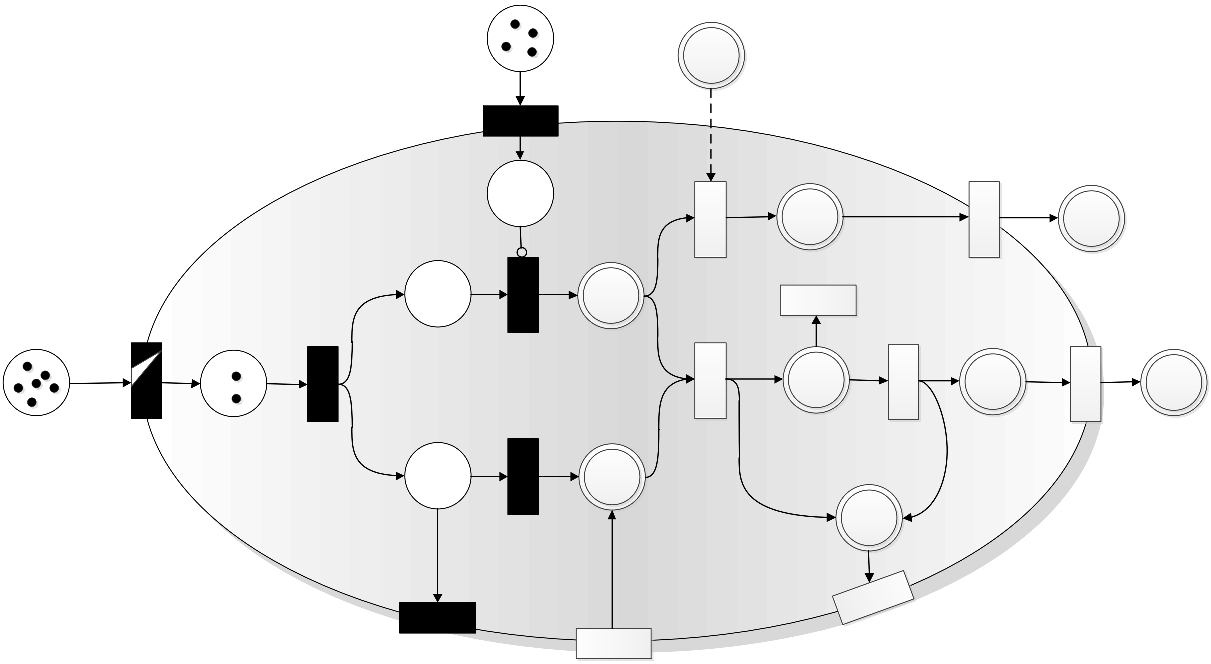
HYBRID MODELING AND OPTIMIZATION OF BIOLOGICAL PROCESSES



Dissertation zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften

Eingereicht an der Technischen Fakultät der Universität
Bielefeld von

Sabrina Proß

Sabrina Proß

Hybrid Modeling and Optimization
of Biological Processes

Dissertation

Technische Fakultät der Universität Bielefeld
AG Bioinformatik, Prof. Dr. Ralf Hofestädt

Fachbereich Ingenieurwissenschaften und Mathematik der Fachhochschule Bielefeld
Angewandte Mathematik, Prof. Dr. Bernhard Bachmann

Gutachter:
Prof. Dr. Ralf Hofestädt, Universität Bielefeld
Prof. Dr. Bernhard Bachmann, Fachhochschule Bielefeld
Prof. Dr. François Cellier, Eidgenössische Technische Hochschule Zürich

Prüfungsausschuss:
Prof. Dr. Bernhard Bachmann, Fachhochschule Bielefeld
Prof. Dr. Ralf Hofestädt, Universität Bielefeld
Prof. Dr. Thomas Noll, Universität Bielefeld
Dr. Roland Wittler, Universität Bielefeld

Druck: Bielefeld, März 2013
Gedruckt auf alterungsbeständigem Papier (ISO 9706)

ACKNOWLEDGEMENTS

This PhD-project was carried out as a part of a cooperation between the Bielefeld University (Faculty of Technology) and the University of Applied Sciences Bielefeld (Department of Engineering and Mathematics) and was funded by gender equality policies of the University of Applied Sciences Bielefeld.

Without the help and support of many people, this thesis could not have been completed. Herewith, I would sincerely like to thank all of them. First of all, my gratitude goes to my supervisor Prof. Dr. Ralf Hofestädt for his valuable input and his support throughout my research for this work. Furthermore, I would like to thank Prof. Dr. Bernhard Bachmann for many most helpful discussions and suggestions as well as his encouragement.

I would like to thank all my colleagues in the department of Applied Mathematics including Prof. Dr. Friedrich Biegler-König, Prof. Dr. Claudia Cottin, Dr. Elke Koppenrade, Prof. Dr. Kruse, Prof. Dr. Svetozara Petrova, Tatiana Schenck, and Jens Schönbohm, for the positive atmosphere as well as for the motivating and inspiring discussions. Special thanks go to Prof. Dr. Rainer Ueckerdt for many helpful comments and great suggestions, to Willi Braun for his excellent Modelica support, and to Ralf Derdau for his outstanding IT-support. In addition, I would like to thank my colleagues at the Bioinformatics/Medical Informatics Department, Sebastian Janowski and Benjamin Kormeier, for the close collaboration and their help. In addition, my gratitude goes to Dr. Perta Lutter, Prof. Dr. Karsten Niehaus, and Dr. Frank Vorhölter at the Center of Biotechnology for their great help and support in all biological topics.

Further thanks go to Prof. Dr. Thomas Noll and Dr. Roland Wittler who agreed to be part of the examination board. I am also grateful to Barbara Davis for the careful proofreading of the manuscript. In addition, I would like to thank the University of Applied Sciences Bielefeld for the financial support of this work.

Finally, I am deeply grateful to my parents Bärbel und Horst and my brother Christian for all their support and encouragement and, most of all, I would like to thank Michael, for all his patience, understanding, and support during the time of this work.

ABSTRACT

Modern computer techniques and large memory capacities make it possible to produce an enormous amount of biological data stored in huge databases. This data is indispensable for scientific progress but does not necessarily lead to insight about the functionality of biological systems. Hence, an approach is needed to achieve usable information from this huge amount of data. A **mathematical model** provides a means for summarizing and structuring experimental data in order to simplify the communication of knowledge progress with other researchers. Additionally, it improves the understanding of the living system and allows the directed design of experiments by predicting system behavior under specific conditions and proofs with experiments.

To enable the processing of experimental data to usable new insights about biological systems, a **general, universally usable modeling process for biological systems** has been developed in this work. This modeling process requires several mathematical methods to achieve a "well-working" parameterized model which is able to predict the behavior of the underlying system and forms the basis for optimizing biological processes. Therefore, an environment has been developed which comprises mathematical methods and tools for covering all steps of the established modeling process.

Petri nets with their various extensions are a universal graphical modeling concept for representing biological systems in nearly all degrees of abstraction. They provide an intuitive and generally comprehensible way for representing and communicating experimental data and knowledge of biological systems. Despite several works and publications with Petri net approaches, there is a serious problem regarding the lacking unity of concepts, notations, and terminologies. The definition of Petri nets is not standardized; every author has personal definitions which are partly not accurate enough, not common, or even contradictory. In this work, to show the research community the power of Petri nets, they are defined precisely together with the corresponding processes, which are essential for simulation. A formalism has been developed which is able to represent nearly all kinds of biological processes. It is called e**x**tended **H**ybrid **P**etri **N**ets and abbreviated with **xHPN**. This formalism has been

further extended by providing the Petri net elements with biological meaning and this extension is called **xHPNbio** (**ex**tended **H**ybrid **P**etri **N**ets for **bio**logical applications).

To use Petri nets as a graphical modeling language for biological systems, the Petri nets, for their part, have been modeled by the object-oriented modeling language **Modelica**. Modelica was developed and promoted by the Modelica Association since 1996 for modeling, simulation, and programming of primarily physical and technical systems and processes and has become the de-facto standard for hybrid, multidisciplinary modeling. Each Petri net element, place and transition, is described by a model in the Modelica language defined on the lowest level by discrete (event-based), differential, and algebraic equations. The developed Petri net components models are structured in a Modelica library called **PNlib** (**P**etri **N**et **lib**rary). An appropriate Modelica-tool then enables graphical and hierarchical modeling, hybrid simulation, and animation.

To simplify the modeling process and, in addition, to give all involved researchers an adapted view of the model at a specific level of detail, the models have to be constructed in a hierarchical structure. The Petri nets on the one hand and the Modelica language on the other, enables hierarchical modeling concepts by **wrapping** the basic Petri net elements into sub-models which represent specific processes or reactions used many times in the same or in different models. Wrappers for common biological reactions and processes are summarized in the Modelica library **PNproBio (P**etri **N**ets for **pro**cess modeling of **Bio**logical systems**)**.

In addition to the modeling concept (xHPN) and libraries (PNlib and PNproBio), appropriate mathematical methods are needed for analyzing the established models in order to determine the model parameters, evaluate their robustness towards small changes, simulate deterministically as well as stochastically, and optimize and control the underlying biological processes. All these mathematical methods are available within one MATLAB-tool, called **AMMod** (**A**nalysis of **M**odelica **Mod**els). Together with the powerful xHPN formalism and the PNlib as well as the PNproBio, all steps of the modeling process are covered by this new environment.

The general modeling procedure as well as the applicability of the developed concepts and tools is shown exemplarily by modeling the **xanthan production** of *Xanthomonas campestris* bacteria.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 MOTIVATION

Conditioned by the huge memory capacity and the possibilities of modern computer techniques, new data is produced incessantly in all imaginable areas of life. This pertains also to the biological sciences; modern **high throughput experiments** generate an enormous amount of data stored in large databases. This data is indispensable but does not necessarily lead to insight about the functionality of biological systems. Hence, the question arises: How is it possible to achieve usable knowledge from this huge amount of data?

One possible approach is to transform the data of biological systems as well as knowledge about functions, interactions, and relationships into a **mathematical model**. A mathematical model provides a means for summarizing and structuring experimental data in order to simplify the communication of knowledge progresses with other researchers. Additionally, it improves the understanding of the living system and allows the directed design of experiments by predicting the system behavior under specific conditions and proving by experimentation. Models cannot replace experiments but they can help plan them and improve as such the employment of resources. The modeling approach enables a ranking of the experiments to execute those where are most economical in terms of both time and financial costs. This economical aspect of modeling and simulation is of great interest. The **in-silico** design of experiments with virtual cells is used, for example, in the development of pharmaceuticals to reduce animal testing.

Once a model is established, hypotheses about system properties and behavior can be derived. These hypotheses can be validated in parallel by in-silico and in-vitro experiments. Simulation data is produced in-silico by simulating the model and experimental data is gained in-vitro from the wet-lab. Such an **iterative process** leads to new knowledge about the regarded systems. But the cycle of simulations and wet-lab experiments can also be performed successively. Then, the in-silico simulations firstly identify the experiments which are indeed necessary and optimize the experimental design.

In general, a strong interaction is essential between experimental data collection and theoretical computer-based modeling and simulation. Hence, investigators from the disciplines biology, mathematics, informatics, and systems science have to work together **interdisciplinarily**. The biologists provide biological phenomena to investigate, the required experimental data and a first visualization of the model. This first visualization is specified by system scientists, and mathematicians as well as computer scientists therefore contribute the necessary modeling, simulation, and analysis tools.

Numerous model formalisms have been proposed for modeling and simulation biological systems (see e.g. (Wiechert 2002)). Generally, it has to be distinguished between qualitative and quantitative approaches. **Qualitative models** represent only the fundamental compounds, their interaction mechanisms, and the relationships between them while **quantitative models** describe, in addition, the time-related changes of the components. Hence, a qualitative model is the basis for every quantitative model and the mentioned improved data basis enables us to extend qualitative models to quantitative ones today. Beyond this, quantitative model formalisms can be further divided into **discrete** and **continuous** approaches as well as **deterministic** and **stochastic** techniques.

The decision as to which modeling approach is used is difficult and strongly influenced by the availability of **data**. If all kinetic data is known, models consisting of ordinary differential equations are mostly the first choice while in the absence of kinetic data only qualitative approaches are usable. An additional difficulty arises in the demand of simultaneously having a model which is easy to understand and an abstraction of the real system as well as a detailed and nearly complete description of it. Besides, the modeling process of biological systems is further complicated by incomplete knowledge, noisy and inaccurate data, and different ways of representing data and knowledge.

Petri nets with their various extensions are a universal graphical modeling concept for representing biological systems in nearly all degrees of abstraction. They support the qualitative modeling approach as well as the quantitative one. Once a qualitative Petri net model has been established, the quantitative data can be added successively. Furthermore, the biological processes can be modeled discretely as well as continuously and, in addition, discrete and continuous processes can also be combined within one Petri net model to so-called **hybrid Petri nets** (see e.g. (David and Alla 2001)). The Petri net formalism with all its extensions is so powerful that all other formalisms are included and, hence, only one formalism is needed regardless of the approach (qualitative vs. quantitative, discrete vs.

continuous, deterministic vs. stochastic) which is appropriate for the respective system. The Petri net formalism is easy to understand for all researchers from the different disciplines (biology, mathematics, informatics, and system sciences) which have worked together in the modeling process and is an ideal way for intuitive representing and communicating experimental data and knowledge of biological systems. Petri nets allow hierarchical structuring of models and offer the possibility of different detailed views for every observer of the model.

