z \rightarrow \phi$	$k_{16} * Z$	k_{16}=0.3
R_{17}	$TF + X \rightarrow TFp + X$	$k_{17} * TF * X/V$	k_{17}=0.01797
R_{18}	$TFp + H \rightarrow TF + H$	$k_{18} * TFp * H/V$	k_{18}=0.03594
R_{19}	$2TFp \rightarrow TFp2$	$k_{19} * TFp * (TFp - 1)/V$	k_{19}=0.81
R_{20}	$TFp2 \rightarrow 2TFp$	$k_{20} * TFp2$	k_{20}=90.0
R_{21}	$G + TFp2 \rightarrow C$	$k_{21} * G * TFp2$	k_{21}=0.0957
R_{22}	$C + TFp2 \rightarrow G + TFp2$	$k_{22} * C * TFp2$	k_{22}=2.7
R_{23}	$Yp \rightarrow Y$	$k_{23} * Yp$	k_{23}=7.0
R_{24}	$C \rightarrow C + mRNA\_z$	$k_{24} * C$	k_{24}=1.35
R_{25}	$mRNA\_z \rightarrow \phi$	$k_{25} * mRNA\_z$	k_{25}=3.5
R_{26}	$mRNA\_tf \rightarrow mRNA\_tf + TF$	$k_{26} * mRNA\_tf/V$	k_{26}=1.607
R_{27}	$TF \rightarrow \phi$	$k_{27} * TF$	k_{27}=0.01936
R_{28}	$TFp \rightarrow \phi$	$k_{28} * TFp$	k_{28}=0.01936
R_{29}	$TFp2 \rightarrow TFp$	$2 * k_{29} * TFp2$	k_{29}=0.01936
R_{30}	$mRNA\_h \rightarrow mRNA\_h + H$	$k_{30} * mRNA\_h * V$	k_{30}=1.607
R_{31}	$H \rightarrow \phi$	$k_{31} * H$	k_{31}=0.00968
R_{32}	$mRNA\_y \rightarrow mRNA\_y + Y$	$k_{32} * mRNA\_y * V$	k_{32}=1.607
R_{33}	$Y \rightarrow \phi$	$k_{33} * Y$	k_{33}=0.01936
R_{34}	$Yp \rightarrow \phi$	$k_{34} * Yp$	k_{34}=0.01936
R_{35}	$X\_Y \rightarrow \phi$	$k_{35} * X\_Y$	k_{35}=0.01936
R_{36}	$Z\_Yp \rightarrow \phi$	$k_{36} * Z\_Yp$	k_{36}=0.01936
R_{37}	$Yp\_X \rightarrow \phi$	$k_{37} * Yp\_X$	k_{37}=0.01936
R_{38}	$Y\_X \rightarrow \phi$	$k_{38} * Y\_X$	k_{38}=0.01936
R_{39}	$Gx \rightarrow Gx + mRNA\_x$	$k_{39} * Gx * V$	k_{39}=1.0
R_{40}	$mRNA\_x \rightarrow \phi$	$k_{40} * mRNA\_x$	k_{40}=3.5
R_{41}	$Gy \rightarrow Gy + mRNA\_y$	$k_{41} * Gy * V$	k_{41}=7.0
R_{42}	$mRNA\_y \rightarrow \phi$	$k_{42} * mRNA\_y$	k_{42}=3.5
R_{43}	$Gtf \rightarrow Gtf + mRNA\_tf$	$k_{43} * Gtf * V$	k_{43}=7.0
R_{44}	$mRNA\_tf \rightarrow \phi$	$k_{44} * mRNA\_tf$	k_{44}=3.5
R_{45}	$Ggh \rightarrow Ggh + mRNA\_gh$	$k_{45} * Ggh * V$	k_{45}=7.0
R_{46}	$mRNA\_gh \rightarrow \phi$	$k_{46} * mRNA\_gh$	k_{46}=3.5
R_{47}	$C \rightarrow TFp + G$	$2 * k_{47} * C$	k_{47}=0.01936

Table 6.1: Reaction set of the eukaryotic cell cycle model

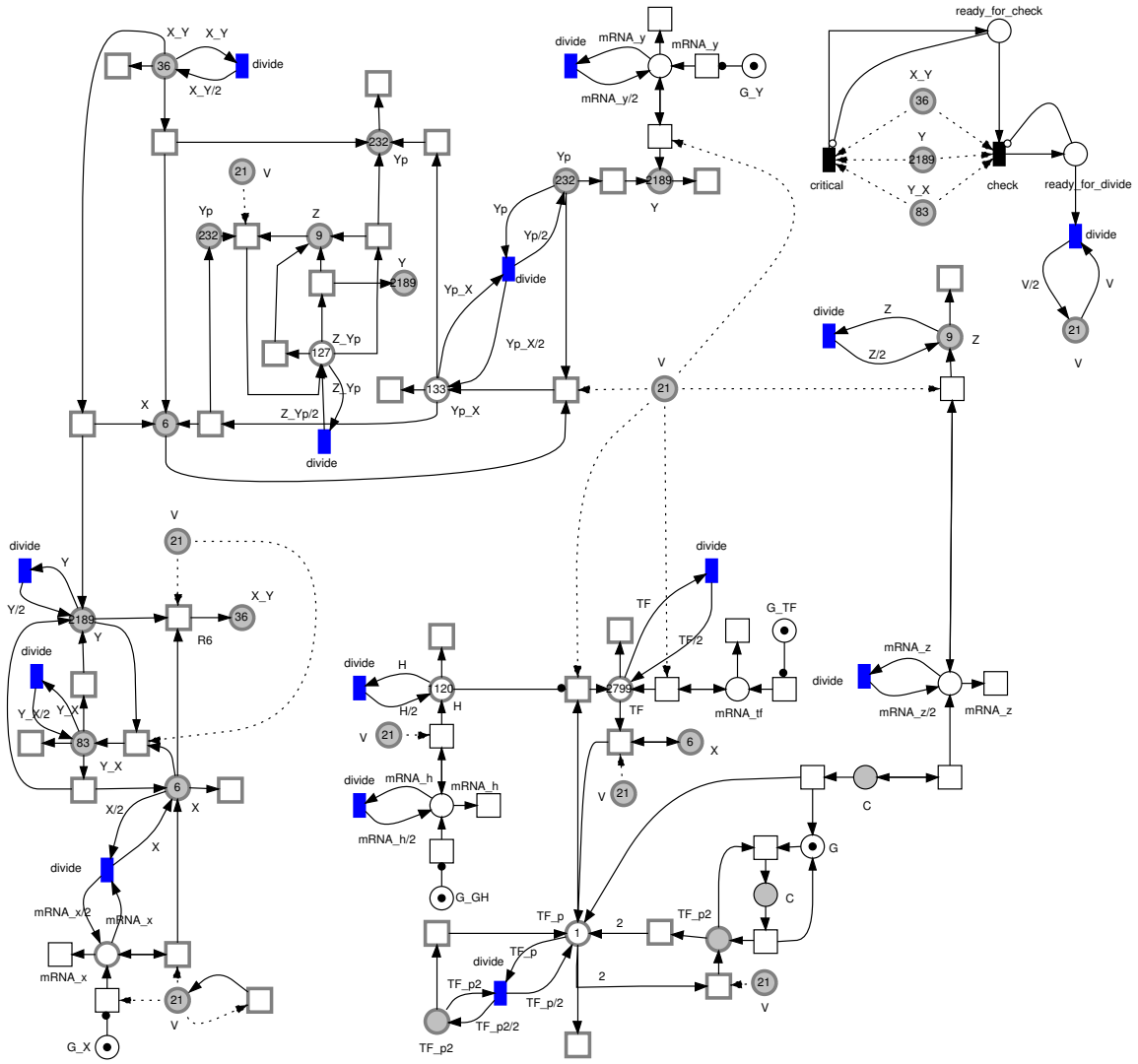


Figure 6.6: A $\mathcal{GHPN}_{bio}$ representation of the eukaryotic cell cycle. The model employs different types of transitions: continuous, stochastic and immediate. All reactions affecting mRNAs are represented and simulated stochastically. Repetitive nodes (places and transitions) with the same name are logical nodes (highlighted in grey). When the transition *divide* fires, it divides the current place marking more or less equally. The type of division (equal, or unequal) depends on the outgoing arc weight specified by marking dependent weights.

6.2.3 Decision to Perform Division

When the number of molecules of $\hat{Y}$ becomes greater than a certain threshold (in our case 1200), the cell can divide the mass and other components (mRNAs and proteins) between the two daughter cells. In Figure 6.7 (which is a subnet of Figure 6.6), this process is represented by an immediate transition *check* with the weight $\hat{Y} > threshold$. Recall that immediate transition weights determine the firing frequencies of immediate transitions in conflict. A weight of zero means that a transition can not fire at all. Therefore, when the transition *check* has weight greater than zero, it adds a token to the place *ready_to_divide* that triggers the transition *divide* to carry out the division. To give the transition *divide* a chance to fire before re-checking the value of $\hat{Y}$, an inhibitor arc is used to constrain this case.

An interesting characteristics of the model is the division process. Although the division can take place when the value of $\hat{Y}$ is greater than a certain threshold, it does not do that all the times. For example, at the beginning of the simulation, the initial value of $\hat{Y}$ satisfies the division criterion. However; the cell should not divide because it is still at G1 phase which means that it has to replicate before it can divide. We model this sitiuation by adding a new immediate transition which detects the critical value of $\hat{Y}$, before checking for division. Therefore the transition *critical* monitors the value of $\hat{Y}$. When the value of $\hat{Y}$ goes below a certain threshold, it enables the division process. For a simulation trace of this scenario, see Figure 6.8.

6.2.4 Cell Division and Marking-dependent Arc Weights

When a cell divides, it divides all of its components more or less evenly between two daughter cells. This is another ideal case to demonstrate marking-dependent weights which have been introduced in [Val78]. In Figure 6.7, when the transition *divide* fires, it removes all of the current marking of the place *V* and adds *V/2* to it. To permit uneven division of the cell volume and other components, arc weights can be a function which operates on the current place marking [MTA+03]. However, we restrict the used places in arc weights to the transitions' pre-places to maintain the Petri net structure.

In Figure 6.7 we model the logics of the division process by the transition *divide*. When there is a token in the place *ready_for_divide*, the transition *divide* becomes enabled. When it fires, it consumes *V* markings and add *V/2* to the place volume. Moreover the division takes place for each component making *divide* better be represented as a logic transition.