To use Petri nets as a graphical modeling concept for biological systems, the Petri nets, for their part, have to be modeled textually by an appropriate language. The object-oriented modeling language **Modelica**, developed and promoted by the Modelica Association since 1996 for modeling, simulation, and programming primarily of physical and technical systems and processes (Modelica Association 2010), is ideally suited for this task. Modelica has become the de-facto standard for hybrid, multidisciplinary modeling. Each Petri net element, place and transition, can be described with the aid of a model in the Modelica language defined on the lowest level by discrete (event-based), differential, and algebraic equations. An appropriate Modelica-tool then enables graphical and hierarchical modeling, hybrid simulation, and animation.

However, model construction based on such a strong formalism alone is not all that is needed for a good working model which can be simulated. Usually, the constructed model comprises several parameters like speed constants or yield coefficients which have to be estimated. Several databases are already available which summarize these specific parameters, for example, the database BRENDA, which provides a collection of enzyme functional data (Schomburg et al. 2002). If the required parameters are not listed in databases or publications, they have to be estimated by experiments. But sometimes these experiments are too expensive, imprecise, or not even feasible. In these cases, specific mathematical optimization methods can provide means to adapt the model behavior to the given experimental data as well as possible. This procedure is called **parameter estimation**. Direct linked with it is the **sensitivity analysis** of model parameters which identify, optimize, reduce, and verify the model.

Furthermore, particular mathematical optimization methods enable the optimization of biological processes, called **process optimization**. This plays an important role in industrial biotechnology. Based on a model, it is thus possible to control biological processes in the best possible way and to gain maximum product yields from the cultivated organisms.

1.2 Objectives

The main objective of this work is the development of a **general, universally usable modeling process** for biological systems from observing a phenomenon to a “well-working” parameterized model which is able to predict the behavior of the underlying system and forms the basis for optimizing biological processes.

Directly connected to this is the aim to provide researchers from different disciplines an **environment** which comprises methods and tools for covering all steps of the developed modeling process. Therefore, specific adapted mathematical concepts, formalisms, and methods are of crucial importance to enable

- Data preprocessing and relationship analysis,
- Graphical, hierarchical modeling,
- Deterministic and stochastic hybrid simulation and animation,
- Parameter estimation,
- Sensitivity analysis, and
- Process optimization.

As mentioned before, it has become apparent that **Petri nets** with their various extensions are the ideal modeling formalism which is universally applicable for all different kinds of biological phenomena. They provide an intuitive and generally comprehensible way for representing and communicating experimental data and knowledge of biological systems. Despite several works and publications with Petri net approaches, there is a serious problem regarding the lacking unity of concepts, notations, and terminologies. The definition of Petri nets is not standardized; every author has his/her own definitions which are partially not accurate enough, not common, or even contradictory. Hence, to show the research community the power of Petri nets, they need to be precisely defined together with the corresponding processes which are essential for the simulation. This has been done within this work; based on the Petri net initially introduced by Carl Adam Petri in 1962 (Petri 1962), a formalism has been developed which is able to represent nearly all kinds of biological processes. It is now called e**x**tended **H**ybrid **P**etri **N**ets and abbreviated with **xHPN**.

This clear and precise definition alone is not sufficient. Once a Petri net model is created, one would like to simulate it for predicting the behavior under different parameter settings, for example, to optimize the experimental design. Therefore, the precise definitions of Petri nets and their corresponding processes are an imperative necessity. The Petri nets for their part, have to be modeled with an appropriate language. As mentioned before, the object-oriented modeling language **Modelica** is ideally suited for this task. A proper Modelica-tool then enables the graphical and hierarchical modeling, hybrid simulation and animation of Petri nets. Within this work, an innovative Modelica Petri net library has been developed called **PNlib** (see also Proß and Bachmann 2009, Proß and Bachmann 2011a, Proß and Bachmann 2011b, Proß and Bachmann 2012b).

To simplify the modeling process itself and, in addition, to give all involved researchers an adapted view of the model at a specific level of detail, the models have to be constructed within a hierarchical structure. The Petri nets on the one hand and the Modelica language on the other enables such hierarchical modeling concepts by **wrapping** the basic Petri net elements into sub-models which represent specific processes or reactions used many times in the same or in different models. Wrappers for common biological reactions and processes are summarized in the Modelica library PNproBio (**P**etri **N**ets for **pro**cess modeling of **Bio**logical systems) (see also Proß et al. 2009, Proß and Bachmann 2011a, Proß and Bachmann 2011b, Proß and Bachmann 2012a).

In addition to the modeling concept (xHPN) and tool (PNlib and PNproBio), appropriate mathematical methods are needed for analyzing the established models in order to determine the model parameters, evaluate their robustness towards small changes, simulate deterministically as well as stochastically, and optimize and control the underlying biological processes. All these mathematical methods are available within one MATLAB-tool, called AMMod (**A**nalysis of **M**odelica **Mod**els). Together with the powerful xHPN formalism and the PNlib, all steps of the modeling process are covered by this new environment (see also Proß and Bachmann 2012a).

1.3 STRUCTURE

This work is structured in the following way, which is also depicted in Figure 1.1. At first, **related works** are presented to strengthen the demand of a powerful modeling formalism, a new modeling tool, and analysis methods especially for biological systems.

Chapter 3 comprises the **basics** which are needed to realize the modeling process. Thereby, the first section introduces the Modelica language concepts which are essential for modeling the components of xHPN, places and transitions, for their part with the Modelica language. The next section comprises optimization methods to enable parameter estimation and process optimization of biological models. Several local, global, and hybrid optimization methods are presented which are applicable to solve the respective optimization problem. Additionally, methods for sensitivity analysis are proposed in Section 3.3 to improve the procedure of parameter estimation and to reduce the model complexity.

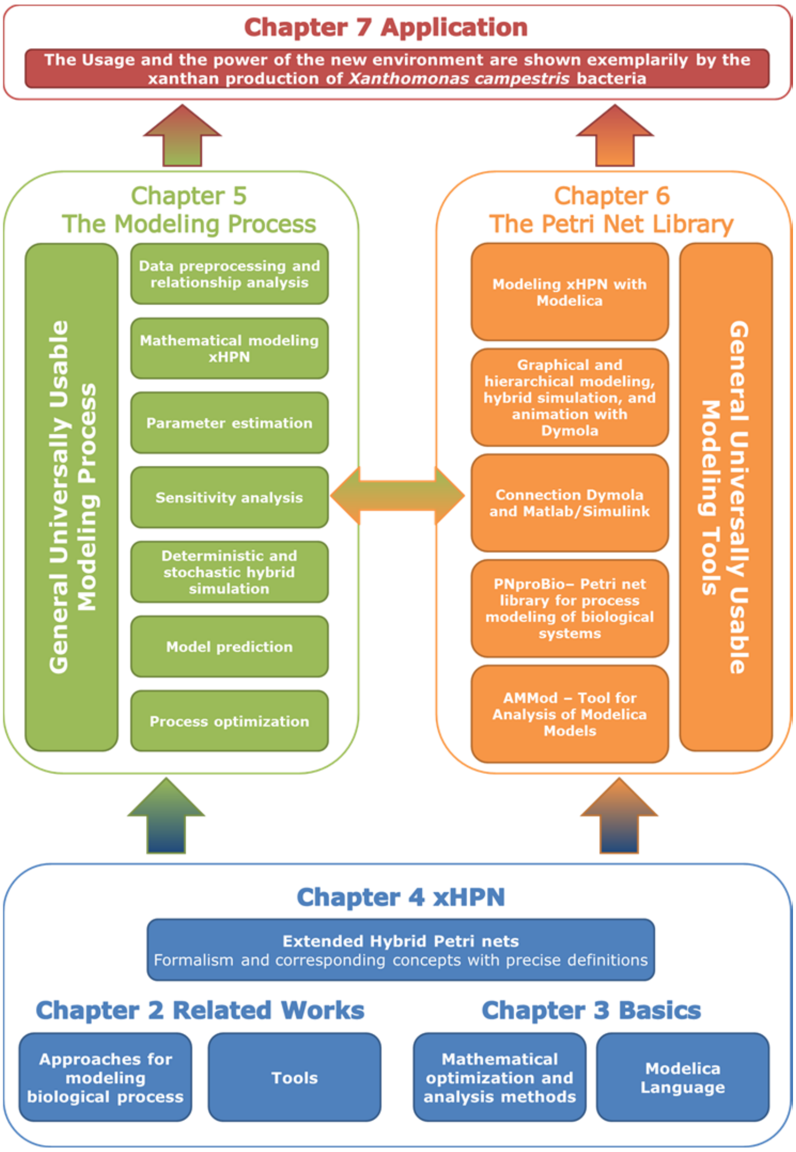

Chapter 7 Application
The Usage and the power of the new environment are shown exemplarily by the xanthan production of *Xanthomonas campestris* bacteria
Chapter 5
The Modeling Process
General Universally Usable Modeling Process
Data preprocessing and relationship analysis
Mathematical modeling xHPN
Parameter estimation
Sensitivity analysis
Deterministic and stochastic hybrid simulation
Model prediction
Process optimization
Chapter 6
The Petri Net Library
Modeling xHPN with Modelica
Graphical and hierarchical modeling, hybrid simulation, and animation with Dymola
Connection Dymola and Matlab/Simulink
PNproBio– Petri net library for process modeling of biological systems
AMMod – Tool for Analysis of Modelica Models
General Universally Usable Modeling Tools
Chapter 4 xHPN
Extended Hybrid Petri nets
Formalism and corresponding concepts with precise definitions
Chapter 2 Related Works
Approaches for modeling biological process
Tools
Chapter 3 Basics
Mathematical optimization and analysis methods
Modelica Language

Figure 1.1: The structure of this work

Chapter 4 precisely defines the developed **Petri net concepts** from the basic one up to the extended hybrid Petri nets (xHPN) and serves as the basis for the implementation of the xHPN formalism in order to model and simulate biological processes.

Chapter 5 describes the developed **modeling process** and how the basic concepts, presented in Chapter 3, and the xHPN formalism are particular adapted for biological processes.

In Chapter 6, it is described how the Modelica language is used for implementing the components of xHPN structured in a Modelica library called **PNlib**. Additionally, the modeling, simulation, and animation of xHPN with the Modelica-tool **Dymola** are described. Furthermore, the Modelica library **PNproBio** (**P**etri **N**ets for **pro**cess modeling of **Bio**logical systems) is presented which comprises wrappers for common biological reactions to simplify the modeling process and, in addition, to improve the view of a model. Thereafter, it is explained how **Dymola** and **Matlab/Simulink** are connected to establish a bridge from PNlib, the modeling-tool, to **AMMod**, the analysis tool. The functionality and usage of this analysis tool for Modelica models is then described in the next section.

Chapter 7 demonstrates exemplarily the modeling process as well as the application of the tools developed in this work by the **xanthan production** of *Xanthomonas campestris* bacteria. Thereby, the chapter focuses on describing the general procedure of modeling but not on the selected example. Hence, no new insights about the bacteria and the xanthan production are presented due to using pseudo experimental data; but rather the usage of the developed environment is shown as well as the power and suitability is proven. Thereby, all steps of the modeling process are performed exemplarily by models with different complexities. Furthermore, the performance of the optimization methods for parameter estimation and process optimization are tested and rated. Additionally, sensitivity analysis is applied in order to improve the parameter estimation procedure. Moreover, a stochastic Petri net model of the xanthan production is constructed to compare the results of the stochastic hybrid simulation with those of the deterministic hybrid simulation.

Finally, Chapter 8 summarizes and discusses the results of this work and provides an **outlook** of ongoing and further extension possibilities. It is shown that the developed modeling and analysis concepts are not restricted to biological processes; they are universally usable for nearly all processes like business processes, production processes, traffic processes etc. Most of all it has come to light that the xHPN formalism in Modelica is an **ideal all-round modeling concept** for graphical and hierarchical modeling, simulation, and optimization of various hybrid processes.

2 RELATED WORKS

This chapter comprises related works corresponding to this study. At first, Petri net approaches for modeling biological processes are introduced and the demand for a precise definition of Petri nets and their corresponding processes which are necessary for simulation is clarified. Afterwards, tools dealing with the hybrid Petri net formalism are listed, in particular, two common tools, Cell Illustrator and Snoopy, are presented which support the modeling of biological systems by means of the hybrid Petri net formalism. The last section comprises already existing Petri net approaches in Modelica and makes clear why these approaches are not sufficient for modeling biological systems.

2.1 PETRI NET APPROACHES FOR MODELING BIOLOGICAL PROCESSES

The Petri net formalism was first introduced by Carl Adam Petri in 1962 for modeling and visualization of concurrency, parallelism, synchronization, resource sharing, and non-determinism (Petri 1962). A Petri net is a graph with two different kinds of nodes, called **transitions** and **places**; thereby, places and transitions are connected by arcs. Every place in a Petri net can contain an integer number of **tokens**. These tokens initiate transitions to fire according to specific conditions. These firings lead to changes of the tokens in the places.

Reddy et al. propose firstly to apply this formalism for biological network modeling in order to represent and analyze metabolic pathways in a qualitative manner (Reddy et al. 1993). Thereby, places represent biological compounds such as metabolites, enzymes, and cofactors which are part of biochemical reactions. These biochemical reactions are modeled by transitions and their stoichiometry is represented by the arc weights. In addition, the tokens indicate the presence of compounds. Reddy et al. showed that the Petri nets were a suitable approach for qualitative analysis of metabolic pathways. Properties of Petri nets such as liveness, reachability, reversibility, fairness, structural reduction, and invariants were applied to identify characteristics of the modeled systems.

Hofestädt adapted the Petri net formalism to metabolic network modeling (Hofestädt 1994). His approach allows the combination of analytic and discrete modeling in order to model metabolic processes in a natural manner by a new specialized graphical representation based on Petri nets.

Moreover, Hofestädt and Thelen expanded the approach of Reddy by introducing **functional Petri nets** to enable quantitative modeling of biochemical networks (Hofestädt and Thelen 1998). Thereby, the arc weights are functions which depend on concrete markings of places in order to model kinetic effects. The tokens then represent the concentration level of a biological compound.

Due to the fact that a random behavior of molecular reactions at low concentrations has been observed in many experiments, Goss and Peccoud introduced **stochastic Petri nets** (Goss and Peccoud 1998). A stochastic transition fires not instantaneously but with a time delay following an exponential distribution which may depend on the token numbers of the places. They illustrated their method by examples of gene regulation and biochemical reactions.

A reasonable way for modeling concentrations of biological compounds is by places containing real token numbers instead of integers and transitions which fire as a continuous flow specified by an assigned speed. The transformation from the discrete to the **continuous Petri net** concept was first introduced by David and Alla in 1987 (David and Alla 1987) and they replaced the term token by **mark** because tokens relate mostly to integer quantities.

Furthermore, Alla and David proposed combining the discrete and the continuous Petri net concept to so-called **hybrid Petri nets** (Alla and David 1998). A hybrid Petri net contains discrete places with integer tokens and discrete transitions with time delays as well as continuous places with non-negative real marks and continuous transitions with firing speeds. Matsuno et al. used this approach for modeling gene regulatory networks by discrete and continuous processes (Matsuno et al. 2000). They improved this approach further by adding the properties of functional Petri nets to it so that the arcs as well as the speeds of the transitions are functions depending on the marks of the places (Matsuno et al 2003). This modification is called **hybrid functional Petri nets**. Additionally, they extended the hybrid functional Petri nets by two specific arcs, called **test** and **inhibitor arcs** (Matsuno et al 2003), to accomplish the modeling of inhibition and activation mechanisms of biological reactions. If places are connected with test or inhibitor arcs to transitions, their markings are not changed during the firing processes. The markings are only read to influence the time of firing. In the case of a test arc, the connected place must have more tokens than the arc weight and in the case of an inhibitor arc, the connected place needs less tokens than the arc weight to enable firings of the transition. Chen and Hofestädt as well as Doi et al. demonstrated the applicability of this approach by modeling molecular networks (Chen and Hofestädt 2003, Doi et al. 2004).

Moreover, Nagasaki et al. extended the hybrid functional Petri nets further by types with which various data types can be regarded to model more complex biological processes which involve various kinds of biological information and data (Nagasaki et al. 2004). They called this approach **hybrid functional Petri nets with extensions** (HFPNe).

Despite these mentioned works and publications, there is a serious problem because of the lacking unity of concepts, notations, and terminologies. The definition of Petri nets is not standardized; every author has personal definitions which are partly not accurate enough, not common, or contradictory. Hence, to show the research community the power of Petri nets, they have to be precisely defined together with the corresponding processes which are essential for the simulation. This has been done within this work; based on the Petri net initially introduced by Carl Adam Petri in 1962 (Petri 1962), a formalism has been developed which is able to represent nearly all kinds of biological processes. It is called e**x**tended **H**ybrid **P**etri **N**ets and abbreviated with **xHPN**.

2.2 PETRI NET TOOLS

Several tools are available for modeling most differing applications by means of the Petri net formalism; an overview can be found in (Petri Net World 2012). Only a few of them are able to simulate hybrid Petri nets; all common tools according to present knowledge are listed in Table 2.1. Two of them were developed specially for modeling biological processes with the aid of hybrid Petri net formalism. The first is the commercial tool Cell Illustrator and the second is the freely available tool Snoopy. They are described in detail hereafter to clarify the demand for a new environment for modeling and analyzing biological systems. The reasons why other tools are unsuitable for this work are summarized in Table 2.1.

Cell Illustrator

The Cell Illustrator, Genomic Object Net called before, is a commercial, widely-used tool available as a Java Web Start application which enables the drawing, modeling, elucidating, and simulating of complex biological processes and systems based on hybrid functional Petri nets with extensions (Nagasaki et al. 2010). This is an extension of the hybrid functional Petri net concept by types with which various data types can be regarded to model more complex biological processes which involve various kinds of biological information and data. Additionally, discrete and continuous processes can be connected to perform hybrid simulations.

The models can be created with the standard Petri net elements or specific icons for several biological entities and processes which allow an ontology representation of the considered biological system. It is also possible to model ODEs with continuous Petri net elements by setting all input connector weights to “nocheck”.

The Cell Illustrator offers two simulation engines: a standard engine and a simulation engine code generator (SECG) (University of Tokyo 2010). Both engines return the same results but the SECG executes the simulation in a different way. At first, the Java source code is generated for the model to be simulated and afterwards it is executed as a usual Java program.

Since version 4.0, it is possible to use additional modules that are provided on servers based on the SaaS (Software as a Service)-technology. One of them is the Pathway Parameter Search Module which executes multiple simulations at once with many initial conditions.

The drawback of the Cell Illustrator is that the simulation is like a “black box”. There is no information about the following points:

- how the Petri nets and the corresponding processes are defined which are necessary for modeling and simulation, e.g. how conflicts in Petri nets are resolved,
- how the hybrid simulation is performed, and
- which integrators are used; in addition, there is no possibility to adapt solver settings in order to achieve reliable simulation results.

Furthermore, at the decision point to implement a new tool, the simulation of the Cell Illustrator does not work in a correct manner. Reactions could proceed backward if specific arc weight functions become negative during simulation. This problem has been investigated and is solved in the current version 5.0. Additionally, several simulations of continuous test models show a barely comprehensible undesirable behavior. Moreover, the conversion from discrete processes to continuous processes and vice versa is interpreted in a different way than is required and defined in this work. If, for example, a continuous place is connected to a discrete transition, this discrete transition fires continuously by deducting the arc weight from the marking at the time of the delay. The discrete transition of the developed PNlib always fires in a discrete manner regardless of whether it is connected to a continuous or discrete place (see Figure 2.1).

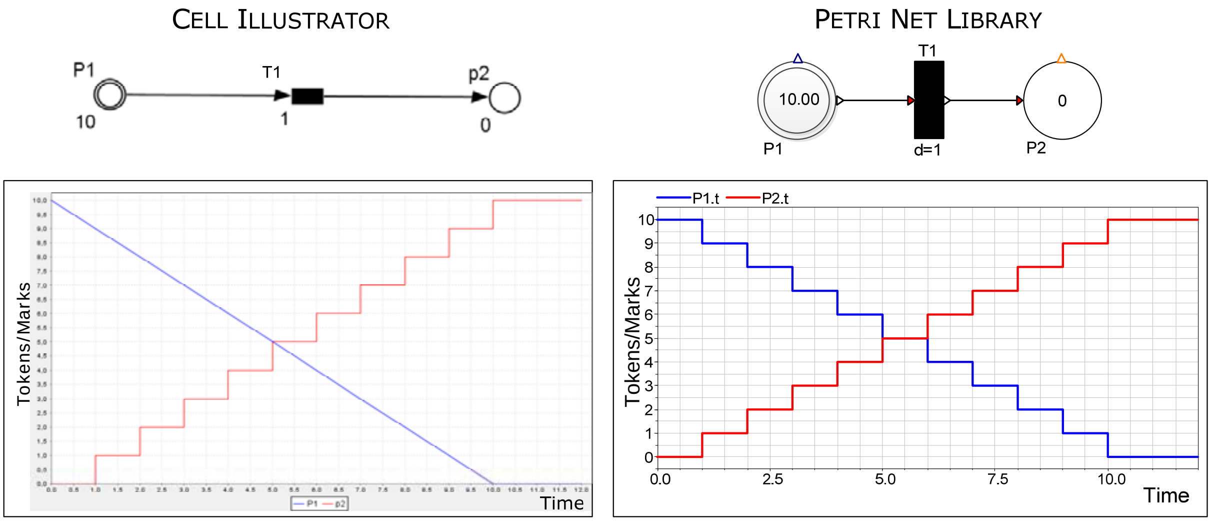


Figure 2.1: Different results of a hybrid Petri net simulated with the Cell Illustrator (left) and the PNlib in Dymola (right). A discrete transition connected to a continuous place fires continuously if the Cell Illustrator is used while it fires discretely if the PNlib is used.

The analysis methods of the Cell Illustrator are also limited to parameter scans over a desired range. According to present knowledge, there is no possibility to perform sensitivity analysis, parameter estimation, process optimization, and stochastic simulation.

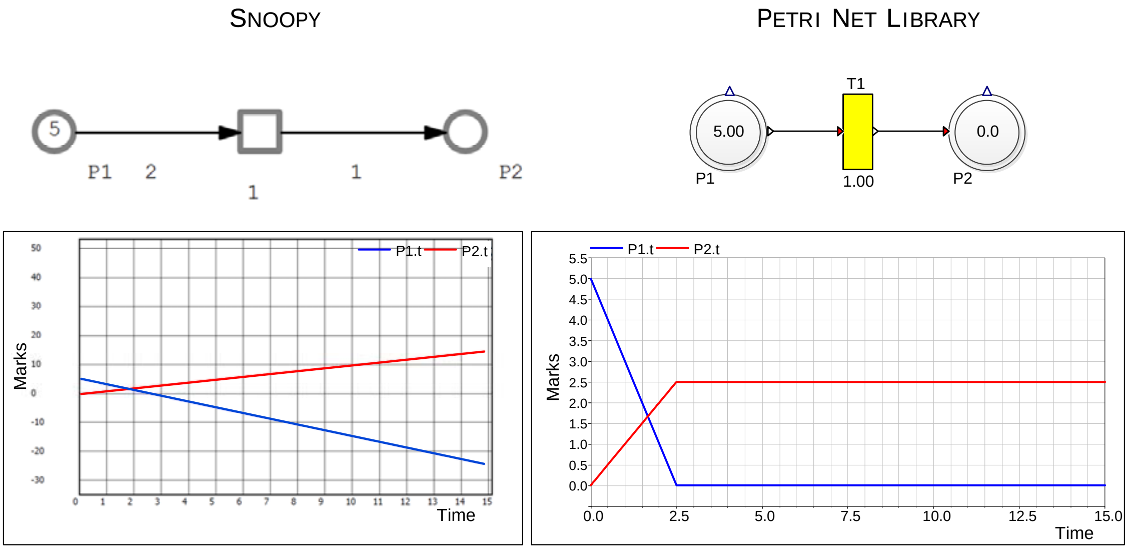
SNOOPY

Snoopy is a freely available unifying Petri net framework to investigate biomolecular networks (Rohr et al. 2010). A Petri net can be modeled time-free (qualitative model) or its behavior can be associated with time (quantitative model) such as stochastic, continuous, and hybrid Petri nets; thereby, different models are convertible into each other. It is also possible to structure the models hierarchically in order to manage complex networks.

A set of stiff and non-stiff solvers are available to perform the deterministic simulation of a continuous Petri net. The stochastic simulation of a stochastic Petri net is executed by Gillespie’s exact algorithm (Gillespie 1977). The synchronization of deterministic and stochastic simulation of a hybrid Petri net is achieved with a specific algorithm detailed by (Herajy and Heiner 2010).

Petri nets created with Snoopy can be analyzed qualitatively with the aid of the tool Charlie (BTU Cottbus 2011). The main features of Charlie are: analysis of structural properties, invariant based analysis, and reachability graph-based analysis.

The drawback of Snoopy is that a continuous Petri net is interpreted as a graphical representation of a system of ordinary differential equations. Hence, the general Petri net property of non-negative tokes cannot be maintained during simulation (see Figure 2.2). Additionally, at the decision point to implement a new tool, hybrid Petri nets were not supported by Snoopy, which is now possible (Herajy and Heiner 2010). But not all conflict situations of hybrid Petri nets are trapped and, thus, negative markings can occur. Furthermore, places cannot be provided with capacities and no functions can be assigned to arcs in hybrid Petri nets. There are also a limited number of predefined kinetic functions which complicates the modeling process. Moreover, there is no possibility to perform sensitivity analysis, parameter estimation, or process optimization with Snoopy according to present knowledge.


SNOOPY

PETRI NET LIBRARY

Figure 2.2: Different results of a continuous Petri net simulated by Snoopy (left) and the PNlib in Dymola (right). Snoopy interprets a continuous Petri net as a graphical representation of a system of ODEs while the PNlib preserves the Petri net properties by transforming the discrete concept to a continuous one (see Section 4.4).