6.2.5 Transition Partitioning

The eukaryotic cell model contains transitions which fire at different rates. For instance, the transition (reaction) *R3*, as illustrated in Figure 6.9a, fires more frequently than *R1*. Slow transitions should be simulated stochastically to account for molecular

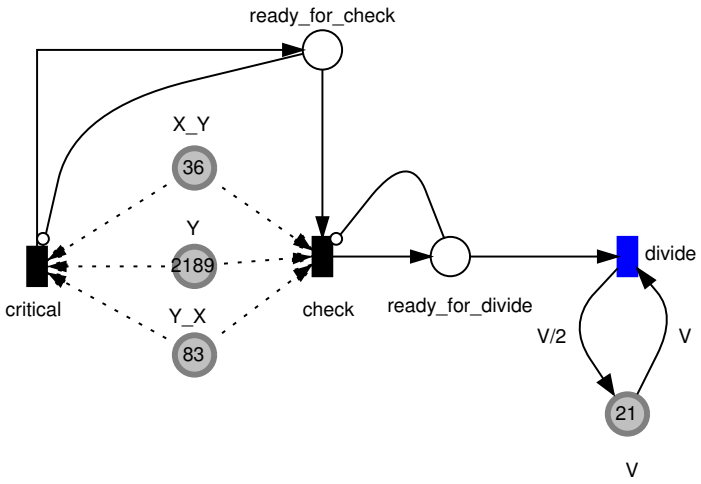


Figure 6.7: A sub-net for modelling the decision of the division process. The transition *critical* monitors the value of $\hat{Y}$ and adds a token to *ready_for_check* when $\hat{Y} < 300$. Later, when the value of $\hat{Y}$ increases and becomes greater than a threshold (1200), the transition *check* fires and adds a token to *ready_for_divide* which trigger the transition *divide* to perform the division. Inhibitor arcs are used as checkpoints for the sequence of events: *critical* → *check* → *divide*.

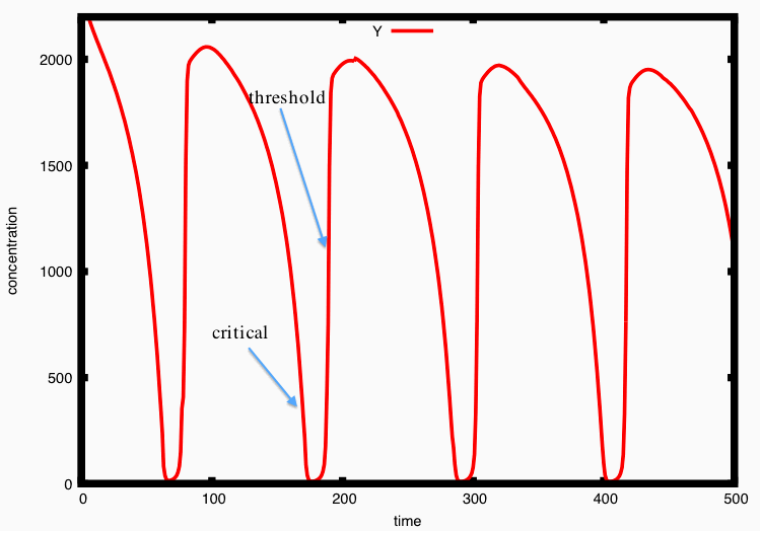
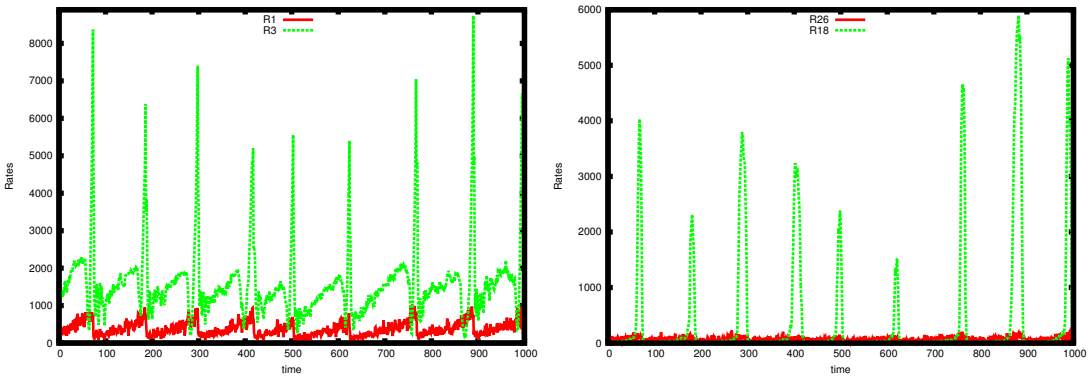


Figure 6.8: Graphical illustration of when a cell divides



(a) (b)

Figure 6.9: Example of different transition firing rates. (a) transition *R3:* $X+Y \rightarrow Y\_X$ fires more frequently than transition *R1:* $mRNA\_x \rightarrow mRNA\_x+X$, and (b) transition *R18:* $H \rightarrow H+TF$ fires more frequently than transition *R26:* $mRNA\_tf \rightarrow mRNA\_tf+TF$.

fluctuations, while fast transitions need to be simulated continuously to increase the numerical efficiency. Indeed, the latter types consume the majority of computational resources.

In this model, transitions are partitioned statically before the simulation starts. The transition type is decided by executing a single run and analyse the results as it is shown in Figure 6.9. Increasing (decreasing) the accuracy of the model's results involves converting more continuous (stochastic) transitions into stochastic (continuous) ones. Similarly, controlling the speed of the model simulation will require the opposite procedure.

Nevertheless, in any case cell growth has to be represented and simulated continuously. Using off-line partitioning, this can be easily told to the simulator by drawing a continuous transition. However, in the case of dynamic partitioning, the transition rate threshold should be set less than the expected rate of cell growth which makes the simulation of this model using dynamic partitioning not possible.

6.2.6 Simulation Results

In this section, we compare the simulation results of the following scenarios: when reactions related only to mRNAs are simulated stochastically, and when all reactions are simulated continuously. In all cases cell growth is simulated using a continuous transition. Figures 6.10 - 6.12 show the results of the continuous and hybrid approaches for three species. Since reactions related to mRNAs are simulated stochastically in

hybrid and stochastic simulations, their results are close to each other.

Figure 6.10 shows time course simulation results of protein Y. In hybrid simulations, Y is affected by fluctuations of mRNAs, while in continuous one there is no such effect.

Figure 6.13 compares continuous and hybrid simulation results for the cellular volume (V). Using continuous simulation, cells divide all the time equal and the model produces no variability in its volume size. The hybrid simulation shows variability in the volume size because species of low numbers of molecules (e.g., mRNAs) are simulated stochastically which account for the molecular fluctuations and therefore, they are responsible for the intrinsic noise [KBPT09].

The partitioning of the reactions into stochastic and continuous ones is carried out using a heuristic approach (see Section 6.2.5). However, a better justification for the partitioning could be given. For instance, the fast processes can be regarded as processes that could be described by a quasi (or pseudo) steady state approach, assuming that they reach equilibrium rapidly. In other words, they could be better described by setting the corresponding ODEs to zero and solving for the high molecular species. In contrast, continuous dynamics could be seen as more appropriate for abundant molecules whose concentrations display a small coefficient of variation, and stochastic dynamics for those molecules evolving at low copy number.

The presented model could be viewed as a sub-net in a bigger network of reactions (e.g., modelling budding yeast cell cycle or Fission yeast cells). Snoopy's hierarchical nodes might simplify such task as they provide an easy means to insert a sub-net in a bigger one.

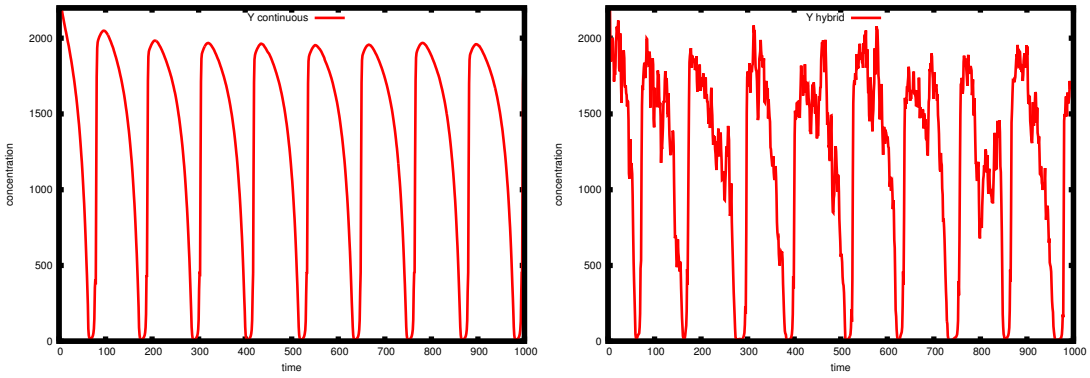
6.3 Circadian Oscillation

In some organisms, there is a control mechanism which is responsible for ensuring a periodic oscillation of certain molecular species [HL07]. This phenomenon is known as circadian rhythm and it can be found in many organisms (e.g., Drosophila).

Maintaining the period and amplitude of a temporal oscillation is an important factor of the evolutionary fitness of an organism [PZV12]. Moreover, such clock networks share common features in a wide rang of organisms. For example, all networks seem to include an interaction between two types of components: activator (positive elements) and repressor (negative elements) [BL00].

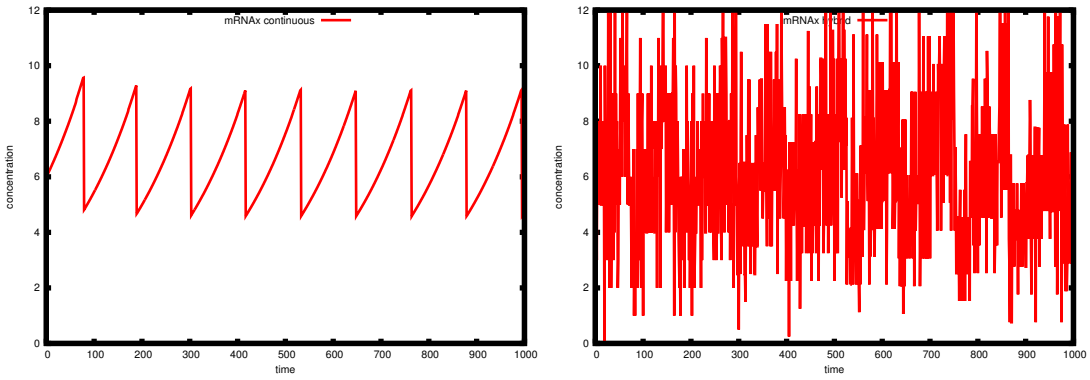
An attractive feature of periodic oscillations in biological organisms is that they can reliably function despite the existence of external noise (such as light or temperature changes) or internal noise due to the fluctuation of species of low numbers of molecules. In the presence of internal noise, oscillations sustain but with period and amplitude that fluctuate with time [BL00].