Table 2.1: Overview of tools for modeling by hybrid Petri nets

	Features			
Overview	**Petri nets supported**	**Components**	**Environment**	**Drawback**
Cell Illustrator commercial (Nagasaki et al. 2010) http://www.cellillustrator.com/	Timed Petri nets, Stochastic Petri nets, Continuous Petri nets, Hybrid Petri nets (Hybrid functional Petri nets with extensions)	Graphical editor, Animation	Java	– Simulation like a “black box”. – Conversion of discrete to continuous markings and verse versa is interpreted in a different way. – Post-processing of simulation results is limited to parameter scans.
HISIm free of charge (Amengual 2009) http://sourceforge.net/projects/hisim/	Timed Petri nets, Colored Petri nets, Hybrid Petri nets (Differential Petri nets)	Graphical editor, Animation	Java	– Continuous places cannot be provided with capacities. – Discrete places cannot be provided with lower capacities. – The firability of discrete transitions is not influenced by the capacities of discrete places. – No conflicts are regarded and solved. – Discrete places must not be connected to continuous transitions even if they are input as well as output of the transition with arc of same weights. – Continuous places must not be connected to continuous transitions by inhibitory arcs. – Arc weights of continuous places can be negative. – Markings of continuous places can be negative. – The markings of continuous places cannot be decomposed. – When a discrete transition becomes

				active, tokens of the input places are reserved for firing. – Continuous transitions can only be inactivated by connections to discrete places via test and inhibitory arcs.
HYPENS free of charge (Sessego et al. 2008) http://www.diee.unica.it/automatica/hypens/	Timed Petri nets, Stochastic Petri nets, Continuous Petri nets, Hybrid Petri nets (First-order hybrid Petri nets)	Textual editor	MATLAB	– No graphical user interface is supported. – No inhibitory and test arcs are supported. – No capacities are supported. – The firing speeds of continuous transitions have to be constant between events.
SimHPN commercial (Júlvez and Mahulea 2012) http://webdiis.unizar.es/GISED/?q=tool/simhpn	Timed Petri nets, Stochastic Petri nets, Continuous Petri nets, Hybrid Petri nets	Graphical editor, Structural analysis	MATLAB R2008a or newer	– The firing speed of continuous transitions cannot be arbitrary functions. They are restricted to infinite and product server semantics. – Only conflict resolutions for discrete transitions are regarded. This conflict type is solved probabilistically with all transitions of the same probability. – No inhibitory and test arcs are supported. – No capacities are supported. – No information about processes important for simulation (firing, conversion discrete to continuous v.v. etc.) is available.
Snoopy free of charge (Rohr et al. 2010) http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy	Timed Petri nets, Stochastic Petri nets, Continuous Petri nets, Hybrid Petri nets	Graphical editor, Animation, Structural analysis (Charlie)	Linus, Windows, Mac OS X	– Continuous Petri net is interpreted as system of ordinary differential equations. – Not all conflicts situations are trapped. – Places cannot be provided with capacities. – No functions can be assigned to arcs in

				hybrid Petri nets. – Limited number of predefined kinetics. – No possibility to use simulation results for post-processing.
Visual Object Net++ free of charge (Drath 2002) http://www.r-drath.de/Home/Visual_Object_Net++.html	Timed Petri nets, Hybrid dynamic nets, Hybrid object nets	Graphical editor, Animation, Structural analysis	Windows	– The updating is stopped. It was further developed by the Genomic Object Net-Project from which the commercial tool Cell Illustrator arises. – Input and Output arcs of continuous transitions have always the weight one. – Firing speeds can become negative. – The markings of continuous places can be negative. – Discrete places must not be connected to continuous transitions even if they are input as well as output of the transition with arc of same weights. – Continuous places must not be connected to continuous transitions by inhibitory arcs.

As Table 2.1 clearly shows, no current tool functions without drawbacks and can interpret the hybrid Petri net formalism as needed for modeling biological systems in this work. Hence, these problems led to the development of a new Petri net simulation environment specified by the xHPN formalism and corresponding definitions for processes important for simulation. This new environment enables among others

- a simulation like a white box, i.e. all processes are defined precisely,
- that negative markings and arc weights cannot occur during simulation because quantities of biological compounds cannot become negative,
- the solution all possible conflicts which could occur during simulation in order to get reliable results; for example, modeling biological processes requires often the solution of conflicts at random due to the fact that the phenomenon is also a random process in nature or it has not yet been exactly investigated,
- the support of inhibitory and test arcs to model inhibition and activation mechanisms,
- the possibility to use the arising simulation results for post-processing,
- an object-oriented implementation which allow an easy way to maintain, extend, and modify the Petri net component models.

2.3 Petri Nets in Modelica

There are already three Petri net libraries available on the Modelica homepage (Modelica Association 2011). The first was developed by Mosterman et al. and enables the modeling of a restricted class of discrete Petri nets, called normal Petri nets (Mosterman et al. 1998). The places of normal Petri nets can only contain zero or one token. Additionally, all arcs have the weight one and external signals initiate the firing of transitions. If a conflict occurs between two or more transitions, the transition with the highest priority fires. Hence, only deterministic behavior is represented by this kind of Petri net.

The second Petri net library is an extension of the previous one and was developed by Fabricius (Fabricius 2001). The places are able to contain a non-negative integer number of tokens and can be provided with non-negative integer minimum and maximum capacities. Furthermore, the transitions are timed with fixed or stochastic delays.

The third library, called StateGraph, is based on Grafcharts which combines the function chart formalism of Grafcet with the hierarchical states of Statecharts (Johnsson and Årzén 1999). The StateGraph library is part of the Modelica standard library and was developed by Otter et al. (Otter et al. 2005).

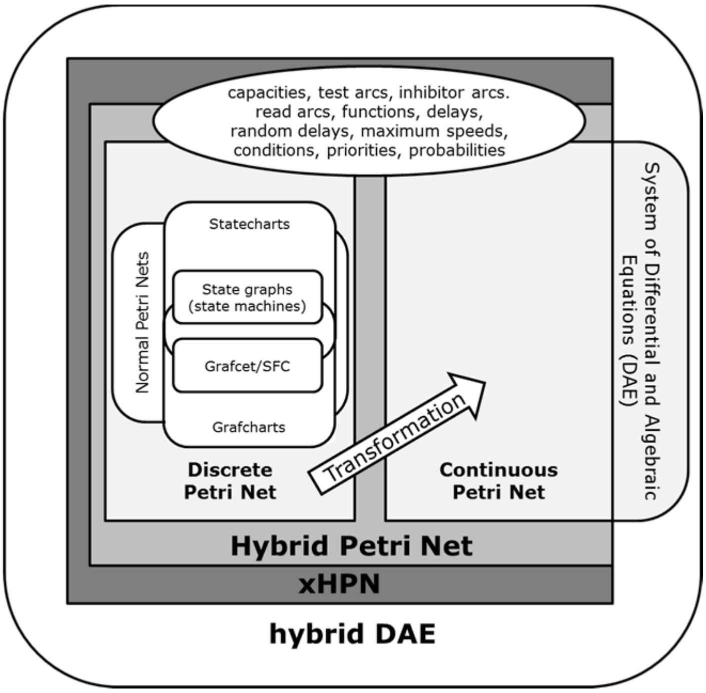


Figure 2.3: Relationships between the different formalisms

The relationships between the mentioned concepts are displayed in Figure 2.3. To enable modeling of biological systems with Petri nets in Modelica, the existing libraries have to be extended by the following aspects:

- Transfer of the discrete Petri net concept to a continuous one,
- Support of arcs with (functional) weightings,
- Support of test-, inhibitor, and read arcs,
- Support of (different) conflict resolutions (random decisions),
- Combination of discrete and continuous Petri net elements to hybrid Petri nets.

3 BASICS

This chapter comprises all basic techniques and methods that are necessary to accomplish the developed modeling process as depicted in Figure 5.1. At first, the object-oriented modeling language Modelica is introduced which is used to build up mathematical models of biological processes based on the xHPN formalism specified in Chapter 4. The Modelica language satisfies all requirements (see Section 6) and allows the programming of places and transitions of xHPNs. The Modelica constructs and principals used to implement the components of xHPNs are subsequently explained. Afterwards, optimization methods are introduced which are needed to perform parameter estimation and process optimization. Parameter estimation is used to find the parameters of a model if the concrete experiments are too expensive, too inaccurate, or even unfeasible. However, process optimization is performed based on a verified model to gain an optimal control of the biological process in order to achieve, for example, maximum product from the cells. Finally, methods for sensitivity analysis are introduced that help reduce the complexity of the model and give some indication of the verification.

3.1 THE MODELICA LANGUAGE

This section introduces the Modelica language constructs used to implement places, transitions, and arcs of the xHPN concept developed for biological applications (xHPNbio, see Section 5.2). For the syntactic meta symbols the extended Backus-Naur form is used (Scowen 1993). The object-oriented concept of Modelica is based on **classes**. From an implemented class any numbers of objects can be generated which are said to be **instances**. A Modelica class usually consists of a variable declaration part and an equation section which contains the equations for the declared variables. Example 3.1 shows a typical class structure with an upper part for the declarations and a lower part for the corresponding equations. A class can also contain **annotations**. Annotations are additional information associated with the model to specify, for example, the graphical representation, documentation text, version handling, code generation, simulation, or the graphical user interface. Annotations are implemented by the following syntax

```
annotation(annotationElements);
```

Example 3.1

```
class className
  declaration1
  declaration2
  ...
equation
  equation1
  equation2
  ...
end className;
```

3.1.1 VARIABLES

Variables, also called **components**, can either be declared by a built-in data type of Modelica (`Boolean, Integer, Real, String, enumeration(…)`) or by an instance of another class. It has to be distinguished between discrete-time and continuous-time variables. **Discrete-time** variables can change their values only at event instants (see Section 3.1.6) while **continuous-time** variables can change their values at any time.

Variables can be provided with the following prefixes that gives them specific properties:

- `constant`: the constant prefix determines that the variable never changes its value.
- `parameter`: the parameter prefix determines that the variable is constant during a simulation run but can be changed between two simulation runs to modify the model behavior. It is a static variable that is initialized once and never changed thereafter.
- `input`: the input prefix determines that the equation for such a variable is not provided in the model but rather by connected components.
- `output`: the output prefix determines that the value of such a variable is utilized by connected components.
- `discrete`: the discrete prefix determines that such a variable is a discrete-time variable.
- `inner`: the inner prefix declares a property that should be common to all components of a model and, hence, accessible from within those components.
- `outer`: the outer prefix references an element instance with the same name but using the prefix `inner`. The lookup is performed through the instance hierarchy instead of through the class nesting hierarchy. (Inner and outer components are used to define common

properties for the animation of a Petri net (see Section 6.2) or to model a fermentation process by which the volume is changed during fermentation. Then, the current volume is needed within several model components to recalculate the concentrations of substances (see Section 6.2).)

The default start values of variables are zero and `false` in the case of Booleans. Other start values have to be set by the `start` attribute within brackets after the name of the variable. Example 3.2 comprises some variable declarations with prefixes and specified start values.

Example 3.2

```
parameter Real realA=6.7;
Integer intB(start=8);
parameter String strC="Hello World";
Boolean boolD(start=true);
discrete Integer intE;
constant Real realF=9.81;
```

Variables can have higher dimensions than one, so-called **arrays**. Arrays are a collection of variables all of the same type. They are declared with the dimension size within square brackets after the variable name. If the size of an array is unknown, the brackets contain colons. This is mainly used for input arguments of functions (see Section 3.1.4).

Example 3.3

```
Real v1[3]={1,2,3};          //3-dimensional vector
Real m1[3,3]={{1,2,3},{3,4,5},{7,8,9}};   //3x3 matrix
Real v2[:];              //vector v2 has unknown size
```

Several built-in functions related to arrays are available. The following table summarizes some of them.

Table 3.1: Built-in functions related to arrays

`zeros(n1,n2,...)`	Returns a $n1 \times n2 \times$ ... matrix with zero-elements.
`fill(s,n1,n2,...)`	Returns a $n1 \times n2 \times$ ... matrix which is filled with the scalar `s`.
`size(A,i)`	Returns the size of the dimension `i` of the array `A`.
`sum(A)`	Returns the sum of all elements of the array `A`.

3.1.2 EQUATIONS

Equations are more powerful than assignment statements known from other programming languages like C++ or Java. The equation

```
a+b=c;
```

can be used in three different ways according to assignment statements

```
a:=c-b;
b:=c-a;
c:=a+b;
```

i.e. the variable `a` can be computed from `c` and `b`, the variable `b` can be computed from `c` and `a`, or the variable `c` can be computed from `a` and `b`. To distinguish between equations and assignment statements, the latter are expressed by `:=` in Modelica and only allowed in algorithm sections or functions (see Section 3.1.3 and 3.1.4) but not in equation sections.

Equations can be classified into four different groups (Fritzson 2004, Modelica Association 2010):

- **Normal equations** which are part of equation sections.
- **Declaration equations** which are part of variable declarations.
- **Modification equations** which are used to modify attributes of variables.
- **Binding equations** comprise declaration equations and modification equations.
- **Initial equations** which are part of initial equation sections or used to modify the start attribute.

NORMAL EQUATIONS

Normal equations appear in the equation section of a model which begins with the keyword `equation` (see Example 3.1). They comprise the following types of equations

- **Simple-equality-equations** with the syntax

  ```
  simpleExpression "=" expression
  ```

  Simple-equality-equations determine an equality relationship between two expressions which is well-known from mathematics.

- **For-equations** with the syntax

  ```
  for forIndices loop
     { equation ";" }
  end for ";"
  ```

  For-equations can express repetitive equation structures in a compact manner.

- **Connect-equations** with the syntax

  ```
  connect "(" componentReference "," componentReference ")" ";"
  ```

  Connect-equations generate connections between two components to enable interaction (see Section 3.1.4).

- **If-equations** with the syntax

```
if expression then
  { equation ";" }
{ elseif expression then
   { equation ";" }
}
[ else
  { equation ";" }
] end if ";"
```

If-equations are conditional equations which can contain discrete-time variables as well as continuous-time variables. If the variables of the if-equations are not specified with `parameter` or `constant`, the else part has to be included. Each part (`if`, `elseif`, `else`) must have the same number of equations.

- **When-equations** with the syntax

```
when expression then
   { equation ";" }
{ elsewhen expression then
   { equation ";" } }
end when ";"
```

When-equations are conditional equations which can only contain discrete-time variables in contrast to if-equations. The equations within a when-equation are only active at event instants (see Section 3.1.6).

DECLARATION EQUATIONS

Declaration equations appear in the declaration part of a model, usually, in term of parameter and constant declarations, e.g.

```
parameter Integer a=5;
constant Real g=9.81;
```

MODIFICATION EQUATIONS

Modification equations also appear in the declaration part of a model. They are used to modify the default values of attributes of a variable, e.g.

```
Real b(start=8.87);
```

(see also Example 3.7).

INITIAL EQUATIONS

Initial equations define the values of variables at the initial time, usually `time=0`, to compute their evaluation over time. This can either be done by setting the `start`-attribute (see Example 3.2) or by equations in the initial equation section (see Example 3.4).

Example 3.4

```
class className
  declaration1
  declaration2
  ...
equation
  equation1
  equation2
  ...
initial equation
  initialEquation1
  initialEquation2;
  ...
end className;
```

3.1.3 ALGORITHMS

Besides the mentioned equation and initial equation sections, a model can also have an algorithm section introduced by the keyword `algorithm` (see Example 3.5). It consists of a sequence of statements which are executed in the order of their appearance.

Example 3.5

```
class className
  declaration1
  declaration2
  ...
equation
  equation1
  equation2
  ...
algorithm
  statement1
  statement2;
  ...
end className;
```

Additionally, the variable has to be on the left-side of an assignment and the assigned value is on the right-side. To distinguish between equations and statements, the assignment operator is chosen to be "`:=`", e.g. `y:=8*x-5`, whereby `y` is the variable and `8*x-5` is the assigned value.

The following types of statements can be used within algorithm sections

- **Simple-assignment-statements** with the syntax

  ```
  componentReference ":=" expression
  ```

  Simple-assignment-statements have on the left-side a variable and on the right-side the assigned value; i.e. the `expression` is evaluated and stored in the variable `componentReference`.

- **For-statements** with the syntax

  ```
  for forIndices loop
     { statement ";" }
  end for ";"
  ```

  For-statements are a compact manner to express iterations.

- **While-statements** with the syntax

  ```
  while expression loop
       {statement ";" }
  end while ";"
  ```

  While-statements are a compact manner to express iterations by which the range of the iteration variable cannot be expressed in a closed form.

- **Break-statements** with the syntax

  ```
  break;
  ```

  Break-statements terminate the execution of for- and while-statements.

- **If-statement** with the syntax

  ```
  if expression then
    { statement ";" }
  { elseif expression then
    { statement ";" }
  }
  [ else
    { statement ";" }
  ] end if ";"
  ```

  If-statements express conditional assignments.

- **When-statement** with the syntax

  ```
  when expression then
      { statement ";" }
  { elsewhen expression then
      { statement ";" } }
  end when ";"
  ```

  When-statements are conditional assignments. The statements within a when-statement are only active when the `expression` becomes true, i.e. they are only executed at event instants (see Section 3.1.6).

3.1.4 Specialized Classes

The key concept of the Modelica language is the previously introduced class. This general class concept founds the basis for specialized classes which have the same properties as a class, apart from restrictions, but also additional properties that make them usable under appropriate conditions. They offer a way to make the Modelica code easier to read and maintain. The specialized classes are:

– `record`: a record is a specialized class to define data structures without behavior. Equations are not allowed within a record definition.

– `type`: a type is a specialized class to define aliases or extensions of predefined types (`Real`, `Integer`, `Boolean`, `String`, `enumeration(…)`), records, or arrays.

– `model`: a model is a specialized class that is identical to the basic class concept with no restrictions or additional properties.

– `block`: a block is specialized class with the same properties of a model except the restriction that every variable must be either input or output, i.e. all variables have to be declared with the prefixes `input` or `output`.

– `function`: a function is a specialized class to implement mathematical functions. The inputs of a function have to be prefixed by `input` and the results by `output`.

– `connector`: a connector is a specialized class to define the variables that are interchanged within a connection between two components. No equations are allowed in a connector.

– `package`: a package is a specialized class to organize and structure Modelica classes and specialized classes. It can contain the declarations of classes, specialized classes, and constants while parameters and variables cannot be declared in a package.

The specialized classes `model`, `block`, `function`, `connector,` and `package` are used in the Petri net library and are discussed in more detail hereafter.

Model

The specialized class `model` has the same properties as a class and is used for modeling purposes. Three different equation types can occur in a model: discrete, differential, and algebraic equations. Example 3.6 contains these equation types.

Example 3.6

```
model example
  parameter Real a=0.8;
  parameter Real b=1.78;
  Real c(start=6.7);
  Real d;
  discrete Boolean e;
equation
  der(c)=-a*c/(b+c);   //Differential equation
  d=-3*c-7.2;          //Algebraic equation
  when c<3.3 then      //Discrete equation
    e=true;
  end when;
end example;
```

The equation

```
der(c)=-a*c/(b+c);
```

is a differential equation and the corresponding variable `c` is a continuous-time variable. The operator `der()` accesses the derivative according to time. However, if an equation only involves algebraic formulas and no derivatives, it is an algebraic equation. In Example 3.6 the equation

```
d=-3*c-7.2;
```

is an algebraic equation while the equation

```
when c<3.3 then
   e=true;
end when;
```

is a discrete equation because the discrete-time variable `e` is only recalculated when the condition `c<3.3` becomes true, i.e. only at event instants (see Section 3.1.6). Equation systems that contain algebraic as well as differential equations are called **differential algebraic equation systems** (DAEs) and if discrete equations also appear they are called **hybrid differential algebraic equation systems** (hybrid DAEs) (see Section 3.1.7).

BLOCK

A `block` has the same properties as a model with the restriction that all variables have to be either declared by the prefix `input` or `output` (see Example 3.7). If a block is used in another component, a binding equation has to be provided for each variable with the `input`-prefix to guarantee a balanced model (see Section 3.1.5). In Example 3.7 an instance of the block `blockExample` is used in the model `modelExample`. The variable `s` of the block has the output-prefix and, hence, the equation is provided in the block. However, the variable u has the input-prefix and the equation has to be defined by a binding equation when the block is used. It is defined in the model by the equation u=time within brackets after the name of the block instance.

Example 3.7

```
block blockExample
  input Real u;
  output Real s;
equation
  s=sin(u);
end blockExample;

model modelExample
  Real v;
  blockExample exSin(u=time, s=v);  //binding equation for u
end modelExample;
```

FUNCTION

The specialized class function is used to implement mathematical algorithms with a sequence of statements. No equation sections are allowed in functions only algorithm sections. A typical structure of a function is outlined in Example 3.8.

Example 3.8

```
function functionName
  input typeI1 in1;
  input typeI2 in2; ...
  output typeO1 out1;
  output typeO2 out2; ...
protected
  <local variables>
  ...
algorithm
  ...
  <statements>
  ...
end functionName;
```

The first part of a function declares inputs and outputs using the keywords input and output. After the keyword protected, local variables can be declared neither input nor output of the function. The algorithmic procedure starts with the keyword algorithm; thereby, all types of statements, mentioned in Section 3.1.3, can be used. The function of Example 3.9 calculates the sum of vector elements. The input is a vector of unknown size which is identified with the operator "[:]" and the output is the scalar y.

Example 3.9

```
function functionExample
  input Real x[:];
  output Real y;
algorithm
  y:=0;
  for i in 1:size(x,1) loop
      y:=y+x[i];
  end for;
end functionExample;
```

A function can be called from within classes, models, blocks, or other functions by the prompt

```
(out1, out2, ...) = functionName(inp1,inp2,...);
```

The function of Example 3.9 can be called by

```
z = functionExample({1,2,3,8});
```

It is also possible to integrate functions written in C or FORTRAN 77 in the Modelica environment. This can be done with the aid of the key word `external` followed by the respective language within quotation marks. The function `random` of Example 3.10 generates a uniformly distributed random number by an external C-function. It has no inputs and returns the random number `x`. The `Include` annotation specifies the C-file that contains the respective implementation.

Example 3.10

Modelica function:

```
function random
  output Integer x;
  external "C" annotation(Include = "#include <random.c>",
                             __Dymola_pure=false);
end random;
```

C-function:

```
#include <stdlib.h>
#ifndef RANDOM_C
#define RANDOM_C
int random()
{
   static int called=0;
   int i;
   if (!called) {
      srand((unsigned) time(NULL));
      called=1;
   }
   return rand();
}
#endif
```

Modelica functions are according to the specification (Modelica Association 2010) **pure** which means that

- Modelica functions are mathematical functions that always return the same result when they are called with the same input arguments.
- Modelica functions are side-effect free with respect to the internal Modelica state.
- Exception: An **impure function** is either an impure external function or a Modelica function which calls an impure external function. An impure function returns different values by calling with the same inputs. They can be called from within a when-equation or a when-statement. The function `random` of Example 3.10 is an example of an impure function which has no input arguments and always returns another value when it is required. The Dymola tool identifies an impure function by the vendor-specific annotation `__Dymola_pure=false`.

Connector

The specialized class `connector` is used to implement connectors. Connectors declare variables that are interchanged between components. A possible structure of a connector is outlined in Example 3.11.

Example 3.11

```
connector connectorName
  input typeI1 in1;
  input typeI2 in2;
  ...
  output typeO1 out1;
  output typeO2 out2;
  ...
end connectorName;
```

Two connectors can be connected by a connector equation (see Section 3.1.2), e.g.

```
connect(connector1, connector2);
```

The prefixes `input` and `output` define the location of the corresponding equation and guarantee in this manner balanced modeling (see Section 3.1.5). When a variable is provided with the `input`-prefix, the equation is part of the connected component while the equation of an `output` variable is located in the component where the connector is used.

PACKAGE

The specialized class package is used to structure and organize Modelica classes. The package of Example 3.12 comprises the examples of this and the previous sections.

Example 3.12

```
package packageExample

function random
  output Integer x;
  external "C" annotation(Include = "#include <random.c>",
     __Dymola_pure=false);
end random;

block blockExample
  input Real u;
  output Real s;
equation
  s=sin(u);
end blockExample;

model modelExample
  Real v;
  example2_6a exSin(u=time, s=v);//binding equation for u
end modelExample;

model example
  parameter Real a=0.8;
  parameter Real b=1.78;
  Real c(start=6.7);
  Real d;
  discrete Boolean e;
equation
  der(c)=-a*c/(b+c);   //Differential equation
  d=-3*c-7.2;          //Algebraic equation
  when c<3.3 then      //Discrete equation
    e=true;
  end when;
end example;

function functionExample
  input Real x[:];
  output Real y;
algorithm
  y:=0;
  for i in 1:size(x,1) loop
      y:=y+x[i];
  end for;
end functionExample;

end packageExample;
```

3.1.5 Balanced Models

A Modelica model can only be simulated if the number of variables is equal to the number of equations. Since Modelica 3.0 there are additional restrictions so that every model must be **locally balanced**, i.e. the number of variables and equations must be identical on every hierarchical level (Olsson et al. 2008). If instances of locally balanced models are used by connecting them to a model, this model will automatically have the same number of variables and equations. It is said to be **globally balanced**. Every model or block that can be simulated is globally balanced. To guarantee globally balanced models, the following conditions have to be maintained:

- The number of flow variables in a connector must be equal to the number of non-causal, non-flow variables (variables without the prefixes `flow`, `input`, `output`, `parameter` and `constant`).
- The number of equations of a model must be identical to the number of variables, excluding input and flow variables.
- When a model is used by making an instance of it, all missing equations must be either provided by connecting connectors or by a modification equation for every non-connector variable with an `input` prefix.

A detailed description and definition of locally and globally balanced models can be found in (Olsson et al. 2008) and (Modelica Association 2010).

3.1.6 Discrete Event and Hybrid Modeling

An **event** is a discrete change when something happens and occurs at a certain point in time. Additionally, events in Modelica have the following properties (Fritzson 2004)

- An event has zero duration.
- An event occurs when the corresponding **event condition** switches from `false` to `true`.
- When an event occurs, variables that are associated with the event are changed corresponding to their equations, i.e. a set of equations associated with the event becomes active.

This event behavior is engendered in Modelica by when-equations and when-statements (see Section 3.1.2 and 3.1.3).

```
when eventCondition then
   eventAction1;
   eventAction2;
   ...
end when;
```

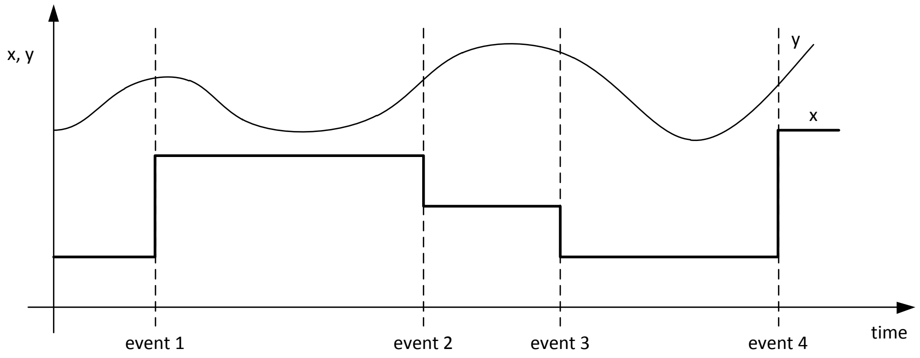


Figure 3.1: Discrete-time variable x and continuous-time variable y

Variables that change their values at event instants and keep constant otherwise, are called **discrete-time variables**. They have to be declared with the `discrete`-prefix (see Section 3.1.1). However, **continuous-time variables** usually change their values continuously over time. Figure 3.1 shows the time evolution of a discrete-time variable x and a continuous-time variable y. The discrete-time variable changes its value only at the four event instants while the continuous-time variable changes its value continuously entire time.