In this section we study a simple model of circadian oscillation. We run continuous, stochastic, and hybrid simulations and compare the results. Moreover, this case study



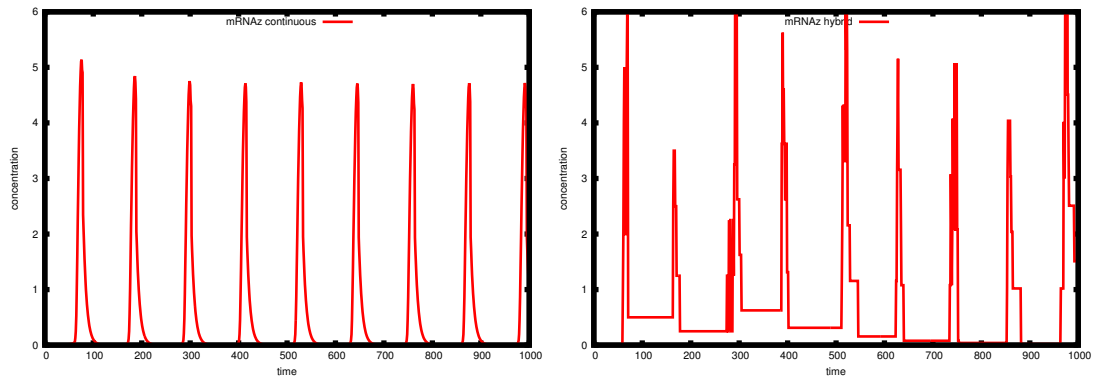
(a) (b)

Figure 6.10: Time course result of the model in Figure 6.6 using Snoopy simulator of the species Y (a) continuous (b) hybrid.



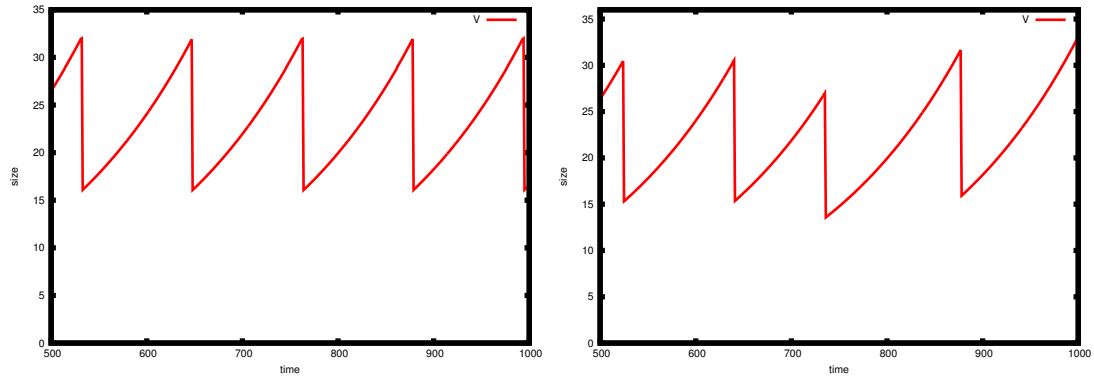
(a) (b)

Figure 6.11: Time course result of $mRNAx$; (a) continuous and (b) hybrid.



(a) (b)

Figure 6.12: Time course result of *mRNAz*; (a) continuos and (b) hybrid.



(a) (b)

Figure 6.13: Continuous and hybrid simulation results of the cellular volume (V); (a) continuous result, and (b) hybrid simulation result. Continuous simulation does not capture the variability of the cellular volume, but hybrid simulation does.

No.	Reaction	Propensity	Rate constant
R_1	$G_1\_active \rightarrow G_1$	$k_1 * G_1\_active$	k_1=50.0
R_2	$G_1 + A \rightarrow G_1\_active$	$k_2 * G_1 * A$	k_2= 1.0
R_3	$G_2\_active \rightarrow G_2$	$k_3 * G_2\_active$	k_3= 100.0
R_4	$G_2 + A \rightarrow G_2\_active$	$k_4 * G_2 * A$	k_4= 1.0
R_5	$\phi \rightarrow mRNA\_G_2$	$k_5 * G_1\_active$	k_5= 500.0
R_6	$\phi \rightarrow mRNA\_G_2$	$k_6 * G_1$	k_6= 50.0
R_7	$mRNA\_G_1 \rightarrow \phi$	$k_7 * mRNA\_G_1$	k_7= 10.0
R_8	$\phi \rightarrow A$	$k_8 * mRNA\_G_1$	k_8= 50.0
R_9	$\phi \rightarrow A$	$k_9 * G_1\_active$	k_9= 50.0
R_{10}	$\phi \rightarrow A$	$k_{10} * G_2\_active$	k_{10}= 100.0
R_{11}	$A \rightarrow \phi$	$k_{11} * A$	k_{11}= 1.0
R_{12}	$R + A \rightarrow A\_R$	$k_{12} * A * R$	k_{12}= 2.0
R_{13}	$\phi \rightarrow mRNA\_G_2$	$k_{13} * G_2\_active$	k_{13}= 50.0
R_{14}	$\phi \rightarrow mRNA\_G_2$	$k_{14} * G_2$	k_{14}= 0.01
R_{15}	$mRNA\_G_2 \rightarrow \phi$	$k_{15} * G_2 * mRNA\_G_2$	k_{15}=0.5
R_{16}	$\phi \rightarrow R$	$k_{16} * mRNA\_G_2$	k_{16}= 5.0
R_{17}	$R \rightarrow \phi$	$k_{17} * R$	k_{17}= 0.2
R_{18}	$A\_R \rightarrow R$	$k_{18} * A\_R$	k_{18}= 1.0

Table 6.2: Reaction set of the circadian oscillation model. The $\mathcal{GHPN}_{bio}$ representations of these reactions are given in Figure 6.14

is a prime example to show how computational steering can be applied to a biological example. Using computational steering, we change some key parameters and monitor the effects of such changes on the period and amplitude of the oscillation. Finally a coloured version of this model is created to show some different performance measures related to our implementation.

6.3.1 Model Overview

We consider a simple model that demonstrates this phenomenon. This model does not produce the details of any biological organism, but it is useful to study oscillation mechanisms in general. It consists of two genes which are denoted by G_1 and G_2. The model includes also one activator and one repressor which are denoted by 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}. The activator A binds to the A and R promotors and subsequently increases their transcription rates. Table 6.2 lists the model reactions, rates, and parameters [JHNS02]. The Petri net in Figure 6.14 contains 9 places, 18

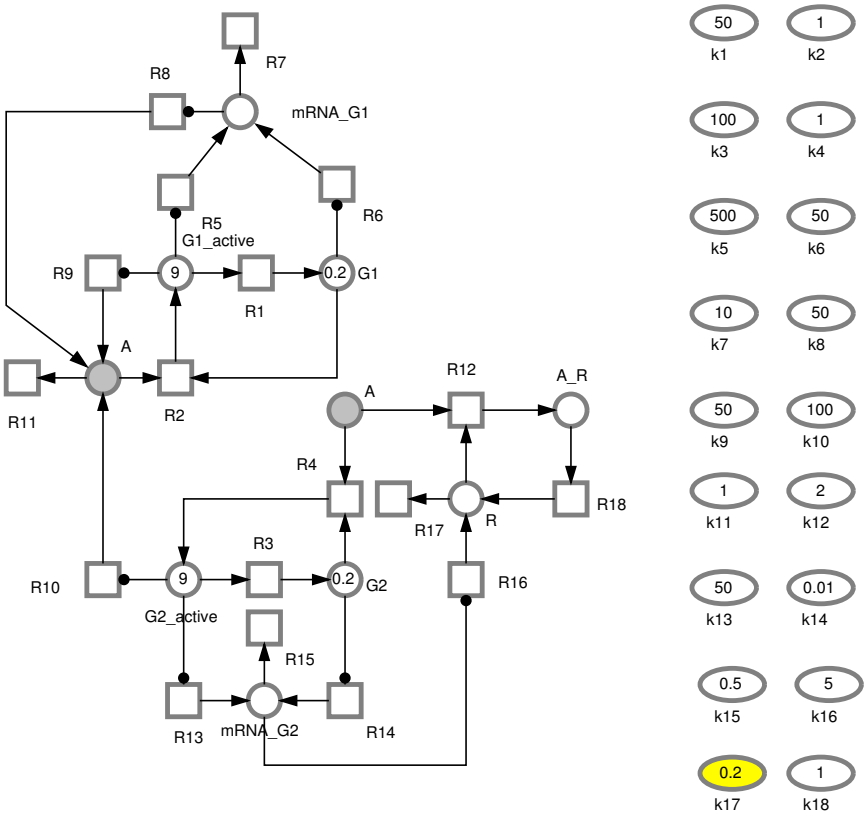


Figure 6.14: A $\mathcal{GHPN}_{bio}$ representation of the circadian oscillation model (Table 6.2). The parameter *k17*, highlighted in yellow, is a key parameter in this model. Transitions are initially represented as continuous, then they are partitioned dynamically into stochastic and continuous ones.

transitions, and 35 arcs.

6.3.2 Simulation Results

Figure 6.15 gives time course simulation results of the $\mathcal{GHPN}_{bio}$ model in Figure 6.14. Using the parameter values given in Figure 6.14, continuous simulation produces oscillations. However, if the rate constant of reaction R_{17} (i.e., k_{17}) is changed, e.g., from 0.2 to 0.08, the continuous simulation fails to produce the desired oscillation [JHNS02, HL07].

It is shown in [JHNS02] that stochastic simulation can still produce the expected oscillation even if there are reactions with low rates.

Hybrid simulation can also produce such an oscillation of this model when species