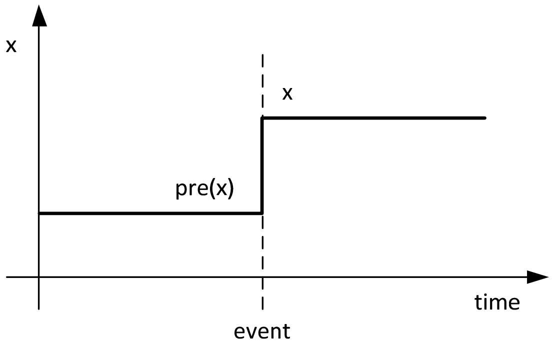


Figure 3.2: The function `pre(x)` obtains the value of the discrete-time variable `x` immediately before the event occurs

Events can be classified according to how they are generated into time events and state events. **Time events** are directly associated with Modelica built-in variable `time` which contains the current simulation time. An event condition of a time event can be, for example,

```
time >= eventTime
```

which triggers a time event when `time=eventTime` due to the switch of the event condition from `false` to `true` at this point in time. Because their occurrence time can be predicted in advanced, time events can be handled more efficiently by the simulator.

However, **state events** cannot be predicted in advanced because they are associated with changes in state variables (see Section 3.1.7). An example for event condition of a state event is

```
sin(x) > 0.7
```

which triggers an event each time when `sin(x)` becomes greater than 0.7, i.e. at the points in time when the condition switches from `false` to `true`.

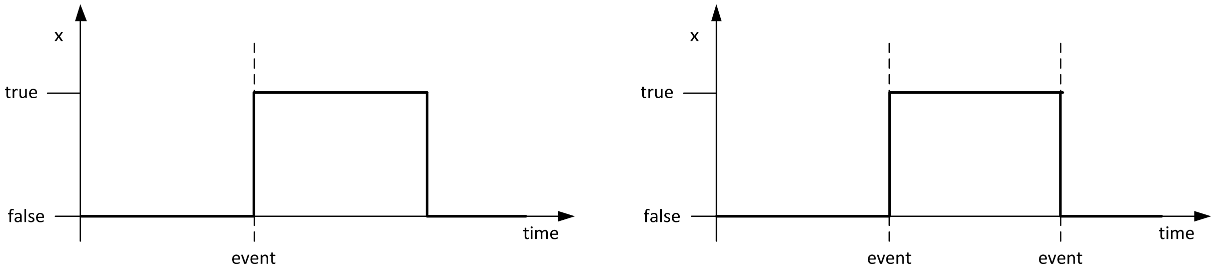


Figure 3.3: The function `edge(x)` detects an event when the variable `x` switches from `false` to `true` (left) and the function `change(x)` detects an event when the variable `x` switches from `false` to `true` or from `true` to `false` (right)

Furthermore, Modelica offers a way to generate, prevent, and detect events by several built-in functions. In addition, it is possible to access the value of a variable immediately before the event occurs. The following table summarizes some of these functions.

Table 3.2: Built-in functions related to events

`initial()`	Generates a time event at the beginning of a simulation, i.e. when the variable `time` is equal to the defined start time (usually `time=0`), it returns `true` and `false` otherwise.
`terminal()`	Generates a time event at the end of a successful simulation, i.e. when the variable `time` is equal to the defined stop time, it returns `true` and `false` otherwise.
`sample(start, interval)`	Generates time events at the times $start + i * interval$ $(i = 0,1,\dots)$.
`noEvent(expr)`	Prevents the generation of time and state events.
`pre(x)`	Obtains the value of the discrete-time variable `x` immediately before the event occurs (see Figure 3.2).
`edge(x)`	Detects the change of a Boolean variable. It returns `true` when `x` changes from `false` to `true`; i.e. it is equal to the condition `x and not pre(x)` (see Figure 3.3, left).

`change(x)`	Detects the change of a Boolean variable. But in contrast to the `edge()`-function, it return `true` if `x` switches from `false` to `true` or from `true` to `false`, i.e. it is equal to the condition `x<>pre(x)` (see Figure 3.3, right).
`reinit(x,expr)`	Reinitializes a state variable `x` with `expr` at an event instant. It can only applied within a when clause. (see also Section 3.1.7 and Example 3.13))

Example 3.13

The model of a bouncing ball below is an example of a hybrid system. Thereby, the movement of the ball is determined by the variable h which represents the height of the ball above the ground and the variable v for its velocity. Between two bounces, the ball moves continuously expressed by the differential equations for h and v while a discontinuous change occurs at every bounce on the ground modeled by a discrete `when` equation (see Figure 3.4). Then the velocity is reversed and slightly decreased which is achieved by the `reinit` operator. Moreover, an additional logic disables the bounces when the height is lower than the simulation tolerances. Then the ball does not fly anymore which is expressed by the variable `flying`.

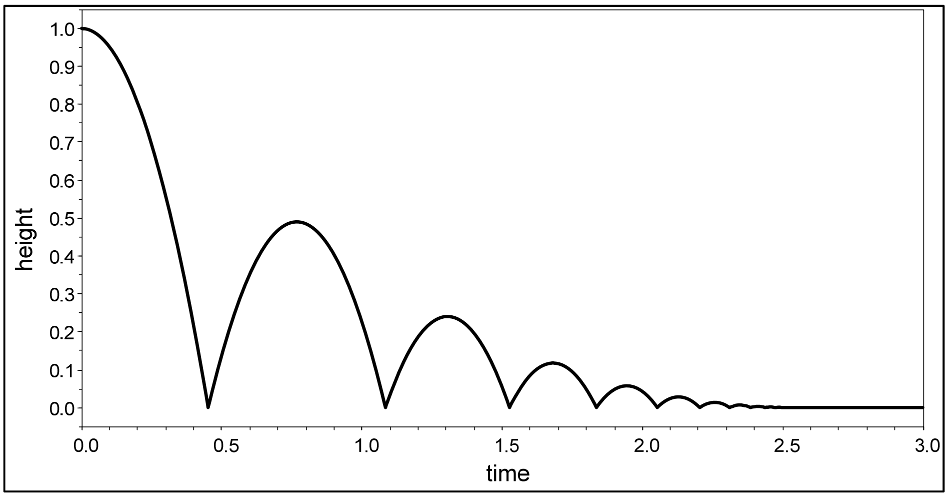


Figure 3.4: A bouncing ball (Example 3.13)

```
model bouncingBall
  parameter Real e=0.7;
  parameter Real g=9.81;
  Real h(start=1);
  Real v;
  Boolean flying(start=true);
  Boolean impact;
  Real v_new;
equation
  der(h) = v;
  der(v) = if flying then -g else 0;
  impact = h < 0;
  when {impact,h < 0 and v < 0} then
    v_new = if edge(impact) then -e*pre(v) else 0;
    flying = v_new > 0;
    reinit(v, v_new);
  end when;
end bouncingBall;
```

3.1.7 Mathematical Representation of Modelica Models

Before a Modelica model can be simulated, the hierarchical model has to be translated to a flat system of **differential and algebraic equations** (DAEs). The term **flat** means that the object-oriented structure is abolished so that no hierarchy remains but rather a set of equations.

The implicit form of the DAE system is given by

$$f(x(t), \dot{x}(t), u(t), y(t), t, p) = 0$$

Eq. 3-1

The vector $x(t)$ comprises all variables for which the time-derivative $\dot{x}(t)$ also occur in the model, called **state variables**, $u(t)$ comprises all input variables, $y(t)$ comprises all algebraic output variables, t is the time, p comprises all parameters, and f is a set of differential and algebraic equations.

Modelica supports not only continuous simulation but also discrete and hybrid modeling. Hence, a system of DAEs is not sufficient to represent a Modelica model. It has to be expanded to **hybrid DAEs** including discrete, differential, and algebraic equations. This can be obtained by adding the vector $q(t)$ of discrete-time variables. The **continuous-time behavior** of a Modelica model is then described by adding $q(t)$ to Eq. 3-1

$$f(x(t), \dot{x}(t), u(t), y(t), q(t), t, p) = 0,$$

Eq. 3-2

whereby the discrete variables are constant during the continuous phase. A fixed causality is achieved by converting the equations in Eq. 3-2 to assignments valid for the continuous phase

$$z(t) = \begin{pmatrix} \dot{x}(t) \\ y(t) \\ q(t) \end{pmatrix} := \begin{pmatrix} f_{\dot{x}}\big(x(t), u(t), pre\big(q(t)\big), t, p\big) \\ f_y\big(x(t), u(t), pre\big(q(t)\big), t, p\big) \\ f_q\big(x(t), u(t), pre\big(q(t)\big), t, p\big) = const \end{pmatrix},$$

Eq. 3-3

whereby $pre\big(q(t)\big)$ accesses the predecessor values of the discrete-time variables which are in the continuous phase equivalent to the values of $q(t)$ (see Section 3.1.6).

The **discrete-time behavior** is mainly described by events. An event occurs at $t_e$when at least one of the condition expressions $c(t_e)$, e.g. from `if` or `when` constructs, switches from `false` to `true`. These condition expressions are a subset of the Boolean discrete-time variables $q^B(t_e)$ $\left(c(t_e) \subseteq q^B(t_e)\right)$. An event is fired by execution of its associated behavior. Therefore, the whole system has to be determined by the function in Eq. 3-3 at t_e to ensure the synchronicity of the equations

$$z(t_e) = \begin{pmatrix} \dot{x}(t_e) \\ y(t_e) \\ q(t_e) \end{pmatrix} := \begin{pmatrix} f_{\dot{x}}\left(x(t_e), u(t_e), pre\left(q(t_e)\right), t_e, p\right) \\ f_y\left(x(t_e), u(t_e), pre\left(q(t_e)\right), t_e, p\right) \\ f_q\left(x(t_e), u(t_e), pre\left(q(t_e)\right), t_e, p\right) \end{pmatrix}. \qquad \text{Eq. 3-4}$$

It is not sufficient to solely determine the discrete-time variables with the function f_q (Braun 2010, Braun et al. 2010). Additionally, discontinuous changes of continuous-time variables can occur caused by the `reinit`-operator in Modelica (see Section 3.1.6). These discontinuous changes are only triggered by events at t_e and the states of the continuous variables are redetermined with the aid of the following assignment

$$x(t_e) := f_x\left(pre\left(x(t_e)\right), \dot{x}(t_e), u(t_e), y(t_e), q(t_e), pre\left(q(t_e)\right), t_e, p\right). \qquad \text{Eq. 3-5}$$

Thereby, all variables in $x(t_e)$ that are not affected by the `reinit`-operator remains unaltered by the function f_x. Afterwards, a recalculation of the system in Eq. 3-4 is necessary if $pre\left(x(t_e)\right) \neq x(t_e)$ or $pre\left(q(t_e)\right) \neq q(t_e)$. This procedure is called **event iteration**.

The hybrid DAE system consists of a combination of Eq. 3-3, Eq. 3-4, and Eq. 3-5 which has the ability to represent a Modelica model comprising of discrete, differential, and algebraic equations.

The whole process from a Modelica model to executable simulation code is outlined in Figure 3.5. After flatting to hybrid DAEs, each of the resulting equations has to be assigned to a variable and sorted topologically according to their dependencies. Therefore, the equations are transformed to a Block Lower Triangular (BLT) form by Tarjans algorithm (Tarjan 1972). This form reveals the structure of the problem and decomposes it to a set of sub-problems which can be solved in sequence. If there is a non-scalar block on the diagonal of the BLT-matrix, this indicates the appearance of an **algebraic loop,** also called **strong component**. The equations involved in an algebraic loop have to be solved simultaneously. In the case of discrete-time variables this is not possible and the algebraic loop has to be cut by hand which is done by using the `pre` operator for one or more involved discrete-time variables (see Example 3.14).