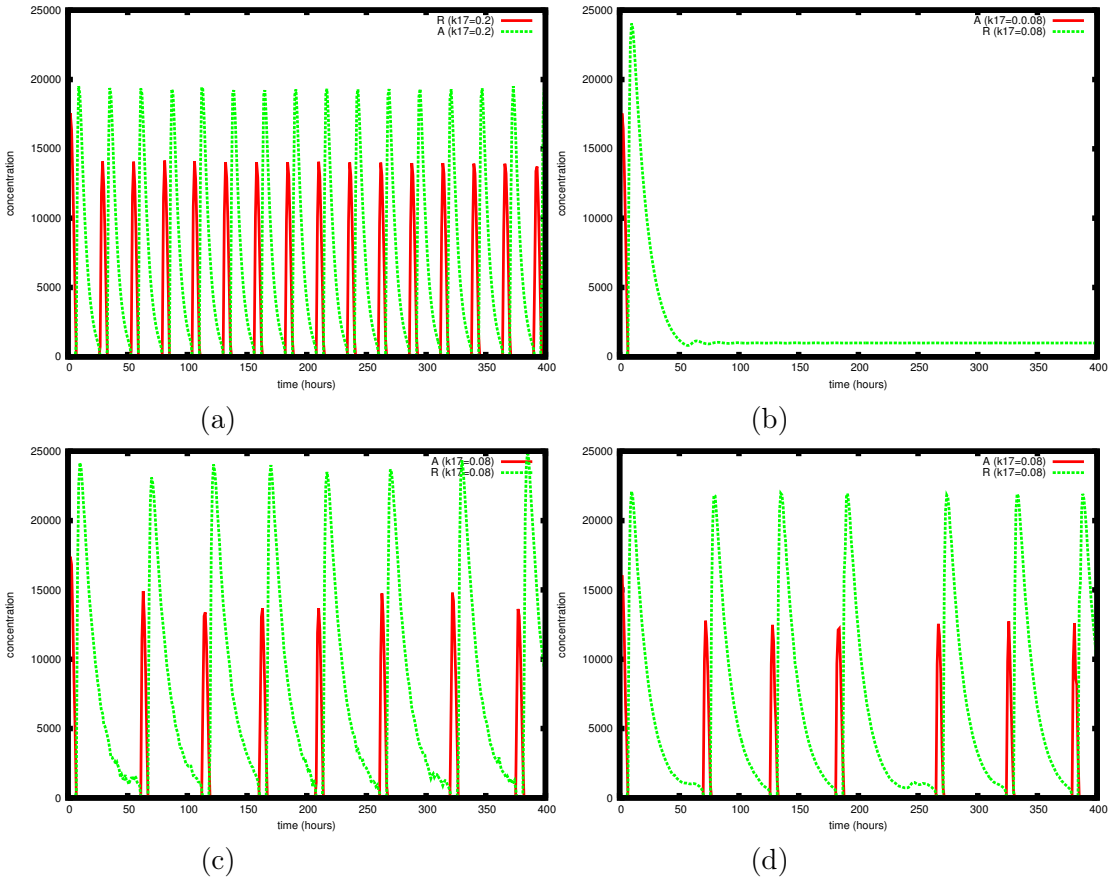


Figure 6.15: Simulation results of repressor (R) and activator (A) proteins of the circadian oscillation model for continuous, stochastic and hybrid methods with two different values of *k17* (0.2 and 0.08). (a) Continuous simulation produces sustained oscillation when *k17*=0.2, (b) but it fails when *k17*=0.08. (c) Contrary, stochastic simulation still produces oscillation for small parameter values (*k17*=0.08), however it is computationally more expensive. (d) Hybrid simulation is also able to produce oscillation when *k17*=0.08, but with substantial runtime improvement.

with low numbers of molecules or reactions of low rates are simulated stochastically, while the others are simulated continuously. However, static partitioning of the Petri net into continuous and stochastic parts will substantially slow down the simulation, since the propensity values are changing during the simulation due to the oscillation. Indeed, this model is well-suited to motivate the dynamic partitioning procedure.

Thus we opted to dynamically partition the model into fast and slow parts during the simulation. Figure 6.15 shows the simulation result when the Petri net in Figure 6.14 is simulated using continuous, stochastic, and hybrid methods with two different values of *k17* (0.2 and 0.08). The parameters in the partitioning algorithm are set to make a trade off between accuracy and speed.

6.3.3 Online Steering of the Model Parameters

Now, we consider the problem of dynamically changing the model parameters. The simulation results presented in Figure 6.15 were produced using the batch mode. That means to check how the model reacts to different values of k_{17}, we have to repeat the simulation experiment each time from the beginning. In this section we perform such changes online while the simulation is in progress.

In Figure 6.16a the simulator is allowed to run without any intervention from the user side with a value of k_{17} equals to 0.2. Thus the oscillation period and the amplitude are constant during the overall simulation time.

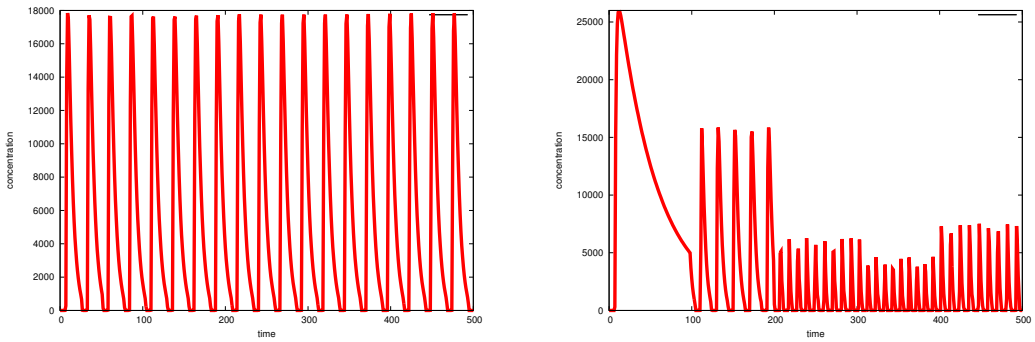
In Figure 6.16b, the continuous simulator is intervened four times by changing the value of k_{17}. During the time period 0-100, the simulator uses a value of 0.02. The system does not produce any oscillation. However, for the rest of the simulation time, the model produces oscillation, but with different periods and amplitudes.

For example, during the time period from 100 to 200, the simulator uses the value of k_{17}=0.3 supplied by the user to execute the model. Increasing the value of k_{17} to 3, the oscillation period as well as the amplitude are decreased. Similar behaviour also occurs during the subsequent changes of k_{17} from 2 to 3 and from 3 to 4 during the time periods 300 - 400 and 400 - 500, respectively.

The advantage of such method of simulation in the context of systems biology is to permit users to experiment with different parameter sets. This could help in solving one of the current challenges of kinetic modelling of biochemical reaction networks, namely parameter estimation.

Computational steering does not play a big role for simple models where there is no big difference, in terms of runtime, of either restarting the simulation from scratch or to continue producing the model dynamics from the current simulation time, but with different parameter values. However, for substantially time-consuming models, it has a big influence on the overall simulation time.

The latter model sizes are typical models in systems biology in the future, since the systems biology's focus is changing from studying and analysing isolated processes to



(a) (b)

Figure 6.16: Simulation results of the circadian oscillation model. (a) k_{17}=0.2 is used throughout the whole simulation time. (b) Different values; of k_{17} are used. k_{17}=0.02 for 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 k_{17} are set to 0.3, 2, 3, 4, respectively.

much bigger views of holistic biological systems.

6.3.4 Coloured Model

In this section we consider a bigger (in size) and scalable version of the continuous Petri net model in Figure 6.14. The coloured model presented here can be easily extended to include more than two genes. We use this example to measure the performance of the implementation of the computational steering framework which has been presented in Chapter 5.

The following declarations are added to the continuous Petri net in Figure 6.14 to get the coloured version. For more details see [Liu12].

```
constant int N=10;
colorset CS=int with 1-N;
variable x:CS;
```

We assign to each place in Figure 6.14 the colour set CS, while the variable x is assigned to the arcs.. Increasing N implies increasing the number of colors in the net which subsequently increases (in the unfold version) dramatically the net size. For the sake of completeness, the coloured model is given in Figure 6.17.

Table 6.3 gives the runtime (in seconds) for different values of N ranging from 10 to 50,000 colours. The communication between the client and the server took place through a LAN connection. The runtimes are computed using the default model view (see Section 5.3.4). We can conclude from this table that transferring the whole result

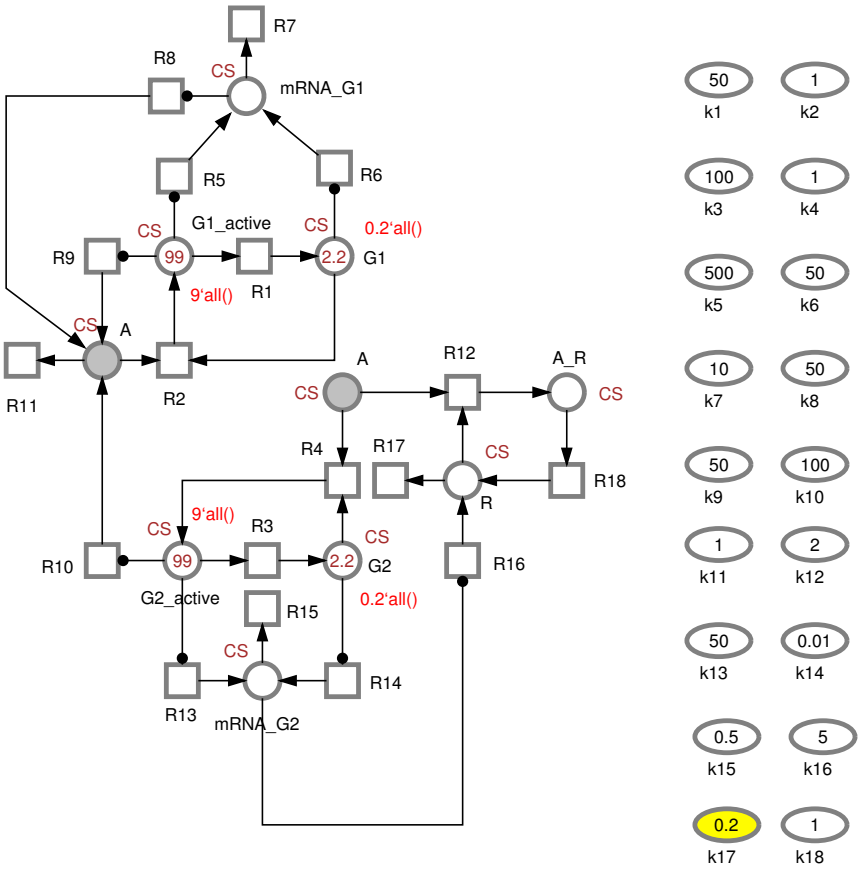


Figure 6.17: A coloured version of the circadian oscillation model in Figure 6.14.

N	Number of places	Number of transitions	Number of arcs	Communicating model specification	Communicating result matrix
10	99	198	385	0.359	0.269
100	909	1,818	1,529	0.468	0.405
1,000	909	18,018	35,035	1.497	1.888
10,000	90,009	180,018	350,035	9.485	10.345
50,000	450,009	900,018	1,750,035	40.648	41.65

Table 6.3: Runtime (in seconds) of executing the coloured model using the steering framework.

matrix from the server to the client might need some time, particularly for larger models. Therefore communicating only parts of the matrix, as it has been discussed in Section 5.6.4, will be useful.

6.4 Discussion

In this section, we discuss and analyse the runtime behaviour of the simulation algorithm applied to four biological models which have been presented in Chapter 4 and 6. Two different comparisons are conducted. First, we compare the three different simulation approaches: continuous, stochastic, and hybrid. Next, we compare the continuous and hybrid simulation runtimes when different types of ODE solvers are used. Tables 6.4 and 6.5 list the runtimes of the selected models. The simulation was performed using, a MAC Pro 8×2.2GHz, 8GB RAM. Please note that the stochastic simulation runtime has not been computed for the eukaryotic cell cycle model, as Snoopy does not support any SPN primitives to simulate the continuous cell growth.

Biological models vary in size and scales which explains the need for different tools to deal with such diversities. To illustrate this point consider Table 6.4 which compares continuous, stochastic, and hybrid simulation in terms of the runtimes required for the four different biological models. These data were produced with our implementation of the three simulation approaches (stochastic, continuous, and hybrid) using Snoopy. The runtimes are computed based on a single run for continuous simulation and 10^4 runs for the stochastic and hybrid setting.

From this table we can conclude that there is no single optimal method (in terms of accuracy and speed) which always performs best for all models. For instance, simulating the Goutsias model using the stochastic approach will be very slow. However, it could capture the true model behaviour. Hybrid simulation saves substantial amount of work as it simulates only a subset of the transitions in a discrete way, while the others are simulated continuously. For the Goutsias model, dynamic partitioning is more ex-

	Continuous	Stochastic	Hybrid (static)	Hybrid (dynamic)
Oscillator	0.197	65,342.273	53,232.862	28,689.125
T7 Phage	0.002	33,567.379	3,149.204	2,249.755
Goutsias	0.002	1,273.084	72.707	75.631
Cell Cycle	2.642	-	194.629*	-

– have not been performed
* only single run for the hybrid simulation is used

Table 6.4: Comparison of continuous, stochastic and hybrid simulation runtimes (in seconds) for four $\mathcal{GHPN}_{bio}$ models using multi-step ODE solver.

	Continuous	Stochastic	Hybrid (static)	Hybrid (dynamic)
Oscillator	1.029	65,342.273	to be done	to be done
T7 Phage	0.002	33,567.379	264.083	210.575
Goutsias	0.005	1,273.084	89. 387	77.205
Cell Cycle	11.578	-	102.467*	-

– have not been performed
* only single run for the hybrid simulation is used

Table 6.5: Comparison of continuous, stochastic and hybrid simulation runtimes (in seconds) for four $\mathcal{GHPN}_{bio}$ models using single-step ODE solver.

pensive than the static one as the former introduces additional overhead without giving any improvement. Indeed the reactions of the Goutsias model are easily separable into two scales.

Contrary, simulation with dynamic partitioning of the oscillator model is faster than using static partitioning. This happens because transition rates change over time from low to high and vice versa, as it can be noticed in the oscillations in Figure 6.15.

Now we return to the point which has been discussed in Section 4.4.2. When a multi-step solver is used in Table 6.4 to perform the continuous simulation of the cell cycle model, it exhibits high performance compared to simulating the same model using a single-step method. However, using the same ODEs solver in the hybrid, simulation the runtime behaviour is reversed.

Such change in the performance is due to the many discontinuities that occur during the hybrid simulation. Such discontinuities require the reinitialisation of the ODE solver. Hence, the performance of single-step ODE solver is not affected by such reinitialisation.

Moreover, this point becomes more obvious in the runtime of simulating the T7 phage model. The continuous part of this model does not exhibit stiffness, therefore a single step explicit ODE solver is well-suited to execute the model behaviour using the hybrid simulator.

For the Goutsias model, we do not have such a difference of the simulator performance. The number of reactions that are simulated stochastically of this model is small. Therefore, the number of discontinuities is limited. Hence, multi-step solvers outperform the single-step one in this case.

As a conclusion, selecting an appropriate ODE solver is important for the performance of the hybrid simulator algorithm. However, such a choice, as in the pure deterministic simulation, is model-dependent.

6.5 Conclusions

In this chapter we have presented three case studies to illustrate different aspects of the two introduced tools in Chapters 4 and 5. Besides the runtime behaviours of these models, the Goutsias model is additionally analysed comparing single-step ODEs solvers with the multi-step ones.

As a conclusion for these case studies, stochastic simulation is always slower, but it is very accurate compared to continuous and hybrid simulation. This motivates us to provide with $\mathcal{GHPN}_{bio}$ a unified framework to simulate one and the same model using different simulation methods which gives system biologists a tool to easily try different methods and to choose the most suitable one.

Moreover, computational steering can be used to simulate the presented case studies in a more convenient way than the framework currently used in kinetic modelling software.

7 Conclusions and Future Work

7.1 Conclusions

During the course of this study, we have presented two different, yet related contributions to an exciting and interdisciplinary field *systems biology*. The two contributions are: the definition as well as the implementation of a new Petri net class, Generalised Hybrid Petri Nets, and the development of a computational steering framework for the collaborative, distributed and interactive simulation of biochemical reaction networks. The mathematical and computational tools, introduced in this work, contribute and help systems biologists in the ongoing effort of understanding biological phenomenon at the system level. Moreover, the two contributions are useful in modelling other technical systems. In the following section we summarise our contributions, while in the next one we discuss possible future extensions of our work.

7.1.1 Generalised Hybrid Petri Nets

With the advance of computational modelling of biochemical reaction networks, hybrid simulation plays an important role in producing the dynamics of computational biological models, particularly for models which require the integration of stochastic and continuous semantics (for some examples see Chapter 6). Therefore, there is a need for a Petri net tool that covers a wide range of modelling capabilities and supports stochastic as well as continuous semantics in one and the same model. Moreover, the implementation of such a tool is of paramount importance to systems biologists. Thus, we have developed Generalised Hybrid Petri Nets.

$\mathcal{GHPN}_{bio}$ provide the full interplay between stochastic and continuous transitions through the monitoring of changes in stochastic transitions' rates during the simulation of continuous transitions. Moreover, $\mathcal{GHPN}_{bio}$ compose a wide range of transition types (e.g., stochastic, continuous, immediate, and scheduled transitions) which renders them possible to model different reaction types.

A special application of $\mathcal{GHPN}_{bio}$ is the representation and simulation of stiff biochemical networks, where reactions that occur frequently are represented and simulated through continuous transitions, while reactions that occur infrequently are modelled using stochastic transitions. Switch-like and fixed time delay semantics can be modelled through immediate and deterministically timed transitions, respectively.

For the requirements of simulating stiff biochemical reactions, we have implemented

two simulation methods: simulation using static or dynamic partitioning. In the former, reactions are partitioned only once during the entire simulation time, namely at the initialisation step of the simulator, while in the latter, partitioning is repeated whenever there is a need to reconsider partitioning. The computational overhead due to repeating the partitioning can gain more simulation speed in some cases (e.g., models which exhibit oscillations).

Furthermore, the implementation of $\mathcal{GHPN}_{bio}$ is a platform-independent and it is available free of charge for academic use.

7.1.2 Computational Steering Framework

Currently available software tools that simulate biochemical reaction networks follow the batch simulation approach. Therefore they look like a black-box from the user point of view, when they perform the simulation. Besides, they position the users away from their simulation. However, users would like to interact with the simulation and depict errors and incorrect results as soon as possible. Furthermore, collaboration between different users in investigating the same model is helpful as it promotes the sharing of knowledge between different users of different backgrounds.

With this motivations, we have developed a computational steering framework for the interactive simulation of biochemical reaction networks. The main components of the proposed framework are: the steering server – responsible for managing the running models and the collaboration and co-ordination between connected users, the steering graphical user interface – the user entry point to run the interactive simulation, the steering application programming interface (API) – carrying out the communication between the different framework components, and the internal and external simulators – produce the model dynamic through the simulation of the Petri nets.

The introduced framework adopts Petri nets as a formal and graphical modelling language of biochemical reaction networks. It supports various helpful features for the modelling and analysis of reaction networks, e.g., intuitive and understandable representation of reaction networks, 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.

7.1.3 Case Studies

Three case studies have been discussed in Chapter 6 to demonstrate the operations of $\mathcal{GHPN}_{bio}$ and the computational steering framework. These case studies have different scales and expose various behaviours.

The T7 phage model shows that continuous and hybrid simulations yield different suggestions. The three-place model can accurately be simulated through stochastic and hybrid simulations rather than the continuous one.

The Eukaryotic cell cycle model demonstrates most of the $\mathcal{GHPN}_{bio}$ elements. The continuous version of this model cannot capture the variability of the cellular volume. Moreover, marking-dependent arc weights are better explained by this model.

Finally the circadian oscillation model discusses different aspect of $\mathcal{GHPN}_{bio}$ namely dynamic partitioning. While the markings are oscillating, transition rates are also oscillating. This makes the dynamic partitioning useful to deal with such behaviour. Furthermore, this model is used to demonstrate the practical use of computational steering framework.

7.2 Outlook

The work presented in this thesis can be extended in many directions. We classify the extensions into those that relate to $\mathcal{GHPN}_{bio}$ and those that relate to the computational steering framework.

7.2.1 Extending Generalised Hybrid Petri Nets

With respect to expressiveness, $\mathcal{GHPN}_{bio}$ support a rich choice of places, transitions, and arcs which make $\mathcal{GHPN}_{bio}$ a super class of the different hybrid Petri net classes discussed in Chapters 3 and 4. Therefore, future developments of $\mathcal{GHPN}_{bio}$ are more concerned with efficiently executing the constructed models. In this context potential extensions are: implementing other stochastic algorithms, enhancing the dynamic partitioning algorithm, implementing the animation of $\mathcal{GHPN}_{bio}$, the introduction of weakly-enabled semantics for continuous transitions, coloured and distributed simulation of coloured hybrid Petri nets, and the exploration of further biological case studies.

Implementing Other SSA Algorithms Snoopy's hybrid simulator currently supports the implementation of the direct method for simulating stochastic transitions. Another intended further extension is to support more than one stochastic simulator within Snoopy. This can easily be achieved due to the modular implementation of the simulator library.

Efficient Dynamic Partitioning Algorithm Although the dynamic partitioning algorithm which has been presented in Chapter 4 is sufficient for small and medium models, it could be improved to keep the dynamic simulation more user friendly for big models. Our suggestion in this direction is to view the dynamic partitioning process as an optimisation problem and use for example linear programming to solve it.

Animation for $\mathcal{GHPN}_{bio}$ Animation of discrete Petri nets has been proved to be useful for the understanding of the Petri net model as it provides execution for the

single firing of transitions. A similar approach can be developed for hybrid Petri nets.

Weakly enabled Continuous Transitions The semantics of $\mathcal{GHPN}_{bio}$, which has been defined in Chapter 4, is mainly motivated by biological modelling. Therefore, conflicts between continuous transitions are not considered (see Section 4.2.6), since transition rates are defined according to specific rules (e.g., Mass Action). That is continuous transitions are always strongly enabled (i.e., continuous transitions can fire all the time using their maximal firing speeds) [DA10]. However, if $\mathcal{GHPN}_{bio}$ are intended to simulate other technical systems where conflicts between continuous transitions might take place, weakly enabled continuous transitions will also be required [DA87]. Such type of semantics should take into account the adjustment of continuous transitions' rates (transition instantaneous speeds) so that negative values do not occur. Nevertheless, this semantics will affect the simulation's efficiency and hence should be used with care when simulating biological models.

Coloured-level Simulation As an extension of the hybrid Petri net class which has been introduced in this thesis, Fei Liu defined in [Liu12] coloured hybrid Petri nets. However, the simulation of this high-level class is done via unfolding of the coloured version into a low-level one. Therefore, a potentially time-consuming process is required before executing the coloured model. Additionally, some features cannot be implemented on the modelling level due to the required unfolding. A better and straightforward strategy might be to execute the simulation directly on the coloured level to overcome the aforemetioned limitations.

Fine-grained Distributed Simulation of Coloured Hybrid Petri Nets Unfolding of coloured Petri nets results in increasingly large networks of low-level nets. The simulation of such Petri nets tends to be very time consuming. However, under some circumstances the unfolded Petri nets consist of repeated components that can be partitioned and simulated separately and simultaneously on different machines. To accomplish this task two issues need to be solved: (i) partitioning the whole net into smaller subnets, and (ii) adjusting the simulation algorithm to deal with such smaller subnets. For the latter issue, Algorithm 4.1, which has been presented in Chapter 4, requires to be extended in order to cope with such scenario.