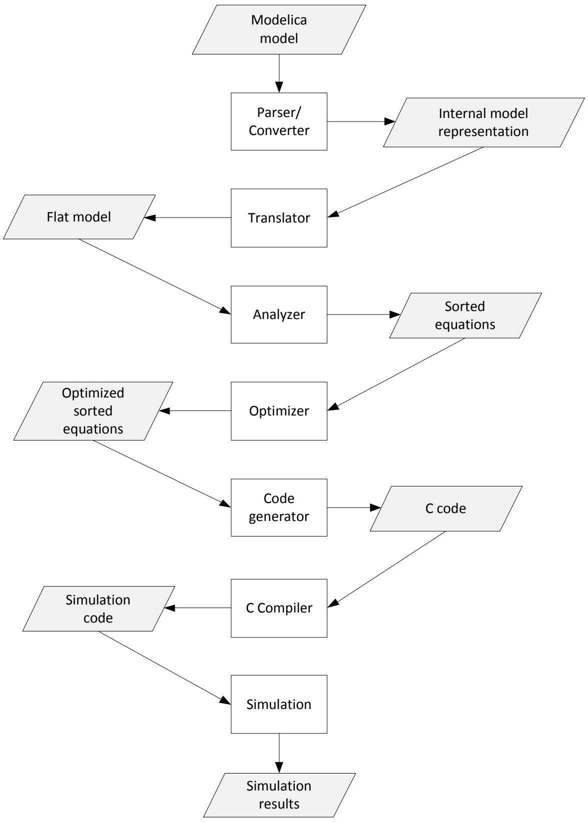


Figure 3.5: Necessary steps from a Modelica model to executable simulation code (Fritzson 2004)

Example 3.14

```
model algebraicLoop
  Boolean bool1(start=true);
  Boolean bool2(start=false);
equation
  when bool1 then
    bool2=true;
  end when;
  when bool2 then
    bool1=false;
  end when;
end algebraicLoop;
```

This model generates an algebraic loop because the variable `bool1` is needed to calculate `bool2` and vice versa. This loop can be cut, for example, by putting a `pre` around the when-condition `bool1`.

```
model algebraicLoopCut
  Boolean bool1(start=true);
  Boolean bool2(start=false);
equation
  when pre(bool1) then
    bool2=true;
  end when;
  when bool2 then
    bool1=false;
  end when;
end algebraicLoopCut;
```

After sorting the equations, optimization methods, like algebraic simplification algorithms and symbolic index reduction methods, are applied to eliminate equations and solve the equation system efficiently by numerical methods. Afterwards, the C-code is generated and compiled to executable simulation code.

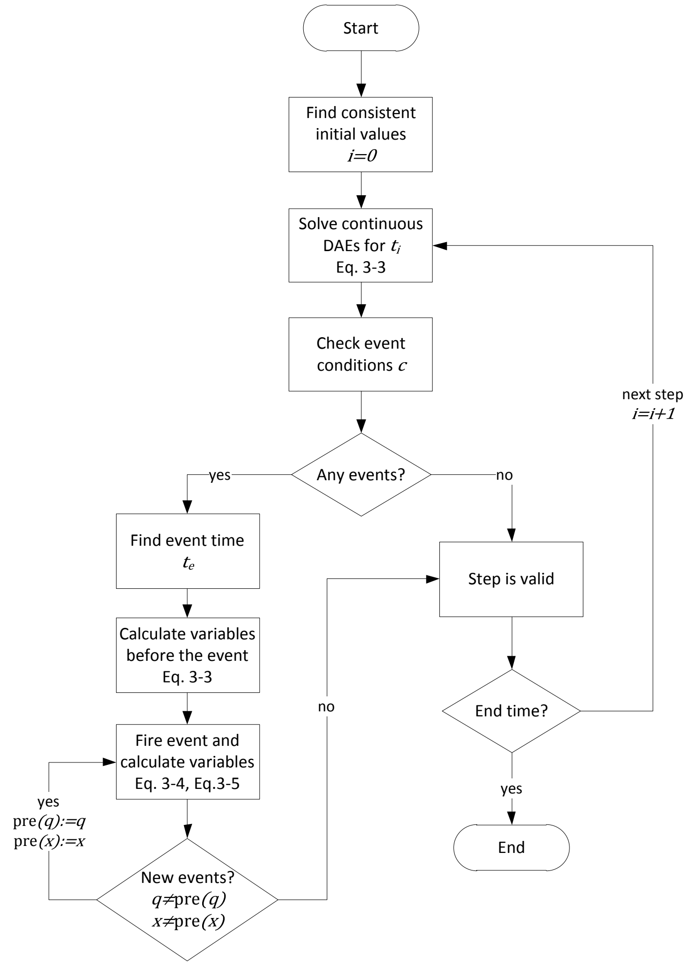

Start
Find consistent initial values i=0
Solve continuous DAEs for t_i Eq. 3-3
Check event conditions c
next step i=i+1
Any events?
yes
no
Find event time t_e
Step is valid
Calculate variables before the event Eq. 3-3
no
End time?
Fire event and calculate variables Eq. 3-4, Eq.3-5
yes pre(q):=q pre(x):=x
yes
End
New events? q≠pre(q) x≠pre(x)

Figure 3.6: Solution process of hybrid DAEs (cp. Braun et al. 2010)

The simulation of a hybrid model is based on the solution of the hybrid DAEs which is performed in the following way (Modelica Association 2010):

1. The consistent set of initial values has to be found according to the given constraints.
2. The continuous DAEs in Eq. 3-3 are solved by a numerical integration method. The discrete-time variables q are kept constant.
3. All condition expressions c are observed. If one variable switches its value, the integration is stopped, the exact event time t_e is determined, the values of the variables before the event are calculated, and an event is triggered (time events are scheduled in advanced and can be treated in another way (see Section 3.1.6)).
4. The hybrid DAEs in Eq. 3-4 and Eq. 3-5 are solved at the event instant. This system is resolved as long as $pre(q) = q$ and $pre(x) = x$ (event iteration***)***.

5. Afterwards the integration is restarted with a consistent set of restart values (go back to step 2).

The solution algorithm of hybrid DAEs is also depicted in Figure 3.6. Modelica tools usually offer a wide range of methods to solve the continuous DAEs; the most common and frequently-used is DASSL (Differential Algebraic System Solver) which was first introduced by Petzold in 1982 (Petzold 1982).

3.2 Optimization Methods

Once a model is constructed, the task is to estimate the model parameters, hereafter called **structure parameters**. Thereby, the parameters have to be chosen so that the model reproduces the given experimental data in the best possible way. This procedure is called **parameter estimation** (PE) or **inverse problem**; thereby, the latter indicates that the structure parameters are identified from measurements (Ueckerdt 1978).

Another aspect is the optimization of processes, which underlie the verified model, in such a way that, for example, a product of the regarded organism is maximized according to the **process parameters**. This procedure is called **process optimization** (PO).

Both steps, PE and PO, of the modeling process engender an optimization problem:

PE: Minimize an **objective function** $Q(\mathscr{p})$ which represents the goodness of a **structure parameter set** $\mathscr{p}$,

PO: Maximize/Minimize an **objective function** $Q(\mathscr{z})$ which represents the goodness of a **process parameter set** $\mathscr{z}$.

These objective functions in combination with an xHPN model are not only non-linear but also usually discontinuous and not-differentiable because of the discrete changes of hybrid Petri nets. Due to the non-differentiability, the usage of methods which determine decent directions from derivatives of the objective function is not possible. However, **derivative-free methods** do not require derivatives to minimize the objective function and, hence, such methods are applicable and used within this study.

It has to be distinguished between local and global methods. **Local methods** try to find the minimum starting from a given point. Thereby, only local information about the objective function from the neighborhood of the current approximation is used to update the approximation; hence, the global structure of an objective function is unknown to a local method. Additionally, it is usually expected that such methods converge to the local minimum which is nearest to the starting point.

However, the objective of **global methods** is to find the global minimum of the optimization problem usually in the presence of multiple local minima by seeking the whole search space. Global methods can be further divided into deterministic and stochastic methods. **Deterministic methods** always achieve the same result starting from the same setting while **stochastic methods** involve random mechanisms.

Local and global methods can also be combined to **hybrid methods** which should avoid the high computational costs of global methods due to their slow convergence near the minimum. Additionally, the entrapment in a local minimum should be prevented which is often the drawback of local methods.

Hereafter some selected local and global derivative-free optimization methods are introduced which are used within this study. Moreover, these methods are combined to hybrid optimization approaches to enhance the optimization process. All methods can be applied for PE as well as for PO so that a general optimization problem is expressed by

$$min\, Q(x) \quad x \in \mathcal{X} \subseteq \mathbb{R}^n, Q\colon \mathcal{X} \to \mathbb{R}$$

subject to

$$\begin{gathered} x^l \leq x \leq x^u \\ model, \end{gathered} \qquad \text{Eq. 3-6}$$

whereby

$$x = \begin{cases} p & if\ PE\ is\ applied\ \mathcal{X} = \mathcal{P} \subseteq \mathbb{R}^n \\ z & if\ PO\ is\ applied\ \mathcal{X} = \mathcal{Z} \subseteq \mathbb{R}^n. \end{cases}$$

Thereby, it is assumed that the objective function is minimized; in the case of PO, a maximization is also possible, then the problem is converted to a minimum problem by changing the signs. Furthermore, the objective function is subject to the constructed model and to upper and lower bounds for each parameter, known from, for example, experiments, literature or laws. Regarding the optimization process, it has to be distinguished between local and global minima. Therefore, the following definition is introduced.

Definition 3.1 ((strict) local/global minimum)

Suppose $Q\colon \mathcal{X} \to \mathbb{R}$ with $\mathcal{X} \subseteq \mathbb{R}^n$ is an objective function. A parameter set $x^* \in \mathcal{X}$ is a

- **global minimum** of Q, if $Q(x^*) \leq Q(x) \quad \forall\, x \in \mathcal{X}$.

- **strict global minimum** of Q, if $Q(x^*) < Q(x) \quad \forall\, x \in \mathcal{X}$ with $x \neq x^*$.
- **local minimum** of Q if there is a neighborhood $\mathcal{N}$ of x^* such that $Q(x^*) \leq Q(x) \quad \forall x \in (\mathcal{X} \cap \mathcal{N})$.
- **strict local minimum** of Q if there is a neighborhood $\mathcal{N}$ of x^* such that $Q(x^*) < Q(x) \quad \forall x \in (\mathcal{X} \cap \mathcal{N})$ with $x \neq x^*$.

To recognize a local minimum, the following necessary and sufficient conditions are proven. These conditions use information about the **gradient**

$$\nabla Q(x) = \begin{pmatrix} \frac{\partial Q(x)}{\partial x_1} & \frac{\partial Q(x)}{\partial x_2} & \cdots & \frac{\partial Q(x)}{\partial x_n} \end{pmatrix} \qquad \text{Eq. 3-7}$$

and the **Hessian** of the objective function

$$\nabla^2 Q(x) = \begin{pmatrix} \frac{\partial^2 Q(x)}{\partial x_1^2} & \frac{\partial^2 Q(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 Q(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 Q(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 Q(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 Q(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 Q(x)}{\partial x_n \partial x_1} & \frac{\partial^2 Q(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 Q(x)}{\partial x_n^2} \end{pmatrix}. \qquad \text{Eq. 3-8}$$

Theorem 3.1 (first-order necessary conditions (Nocedal and Wright 1999))

If x^* is a local minimizer and Q is continuously differentiable in an open neighborhood of x^*, then $\nabla Q(x^*) = 0$.

Theorem 3.2 (second-order necessary conditions (Nocedal and Wright 1999))

If x^* is a local minimizer of Q and $\nabla^2 Q(x)$ is continuous in an open neighborhood of x^*, then $\nabla Q(x^*) = 0$ and $\nabla^2 Q(x)$ is positive semi-definite.

Theorem 3.3 (second-order sufficient conditions (Nocedal and Wright 1999))

Suppose that $\nabla^2 Q(x)$ is continuous in an open neighborhood of x^* and that $\nabla Q(x^*) = 0$ and $\nabla^2 Q(x)$ is positive definite. Then x^* is a strict local minimizer of Q.

Example 3.15

Figure 3.7 shows the six-hump camel back function (Dixon and Szegö 1978)

$$f(x) = \left(4 - 2.1 \cdot x_1^2 + \frac{x_1^4}{3}\right) \cdot x_1^2 + x_1 \cdot x_2 + (-4 + 4 \cdot x_2^2) \cdot x_2^2$$

which is an example of a so-called **multimodal function**, i.e. a function with several minima. Within the bounded region

$$-3 \leq x_1 \leq 3, \qquad -2 \leq x_2 \leq 2$$

there are six minima, two of them are global

$$x_1^* = (-0.0898, 0.7126), \qquad x_2^* = (0.0898, -0.7126).$$

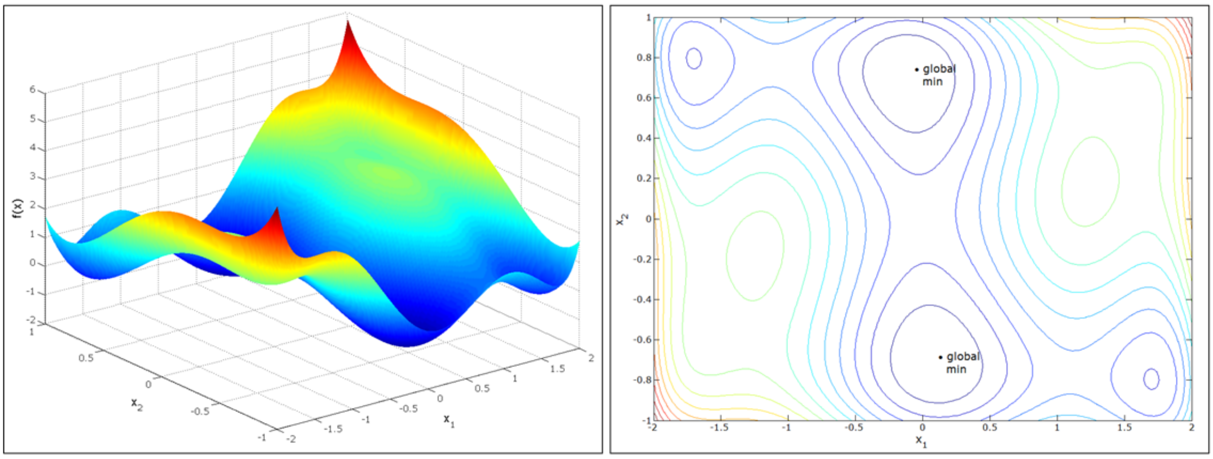


Figure 3.7: Six-hump camel back function with six minima in the bounded region; two of them are global minima; left: surface plot, right: contour plot

3.2.1 Local Optimization Methods

In this section two local methods are introduced to find the minimum of the optimization problem in Eq. 3-6 starting from a given parameter set x^0. Both algorithms cope without derivatives and, thus, belong to the general class of **direct search methods** which can be further divided to **pattern search methods** and **simplex methods**. At first, a pattern search method, the Hooke-Jeeves method, is introduced and afterwards a simplex method, the Nelder-Mead simplex method, is presented.