Exploring More Biological Case Studies The case studies, which have been presented in Chapter 6, are chosen to illustrate different aspects of the hybrid approach. Further complex biological examples can be modelled using $\mathcal{GHPN}_{bio}$.

7.2.2 Extending the Computational Steering Framework

The computational steering framework could be extended from different perspectives to provide more features to the user. Further options are:

Condition-based steering Condition-based steering means that the user defines some conditions and a corresponding response before the simulation starts. Later, during the simulation, the simulator checks the conditions and if one or more of them hold, the corresponding actions are taken. It is similar to scheduled transitions which have been presented in [HGD08]. However, condition-based steering will give more flexibility to include other rules which cannot be implemented on the Petri net level. As an example, consider the problem of determining the simulation time point when no further stochastic event takes place.

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

Steering the Model Structure The discussion in Chapter 5 is centred about the steering of the model parameters and current markings. This has nothing to do with the structure. Permitting the user to change the Petri net model structure during the simulation will help to implement many biological models which require such option.

Wrapper APIs: to integrate legacy code written in other languages The steering API is completely written in C++ and can be run under many different operating systems. However, some computational codes were developed using other programming languages, e.g., FORTRAN. Writing interface functions to call the API library from these languages will facilitate the adaptation of our framework for those simulation codes.

Web and Mobile Steering GUI Finally, a steering GUI client can be written as a web or mobile application to keep systems biologists closely connected with their remotely running simulation.

Bibliography

[ABF+10] Atanasov, A.; Bungartz, H.; Frisch, J.; Mehl, M.; Mundani, R.; Rank, E. ; Treeck, C.: Computational steering of complex flow simulations. In: Wagner, Siegfried; Steinmetz, Matthias; Bode, Arndt; Müller, Markus M. (Eds.): *Proceedings of High Performance Computing in Science and Engineering, Garching/Munich 2009*. Springer Berlin Heidelberg, 2010, pp. 63 – 74 {91}

[ACT+05] Alfonsi, A.; Cancès, E.; Turinici, G.; Ventura, B. ; Huisinga, W.: Adaptive simulation of hybrid stochastic and deterministic models for biochemical systems. In: *ESAIM: Proc.* 14 (2005), pp. 1–13 {29, 42, 43, 45, 74, 77, 116}

[AD98] Alla, H.; David, R.: Continuous and hybrid Petri nets. In: *J. Circ. Syst. Comp.* 8 (1998), Nr. 1, pp. 159 –188 {53}

[Ani02] Anirudh, M.: *Real-time visualization of aerospace simulations using computational steering and beowulf clusters*, The Pennsylvania State University, PhD thesis, 2002 {23, 91}

[ASM+11] Ahern, S.; Shoshani, A.; Ma, K.; Choudhary, A. ; Chritchlow, T.: Scientic discovery at the exascale / Report from the DOE ASCR 2011 Workshop on Exascale Data Management, Analysis, and Visualization. 2011. – Technical Report {9}

[AVS12] *Advanced Visual Systems website*. `http://www.avs.com/`. 2012. – Accessed: 1/24/2012 {11}

[BGHO08] Breitling, R.; Gilbert, D.; Heiner, M. ; Orton, R.: A structured approach for the engineering of biochemical network models, illustrated for signalling pathways. In: *Brief Bioinform* 9 (2008), Nr. 5, pp. 404 – 421 {31}

[BGM00] Balduzzi, F.; Guia, A. ; Menga, G.: First-order hybrid Petri nets: a model for optimization and control. In: *IEEE Trans. on Robotics and Automation* 16 (2000), Nr. 4, pp. 382–399 {55, 60}

[BJBH93] Brunner, J.; Jablonowski, D.; Bliss, B. ; Haber, R.: VASE: the visualization and application steering environment. In: *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 1993, pp. 560–569 {17, 26}

[BL00] Barkai, N.; Leibler, S.: Biological rhythms: circadian clocks limited by noise. In: *Nature* 403 (2000), pp. 267–268 {129}

[BMS+12] 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. In: *Procedia Computer Science* 9 (2012), pp. 1149 – 1158 {9}

[CCCN+04] Chen, K.; Calzone, L.; Csikasz-Nagy, A.; Cross, F.; Novak, B. ; Tyson, J.: Integrative analysis of cell cycle control in budding yeast. In: *Mol. Biol. Cel* 5 (2004), Nr. 8, pp. 3841–3862 {121, 122}

[CFH+05] Coveney, P.; Fabritiis, G.; Harvey, M.; Pickles, S. ; Porter, A. *On steering coupled models*. e-Science All Hands Meeting. 2005 {91}

[CGP05] Cao, Y.; Gillespie, D. ; Petzold, L.: Avoiding negative populations in explicit Poisson tau-leaping. In: *J. Chem. Phys.* 123 (2005), Nr. 054104 {39, 40}

[CGP06] Cao, Y.; Gillespie, Daniel ; Petzold, L.: Efficient step size selection for the tau-leaping simulation method. In: *J. Chem. Phys.* 124 (2006), Nr. 044109 {39, 40}

[CGP07] Cao, Y.; Gillespie, D. ; Petzold, L.: Adaptive explicit-implicit tau-leaping method with automatic tau selection. In: *J. Chem. Phys* 126 (2007), Nr. 22, pp. 224101 {41}

[CLP04] Cao, Y.; Li, H. ; Petzold, L.: Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. In: *J. Chem. Phys* 121 (2004), Nr. 9, pp. 4059 {41}

[CP05] Cao, Y.; Petzold, L.: Trapezoidal tau-lepaing formula for the stochastic simulation of biochemical systems. In: *Proceedings of Foundations of Systems Biology in Engineering*, 2005, pp. 149–152 {41}

[CW01] Chatzinikos, F.; Wright, H.: Computational steering by direct image manipulation. In: *Proceedings of the Vision Modeling and Visualization Conference 2001*, Aka GmbH, 2001, pp. 455–462 {13}

[DA87] David, R.; Alla, H.: Continuous Petri nets. In: *Proceedings of the 8th Eurpean Workshop on Application and Theory of Petri Nets.* Saragossa, Spain, 1987, pp. 275–294 {51}

[DA10] David, R.; Alla, H.: *Discrete, Continuous, and Hybrid Petri Nets.* Springer, 2010 {2, 52, 53, 60, 61, 66, 67, 71, 73}

[DeF87] DeFanti, T.: Special issue on visualization in scientific computing. In: *Computer Graphics* 21 (1987), Nr. 6 {9}

[DK98] Demongodin, I.; Koussoulas, N.: Differential Petri nets: Representing continuous systems in a discrete-event world. In: *IEEE Trans. Automat. Contr.* 43 (1998), Nr. 4, pp. 573 – 579 {54, 60, 73}

[Dra98] Drath, R.: Hybrid object nets: an object oriented concept for modeling complex hybrid systems. In: *Proceedings of 3rd International Conference on Automation of Mixed Processes: Hybrid Dynamical Systems*, 1998, pp. 437–442 {55}

[DWB+12] Denham, M.; Wendt, K.; Bianchini, G.; Cortés, A. ; Margalef, T.: Dynamic data-driven genetic algorithm for forest fire spread prediction. In: *Journal of Computational Science* 3 (2012), Nr. 5, pp. 398 – 404 {9}

[ERC06] Esnard, A.; Richart, N. ; Coulaud, O.: A steering environment for online parallel visualization of legacy parallel simulations. In: *Proceedings of the 10th International Symposium on Distributed Simulation and Real-Time Applications.* Torremolinos, Malaga, Spain: IEEE Press, 2006, pp. 7–14 {25}

[Feh93] Fehling, Rainer: A concept of hierarchical Petri nets with building blocks. In: Rozenberg, Grzegorz (Ed.): *Advances in Petri Nets 1993, Lecture Notes in Computer Science* Volume 674. Springer Berlin / Heidelberg, 1993, pp. 148–168 {55, 56}

[FMJ+08] Funahashi, A.; Matsuoka, Y.; Jouraku, A.; Morohashi, M.; Kikuchi, N. ; Kitano, H.: CellDesigner 3.5: A versatile modeling tool for biochemical networks. In: *Proceedings of the IEEE* (2008), Nr. 8, pp. 1254 – 1265 {1, 4}

[GB00] Gibson, M.; Bruck, J.: Exact stochastic simulation of chemical systems with many species and many channels. In: *J. Phys. Chem.* 105 (2000), pp. 1876 – 89 {31, 37, 38}

[GCPS06] Griffith, M.; Courtney, T.; Peccoud, J. ; Sanders, W.: Dynamic partitioning for hybrid simulation of the bistable HIV-1 transactivation network. In: *Bioinformatics* 22 (2006), Nr. 22, pp. 2782–2789 {29, 42, 43, 45, 74, 77}

[GH06] Gilbert, D.; Heiner, M.: From Petri nets to differential equations - an integrative approach for biochemical network analysis. In: Donatelli, Susanna; Thiagarajan, P. (Eds.): *proceedings of Petri Nets and Other Models of Concurrency - ICATPN 2006, Lecture Notes in Computer Science* Volume 4024. Springer Berlin / Heidelberg, 2006, pp. 181–200 {2, 53, 66}

[GHL07] Gilbert, D.; Heiner, M. ; Lehrack, S.: A unifying framework for modelling and analysing biochemical pathways using Petri nets. In: Calder, Muffy; Gilmore, Stephen (Eds.): *Computational Methods in Systems Biology, Lecture Notes in Computer Science* Volume 4695. Springer Berlin / Heidelberg, 2007, pp. 200–216 {2, 49, 81, 82}

[Gib00] Gibson, M.: *Computational methods for stochastic biological systems.* Pasadena, California, California institute of Technology, PhD thesis, 2000 {38}

[Gil76] Gillespie, D.: A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. In: *J. Comput. Phys.* 22 (1976), Nr. 4, pp. 403 – 434 {5, 29, 30, 34, 35, 36, 37, 79, 122}

[Gil77] Gillespie, D.: Exact stochastic simulation of coupled chemical reactions. In: *J. Phys. Chem.* 81 (1977), Nr. 25, pp. 2340–2361 {5, 29, 34, 36}

[Gil91] Gillespie, D.: *Markov processes: an introduction for physical scientists.* Academic Press, October 1991 {45}

[Gil01] Gillespie, D.: Approximate accelerated stochastic simulation of chemically reacting system. In: *J. Chem. Phys.* 115 (2001), pp. 1716–1733 {31, 39, 40}

[Gil07] Gillespie, D.: Stochastic simulation of chemical kinetics. In: *Annu Rev Phys Chem.* 58 (2007), Nr. 1, pp. 35–55 {5, 29, 30, 31, 34, 35, 37, 39, 41, 42, 121}

[GKP97] Geist, G.; Kohl, J. ; Papadopoulos, P.: CUMULVS: providing fault-tolerance, visualization and steering of parallel applications. In: *International Journal of High Performance Computing Applications* 11 (1997), Nr. 3, pp. 224–236 {2, 10, 21}

[Gou05] Goutsias, J.: Quasiequilibrium approximation of fast reaction kinetics in stochastic biochemical systems. In: *J. Chem. Phys.* 122 (2005), Nr. 184102, pp. 1–15 {41, 85, 88}

[GP03] Gillespie, D.; Petzold, L.: Improved leap-size selection for accelerated stochastic simulation. In: *J. Chem. Phys.* 119 (2003), pp. 8229–8234 {39, 40}

[HBG+05] Hindmarsh, A.; Brown, P.; Grant, K.; Lee, S.; Serban, R.; Shumaker, D. ; Woodward, C.: SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. In: *ACM Trans. Math. Softw.* 31 (2005), September, pp. 363–396. {79, 81, 101}

[HEA07] Herajy, M.; EL-Desouky, B. ; Atta, A.: A distributed computational steering environment for electronic learning applications. In: *proceedings of 3rd ACM International Conference on Intelligent Computing and Information Systems* Volume 3, 2007, pp. 208–213 {25}

[HGD08] Heiner, M.; Gilbert, D. ; Donaldson, R.: Petri nets for systems and synthetic biology. In: Bernardo, Marco; Degano, Pierpaolo; Zavattaro, Gianluigi (Eds.): *Formal Methods for Computational Systems Biology, Lecture Notes in Computer Science* Volume 5016. Springer Berlin / Heidelberg, 2008, pp. 215–264 {59, 96, 144}

[HH11] Herajy, M.; Heiner, M.: Hybrid representation and simulation of stiff biochemical networks through generalised hybrid Petri nets / Brandenburg University of Technology Cottbus, Dept. of CS. 2011 (02/2011). – Technical Report {2}

[HH12a] Herajy, M.; Heiner, M.: Hybrid representation and simulation of stiff biochemical networks. In: *Nonlinear Analysis: Hybrid Systems* 6 (2012), Nr. 4, pp. 942–959 {2, 57, 61}

[HH12b] Herajy, M.; Heiner, M.: Towards a computational steering and Petri nets framework for the modelling of biochemical reaction networks. In: Popova-Zeugmann, Louchka (Eds.): *In Proc. of 21th international Workshop on Concurrency, Specification and Programming (CS&P 2012)* Volume 21, Humboldt University of Berlin, 2012, pp. 147 – 159 {3, 91}

[HHL+12] Heiner, M.; Herajy, M.; Liu, F.; Rohr, C. ; Schwarick, M.: Snoopy – A Unifying Petri Net Tool. In: Haddad, Serge; Pomello, Lucia (Eds.): *Application and Theory of Petri Nets, Lecture Notes in Computer Science* Volume 7347. Springer Berlin / Heidelberg, 2012, pp. 398–407 {3, 57, 79, 102}

[HK99] Horton, G.; Kowarschik, M.: Discrete-continuous modeling using hybrid stochastic Petri nets. In: *proceedings of European Simulation Symposium*, SCS Publishing House., 1999 {55, 85}

[HL07] Hellander, A.; Lötstedt, P.: Hybrid method for the chemical master equation. In: *J. Comput. Phys.* 227 (2007), pp. 100–122 {42, 129, 133}

[HLGM09] Heiner, M.; Lehrack, S.; Gilbert, D. ; Marwan, W.: Extended stochastic Petri nets for model-based design of wetlab experiments. In: *Transactions on Computational Systems Biology XI*. Berlin, Heidelberg: Springer, 2009, pp. 138–163 {42, 49, 59}

[HM90] Haber, R; McNabb, D.: *Visualization idioms: a conceptual model for visualization systems*. IEEE Computer Society Press, 1990 (Visualization and Scientific Computing), pp. 74–93 {11}

[HMMW10] Henzinger, T.; Mikeev, L.; Mateescu, M. ; Wolf, V.: Hybrid numerical solution of the chemical master equation. In: *Proceedings of the 8th International Conference on Computational Methods in Systems Biology*. New York, NY, USA: ACM, 2010, pp. 55–65 {35, 88}

[HNW93] Hairer, E.; Nørsett, S. ; Wanner, G.: *Springer Series in Comput. Mathematics*. Volume 8: *solving ordinary differential equations I: nonstiff problems*. Springer-Verlag, 1993 {32}

[HR02] Haseltine, E.; Rawlings, J.: Approximate simulation of coupled fast and slow reactions for stochastic chemical kinetics. In: *J. Chem. Phys.* 117 (2002), Nr. 15, pp. 6959–6969 {29, 31, 42, 45, 74, 123}

[HS12] Herajy, M.; Schwarick, M.: A hybrid Petri net model of the eukaryotic cell cycle. In: Heiner, Monika; Hofestädt, Ralf (Eds.): *Proceedings of the 3rd International Workshop on Biological Processes and Petri Nets (BioPPN)*, CEUR-WS.org, 2012, pp. 29–43 {2}

[HSG+06] 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. In: *Bioinformatics* 22 (2006), pp. 3067–74 {1}

[HW96] Hairer, E.; Wanner, G.: *Springer Series in Comput. Mathematics*. Volume 14: *Solving ordinary differential equations II: stiff and differential-algebraic problems*. Springer-Verlag, 1996 {32}

[Jen95] Jensen, K.: *Coloured Petri nets: basic concepts, analysis methods and practical use*. Volume 2. Springer-Verlag, 1995 {55}

[JHNS02] Jose, J.; Hao, H.; Naama, N. ; Stanislas, S.: Mechanisms of noise-resistance in genetic oscillators. In: *Proceedings of the National Academy of Sciences of the United States of America* 99 (2002), April, Nr. 9, pp. 5988–5992. {132, 133}

[JPH+99] Johnson, C.; Parker, S.; Hansen, C.; Kindlmann, G. ; Livnat, Y.: Interactive simulation and visualization. In: *Computer* 32 (1999), pp. 59–65 {11, 91}

[KBD+94] Kartson, D.; Balbo, G.; Donatelli, S.; Franceschinis, G. ; Conte, G.: *Modelling with generalized stochastic Petri nets*. John Wiley & Sons, Inc., 1994 {49, 66, 71, 72}

[KBPT09] Kar, S.; Baumann, W.; Paul, M. ; Tyson, J.: Exploring the roles of noise in the eukaryotic cell cycle. In: *Proceedings of the National Academy of Sciences of the United States of America* 106 (2009), April, Nr. 16, pp. 6471–6476. – ISSN 1091–6490 {4, 119, 120, 121, 122, 123, 129}

[KGM11] Kitano, H.; Ghosh, S. ; Matsuoka, Y.: Social engineering for virtual 'big science' in systems biology. In: *Nature chemical biology* 7 (2011), Juni, Nr. 6, pp. 323–326 {3, 96}

[Kit02] Kitano, H.: Systems Biology: A Brief Overview. In: *Science* 295 (2002), März, Nr. 5560, pp. 1662–1664 {1, 3}

[KMS04] Kiehl, T.; Mattheyses, R. ; Simmons, M.: Hybrid simulation of cellular behavior. In: *Bioinformatics* 20 (2004), pp. 316–322. {29, 42, 44, 58, 116}

[LCPG08] Li, H.; Cao, Y.; Petzold, L. ; Gillespie, D.: Algorithms and software for stochastic simulation of biochemical reacting systems. In: *Biotechnol. Progr.* 24 (2008), Nr. 1, pp. 56–61 {5, 29, 41}

[LGS+07] Lloyd, S.; Gavaghan, D.; Simpson, A.; Mascord, M.; Seneurine, C.; Williams, G.; Pitt-Francis, J.; Boyd, D.; Randal, D.; Sastry, L.; Nagella, S.; Weeks, K.; Fowler, R.; Hanlon, D.; Handley, J.; de Fabritiis, G.: Integrative Biology — the challenges of developing a collaborative research environment for heart and cancer modelling. In: *Future Generation Computer Systems* 23 (2007), Nr. 3, pp. 457 – 465 {91}

[Liu12] Liu, F.: *Colored Petri Nets for Systems Biology*, Brandenburg University of Technology Cottbus - Computer Science Institute, PhD thesis, 2012 {2, 55, 56, 136}

[LJPS05] Liu, H.; Jiang, L.; Parashar, M. ; Silver, D.: Rule-based visualization in the discover computational steering collaboratory. In: *J. Future Generation Comput. Syst* 21 (2005), pp. 53 – 59 {2, 10, 22}

[LMW96] Liere, R.; Mulder, J. ; Wijk, J.: Computational steering. In: Liddell, Heather; Colbrook, Adrian; Hertzberger, Bob; Sloot, Peter (Eds.): *High-Performance Computing and Networking, Lecture Notes in Computer Science* Volume 1067. Springer Berlin / Heidelberg, 1996, pp. 696–702 {2, 10, 21}

[LPL+12] Liu, Z.; Pu, Y.; Li, F.; Shaffer, C.; Hoops, S.; Tyson, J. ; Cao, Y.: Hybrid modeling and simulation of stochastic effects on progression through the eukaryotic cell cycle. In: *J. Chem. Phys* 136 (2012), Nr. 34105 {123}

[LR12] Lesage, J.; Raffin, B.: A hierarchical component model for large parallel interactive applications. In: *The Journal of Supercomputing* 60 (2012), Nr. 3, pp. 389–409 {9}

[LW97] Liere, R.; Wijk, J.: An environment for computational steering. In: Nielson, G.; Muller, H (Eds.): *Scientific Visualization: Overviews, Methodologies, and techniques.* Computer Society Press, 1997, pp. 89–110 {21}

[MA99] McAdams, H.; Arkin, A.: It's a noisy business! Genetic regulation at the nanomolar scale. In: *Trends in Genetics* 15 (1999), Nr. 2, pp. 65 – 69 {5, 29, 34}

[Mar96] Marksteiner, P.: High-performance computing —an overview. In: *Computer Physics Communications* 97 (1996), 8, Nr. 1–2, pp. 16–35. {24}

[Mat12] *Matlab Website*. `http://www.mathworks.de/products/matlab/`. 2012. – Accessed: 1/22/2012 {16}

[MCN08] Mura, I.; Csikász-Nagy, A.: Stochastic Petri net extension of a yeast cell cycle model. In: *Journal of Theoretical Biology* 254 (2008), Nr. 4, pp. 850 – 860 {119, 121, 122}

[McQ67] McQuarrie, D.: Stochastic approach to chemical kinetics. In: *Journal of Applied Probability* 4 (1967), Nr. 3, pp. 413–478 {34, 35}

[MLP02] 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),*, 2002, pp. 787 – 800 {2, 10, 23}

[MMMP01] Mann, V.; Matossian, V.; Muralidhar, R. ; Parashar, M.: DISCOVER: An environment for Web-based interaction and steering of high-performance scientific applications. In: *Concurrency and Computation: Practice and Experience* 13 (2001), pp. 737–754 {2, 10, 17, 22}

[MNM11] Matsuno, H.; Nagasaki, M. ; Miyano, S.: Hybrid Petri net based modeling for biological pathway simulation. In: *Natural Computing* 10 (2011), September, Nr. 3, pp. 1099–1120. {4}

[MP02a] Mao, G.; Petzold, L.: Efficient integration over discontinuities for differential-algebraic systems,. In: *Computers and Mathematics with Applications* 43 (2002), pp. 65–79 {80, 81}

[MP02b] Muralidhar, R.; Parashar, M.: A distributed object infrastructure for interaction and steering. In: *Concurrency Computat.: Pract. Exper* 15 (2002), pp. 957–977 {12, 17, 22}

[MPC+06] McCollum, J.; Peterson, G.; Cox, C.; M.Simpson ; Samatova, N.: The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior. In: *Computational Biology and Chemistry* 30 (2006), Nr. 1, pp. 39 – 49 {41}

[MRH12] Marwan, W.; Rohr, C. ; Heiner, M.: Petri nets in Snoopy: A unifying framework for the graphical display, computational modelling, and simulation of bacterial regulatory networks. In: Helden, Jv; Toussaint, A; Thieffry, D (Eds.): *Methods in Molecular Biology – Bacterial Molecular Networks* Volume 804. Humana Press, 2012, Chapter 21, pp. 409–437 {2, 49}

[MSS+08] Moraru, I.; Schaff, J.; Slepchenko, B.; Blinov, M.; Morgan, F.; Lakshminarayana, F.; Li, Y. ; Loew, L.: Virtual Cell modelling and simulation software environment. . In: *IET Syst Biol.* 2 (2008), Nr. 5, pp. 352–62 {1}

[MTA+03] Matsuno, H.; Tanaka, Y.; Aoshima, H.; Doi, A.; Matsui, M. ; Miyano, S.: Biopathways representation and simulation on hybrid functional Petri net. In: *In silico biology* 3 (2003), Nr. 3 {2, 4, 55, 68, 126}

[Mur89] Murata, T.: Petri nets: Properties, analysis and applications. In: *Proceedings of the IEEE* 77 (1989), April, Nr. 4, pp. 541–580 {2, 48}

[MWL99] Mulder, J.; Wijk, J. ; Liere, R.: A survey of computational steering environments. In: *Future Generation Computer Systems* 15 (1999), Nr. 1, pp. 119–129 {1, 9, 11, 12, 15, 20}

[NDMM04] Nagasaki, M.; Doi, A.; Matsuno, H. ; Miyano, S.: A versatile Petri net based architecture for modeling and simulation of complex biological processes. In: *Genome informatics.* 15 (2004), Nr. 1, pp. 180–197 {4, 55}

[NSJ+10] Nagasaki, M.; Saito, A.; Jeong, E.; Li, C.; Kojima, K.; Ikeda, E. ; Miyano, S.: Cell Illustrator 4.0: A computational platform for systems biology. In: *In Silico Biol* 10 (2010), Nr. 0002 {1}

[Pah09] Pahle, J.: Biochemical simulations: stochastic, approximate stochastic and hybrid approaches. In: *Brief Bioinform* 10 (2009), Nr. 1, pp. 53–64 {5, 29, 36, 39, 42, 45, 77}

[Par99] Parker, S.: *The SCIRun Problem Solving Environment and Computational Steering Software System*, University of Utah, PhD thesis, 1999 {15, 16, 19, 20}

[Pet62] Petri, Carl A.: *Kommunikation mit Automaten*, Bonn: Institute für Instrumentelle Mathematik, PhD thesis, 1962 {46}

[PHPP05] Pickles, S.; Haines, R.; Pinning, R. ; Porter, A.: A practical toolkit for computational steering. In: *Phil Trans R Soc* 363 (2005), Nr. 1833, pp. 1843–1853 {1, 2, 10, 23}

[PJ95] Parker, S.; Johnson, C.: SCIRun: a scientific programming environment for computational steering. In: *Proceedings of the 1995 ACM/IEEE conference on Supercomputing.* New York, NY, USA, 1995 {2, 10, 16, 24}

[PJB97] Parker, S.; Johnson, C. ; Beazley, D.: Computational steering: Software systems and strategie. In: *IEEE Computational Science Engineering* 4 (1997), Nr. 4, pp. 50–59 {11, 12, 15, 16, 17, 24}

[PTVF02] Press, William H.; Teukolsky, Saul A.; Vetterling, William T. ; Flannery, Brian P.: *Numerical Recipes in C++: The Art of Scientific Computing.* Cambridge University Press, Februar 2002. {33, 40}

[PVM12] *Parallel Virtual Machine website.* `http://www.csm.ornl.gov/pvm/`. 2012. – Accessed: 1/31/2012 {21}

[PWC11] Pu, Y.; Watson, L ; Cao, Y.: Stiffness detection and reduction in discrete stochastic simulation of biochemical systems. In: *J. Chem. Phys* 134 (2011), Nr. 054105 {40}

[Pyt12] *Python Website.* `http://python.org/`. 2012. – Accessed: 1/22/2012 {16}

[PZV12] Potapov, I.; Zhurov, B. ; Volkov, E.: "Quorum sensing" generated multistability and chaos in a synthetic genetic oscillator. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 22 (2012), Nr. 2, pp. 023117 {129}

[R12] *R Project Website*. `http://www.r-project.org/`. – Accessed: 1/22/2012 {16}

[Ray86] Raynal, Michel: *Algorithms for Mutual Exclusion*. MIT Press, 1986 {109}

[RMH10] Rohr, C.; Marwan, W. ; Heiner, M.: Snoopy–a unifying Petri net framework to investigate biomolecular networks. In: *Bioinformatics* 26 (2010), April, Nr. 7, pp. 974–975 {3, 57, 102}

[RML93] Reddy, V.; Mavrovouniotis, M. ; Liebman, M.: Petri Net Representations in Metabolic Pathways. In: *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology*, AAAI Press, 1993, pp. 328–336 {2}

[ROB05] Ramsey, S.; Orrell, D. ; Bolouri, H.: Dizzy: stochastic simulation of large-scale genetic regulatory networks. In: *J Bioinform Comput Bio* 3 (2005), Nr. 2, pp. 415–36 {1}

[RPCG03] Rathinam, M.; Petzold, L.; Cao, Y. ; Gillespie, D.: Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method. In: *J. Chem. Phys.* 119 (2003), Nr. 12784 {39, 41}

[Sch02] Schulze, T.: Kinetic Monte Carlo simulations with minimal searching. In: *Physical Rev. E.* 65 (2002), Nr. 3, pp. 036704 {41}

[SFR03] Stevens, W.; Fenner, B. ; Rudoff, A.: *Unix Network Programming: The Sockets Networking API*. Volume Volume 1. 3rd Edition. Addison-Wesley Professional, 2003 {110}

[SGCK+08] Sabouri-Ghomi, M.; Ciliberto, A.; Kar, S.; Novak, B. ; Tyson, J.: Antagonism and bistability in protein interaction networks. In: *Journal of Theoretical Biology* 250 (2008), Nr. 1, pp. 209 – 218. {119, 121, 122}

[SK05] Salis, H.; Kaznessis, Y.: Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions. In: *J. Chem. Phys* 122 (2005), Nr. 5 {29, 42, 45}

[Sno12] *Snoopy website*. `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy`. 2012. – Accessed: 2/2/2012 {57, 79}

[SSJT11] Singhania, R.; Sramkoski, R.; Jacobberger, J. ; Tyson, J.: A hybrid model of mammalian cell cycle regulation. In: *PLoS Comput Biol* 7 (2011), 02, Nr. 2, pp. e1001077 {119, 123}

[Ste04] Steuer, R.: Effects of stochasticity in models of the cell cycle: from quantized cycle times to noise-induced oscillations. In: *Journal of Theoretical Biology* 228 (2004), Nr. 3, pp. 293 – 301 {121}

[SWR+11] Shu, J.; Watson, L.; Ramakrishnan, N.; Kamke, F. ; Deshpande, S.: Computational Steering in the Problem Solving Environment WBCSim. In: *Engineering Computations* 28 (2011), Nr. 7, pp. 888 – 911 {1, 2, 107}

[SYSY02] Srivastava, R.; You, L.; Summers, J. ; Yin, J: Stochastic vs. Deterministic Modeling of Intracellular Viral Kinetics. In: *J. theor. Biol.* 218 (2002), Nr. 3, pp. 309–321 {74, 116, 117, 119}

[TK93] Trivedi, K.; Kulkarni, V.: FSPNs: Fluid Stochastic Petri Nets. In: *14th International Conference on Application and Theory of Petri Nets*, 1993, pp. 24–31 {55, 62, 85}

[TKHT04] Takahashi, K.; Kaizu, K.; Hu, B. ; Tomita, M.: A multi-algorithm, multi-timescale method for cell simulation. In: *Bioinformatics* 20 (2004), März, Nr. 4, pp. 538–546 {45}

[TN01] Tyson, J.; Novak, B.: Regulation of the eukaryotic cell cycle: molecular antagonism, hysteresis, and irreversible transitions. In: *Journal of Theoretical Biology* 210 (2001), Nr. 2, pp. 249 – 263 {119, 121, 122}

[TN11] Tyson, J.; Novak, B.: *A systems biology view of the cell cycle control mechanisms*. Elsevier, San Diego, CA,, 2011 {120}

[Val78] Valk, R.: Self-modifying nets, a natural extension of Petri nets. In: *Proceedings of the Fifth Colloquium on Automata, Languages and Programming*. London, UK,: Springer-Verlag, 1978, pp. 464–476 {68, 126}

[VS96] Vetter, J.; Schwan, K.: Models for Computational Steering. In: *Proceedings of the 3rd International Conference on Configurable Distributed Systems*. Washington,USA: IEEE Computer Society, 1996, pp. 100–108 {9, 12, 14, 20, 26, 27}

[VS99] Vetter, J.; Schwan, K.: Techniques for high-performance computational steering. In: *IEEE Concurrency* 7 (1999), pp. 63–74. – {17, 19, 20, 24, 27}

[VTK12] *Visualization Toolkit website*. `http://www.vtk.org/`. 2012. – Accessed: 1/24/2012 {11}

[WGMH10] Wolf, V.; Goel, R.; Mateescu, M. ; Henzinger, T.: Solving the chemical master equation using sliding windows. In: *BMC Syst. Biol.* 4 (2010), Nr. 1, pp. 42 {35, 36, 85}

[wik12] *wikipedia website*. `http://www.wikipedia.org/`. 2012. – Accessed: 20/6/2012 {121}

[WUKC04] Wolkenhauer, O.; Ullah, M.; Kolch, W. ; Cho, K.: Modeling and simulation of intracellular dynamics: choosing an appropriate framework. In: *IEEE Trans. Nanobiosci.* 3 (2004), Nr. 3, pp. 200–207 {29, 31, 35, 36}

[wxW12] *WxWidgets website*. `http://www.wxwidgets.org`. 2012. – Accessed: 30/3/2012 {110}

[YLL09] Yang, H.; Lin, C. ; Li, Q.: Hybrid simulation of biochemical systems using hybrid adaptive Petri nets. In: *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, ICST, 2009, pp. 410–420 {55, 116}