Hooke-Jeeves Method

The direct search method of Hooke and Jeeves was introduced in 1961 (Hooke and Jeeves 1961). It consists of two procedures: **exploratory moving** and **pattern search**. In the first, the parameter vector is changed locally by a positive and negative variation of one parameter at a time to obtain information in which direction the objective function decreases. This information is used in the second procedure to find the best direction for the minimization process. If the exploratory move was successful, i.e. the function value decreases, further progress may be possible in this direction; otherwise, the step size of the exploratory move has to be reduced.

The algorithm is outlined in Algorithm A1 (Appendix A1) which is taken from (Bell and Pike 1966). A detailed description of the Hooke-Jeeves method can be found in (Hooke and Jeeves 1961) and (Kolda et al. 2003). Furthermore, Torczon show global convergence of the Hooke-Jeeves method under specific assumptions (Torczon 1997).

Nelder-Mead Simplex Method

The Nelder-Mead simplex method was introduced in 1965 (Nelder and Mead 1965). It maintains at each step a non-degenerate **simplex**. This is a n-dimensional figure with $n+1$ vertices forming the convex hull. Each iteration starts with a simplex specified by its $n+1$ vertices and the associated values of the objective function. One or more test points are computed and a new simplex is generated according to the function values of these test points. The function values of the vertices of the new simplex satisfy then some form of descent condition compared to the previous simplex.

The algorithm is outlined in Algorithm A2 (Appendix A1) which is taken from (Lagarias et al. 1999). A detailed description of the Nelder-Mead simplex method can be found in (Lagarias et al. 1999). The Nelder-Mead simplex algorithm is one of the most popular optimization methods and is used in many numerical software packages like Matlab. However, no general convergence results are proven till now; only those for strictly convex functions in dimensions 1 and 2 are presented in (Lagarias et al. 1999) with various limitations for the 2-dimensional case. Furthermore, several negative examples are known for which the algorithm converge to a non-stationary point (McKinnon 1999).

3.2.2 Global Optimization Methods

In this section two global optimization methods are introduced which try to find the global minimum of the optimization problem in Eq. 3-6 by seeking the whole search space. The first - DIRECT method - is a deterministic approach and the second - evolution strategy - is performed in a stochastic manner. Thereby, the basic evolution strategy (ES), introduced by Schwefel (Schwefel 1995), is considered as well as a modified variant of it - covariance matrix adaption evolution strategy (CMAES) - which was introduced by Hansen and Ostermeier (Hansen and Ostermeier 2001).

DIRECT METHOD

The DIRECT (DIviding RECTangles) method was introduced by Jones et al. in 1993 (Jones et al. 1993). It is a deterministic method for seeking the global minimum of an objective function. The algorithm is a modification of the standard Lipschitz approach (e.g. Shubert 1972) which eliminates the specification of the Lipschitz constant, improves the convergence speed, and reduces the computational complexity. The main idea is to carry out simultaneous searches using all possible Lipschitz constants from zero to infinite to determine if a region of the search space should be broken into sub-regions during the current iteration. The Lipschitz constant can then be viewed as a weight between global and local search because the DIRECT method operates on both levels to enhance the convergence speed. When the global part finds the basin of convergence of the minimum, the local part exploits it, hence, the DIRECT method is also a hybrid approach.

The first step is to transform the search space to a n-dimensional unit hypercube by the lower and upper bounds of the parameters given in Eq. 3-6, thus, the algorithm works in a normalized space, and the original space is only used when the objective function is called. Then the objective function value of the midpoint c_1 of this hypercube is evaluated. This hypercube is divided into smaller hyper-rectangles by evaluating the objective function at the points

$$c_1 \pm \delta e_i, \;\; i = 1, \dots, n \qquad \text{Eq. 3-9}$$

where δ is one-third of the side length of the hyper cube and e_i is the ith unit vector. Thereby, the points with best function values should be located in the largest rectangles. This leads to the following rule: Let

$$w_i = min\big(Q(c_1 + \delta e_i), Q(c_1 - \delta e_i)\big) \quad i = 1, \dots, n \qquad \text{Eq. 3-10}$$

be the best function value sampled along dimension i. Divide the dimension with the smallest w_i into thirds so that $\mathrm{c}_1 \pm \delta e_i$ are the centers of the new hyper rectangles. This pattern is continued until all dimensions are split. Thereby, the next dimension is chosen by determining the next smallest w_i.

Afterwards the iteration loop begins with the identification of potentially optimal rectangles which are divided and sampled at their midpoints.

Definition 3.2 (potentially optimal (Jones et al. 1993))

Suppose that a unit hypercube is divided into m hyperrectangles. Let c_i denote the center point of the ith hyperrectangle, let d_i denote the distance from the center to the vertices, and let Q_{min} denote the current best objective function value. Let $\varepsilon > 0$ be a positive constant. A hyperrectangle j is said to be **potentially optimal** if there exists some $K > 0$ such that

$$Q(c_j) - Kd_j \leq Q(c_i) - Kd_i, \quad \forall\, i, \text{and}$$

$$Q(c_j) - Kd_j \leq Q_{min} - \varepsilon|Q_{min}|.$$

Thereby, the parameter ε is used to gurantee that the new best solution exceeds the current best solution by a specific amount. Experiments have shown that the value of ε has a minor effect on the results. A good setting should be 10^{-4} (Jones et al. 1993).

The potentially optimal rectangles are again divided into smaller rectangles and the corresponding function values are evaluated as just described. The process continues until at least one of the predefined abort criteria is fulfilled. An efficient implementation to identify potential optimal rectangles according to Definition 3.2 can be found in (Finkel 2003).

The algorithm is outlined in Algorithm A3 (Appendix A1) which is taken from (Jones et al. 1993). For a detailed description of the algorithm including its convergence and performance properties, refer to (Jones et al. 1993). Furthermore, the DIRECT method guarantees convergance to the globally optimal objective function value if the objective function is continuous – or at least continuous in a neighborhood of a global optimum. Due to the fact that if the number of iterations goes to infinity, the set of points sampled by the DIRECT method form a dense subset of a unit hypercube (Jones et al. 1993).

Evolution Strategy

The evolution strategy (ES) is a biologically inspired method founded by Rechenberg and Schwefel in the early 1970s (Rechenberg 1971, Schwefel 1975). It belongs to the class of **evolutionary algorithms** whose general structure is depicted in Figure 3.8.

ES bases on a collective learning process within a population of individuals. Each of these individuals represents a possible parameter set x of the optimization problem in Eq. 3-6. The initialization of the population is arbitrary and it is increasingly improved by the (probabilistic) processes **selection**, **mutation**, and **recombination**. The selection process prefers individuals with a higher **fitness value** to reproduce more often than those of lower fitness. The fitness value of an individual is identical to the value of the objective function $Q(x)$ with the corresponding parameter set (individual) x. Based on this information, ES makes use of Darwin's principle: "Survival of the fittest". The recombination process combines two or more parental individuals to produce new individuals and the mutation process changes the individuals at random to innovate the population.

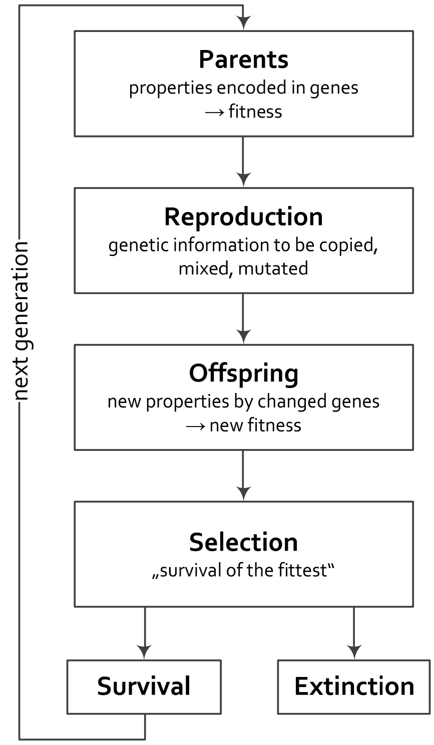

Parents
properties encoded in genes
→ fitness
Reproduction
genetic information to be copied, mixed, mutated
Offspring
new properties by changed genes
→ new fitness
Selection
„survival of the fittest"
Survival
Extinction
next generation

Figure 3.8: The general structure of evolutionary algorithms (Beyer 2001)

The following notions are used in this section

- $Q: \mathcal{X} \rightarrow \mathbb{R}$ is the objective function (Eq. 3-6),
- I is the space of individuals,
- $\Phi: I \rightarrow \mathbb{R}$ is the fitness function,
- $a \in I$ is an individual,
- $x \in \mathcal{X} \subseteq \mathbb{R}^n$ is the vector of parameters,
- $\mu \geq 1$ is the size of the parent population,
- $\lambda \geq 1$ is the size of offspring population, i.e. the number of individuals that are created by recombination and mutation at each generation,
- $A(g) = \{a_1(g), a_2(g), \dots, a_\mu(g)\}$ is the population at generation g with the individuals $a_i(g) \in I$,
- $r_{\Phi_r}: I^\mu \rightarrow I^\lambda$ is the recombination operator which generates the offspring of a population,
- $m_{\Phi_m}: I^\lambda \rightarrow I^\lambda$ is the mutation operator which modifies the offspring of a population,

- $s_{\Phi_s}: W \rightarrow I^{\mu}$ is the selection operator which selects the parent population of the next generation, where $W = I^{\lambda}$ or $W = I^{\mu+\lambda}$, and
- $\iota: I^{\mu} \rightarrow \{true, false\}$ is the termination criterion.

Subsequently the algorithmic description of ES is outlined by means of these notations (Bäck and Schwefel 1993).

```
Set $g = 0$
Initialize $A(0) = \{a_1(0), a_2(0), \dots, a_{\mu}(0)\} \in I^{\mu}$
Evaluate $A(0)$: $\{\Phi(a_1(0)), \Phi(a_2(0)), \dots, \Phi(a_{\mu}(0))\}$
while not $\iota(A(g))$ do
    Recombine: $A'(g) = r_{\Phi_r}(A(g))$
    Mutate: $A''(g) = m_{\Phi_m}(A'(g))$
    Evaluate $A''(g)$: $\{\Phi(a_1(g)), \Phi(a_2(g)), \dots, \Phi(a_{\mu}(g))\}$
    Select: $A(g+1) = s_{\Phi_s}(A''(g) \cup W)$, where $W \in \{\emptyset, A(g)\}$
    Set $g = g + 1$
end while
```

Hereafter the fitness evaluation, the representation of individuals, and the three main processes - recombination, mutation, and selection - are discussed in more detail.

Fitness Evaluation and Individual Representation

The fitness value of an individual is identical to the objective function value of the corresponding parameter set, i.e. $\Phi(a_k) = Q(x_k)$, where x_k is the parameter set corresponding to individual a_k. An individual a_k is comprised of a set of **objective parameters** x_k and, additionally, a set of **endogenous strategy parameters** s_k

$$a_k = (x_k, s_k) \in I \qquad \text{Eq. 3-11}$$

The endogenous strategy parameters are used to control the statistics of the mutation process.

Recombination

The recombination process produces one new individual from two randomly selected parent individuals or the global form which allows taking components for one new individual from potentially all individuals of the parent population. Recombination is performed for objective parameters as well as for the endogenous strategy parameters, and different mechanisms may be used for objective and strategy parameters. The mechanisms are only presented for the objective parameters (Bäck and Schwefel 1993), whereby the operator $r': I^{\mu} \rightarrow I$ is used for producing one new individual $r'(P(t)) = a' = (x', s') \in I$

$$x_i' = \begin{cases} x_{S,i} & no\ recombination \\ x_{S,i}\ or\ x_{T,i} & discrete \\ x_{S,i} + \chi \cdot (x_{T,i} - x_{S,i}) & generalized\ intermediate \\ x_{S_i,i}\ or\ x_{T_i,i} & global, discrete \\ x_{S_i,i} + \chi_i \cdot (x_{T_i,i} - x_{S_i,i}) & global, generalized\ intermediate \end{cases} \quad \text{Eq. 3-12}$$

$$\forall\ i = 1, 2, \ldots, n$$

The indices S and T denote two parent individuals randomly selected from the population $A(g)$ and $\chi \in [0,1]$. For global recombination, new parents S_i and T_i are selected for each component x_i as well as values for the variable χ_i. Fixing $\chi = 1/2$ and $\chi_i = 1/2, \forall\ i = 1, 2, \ldots, n$, respectively, reduces the (global) generalized intermediate recombination to **intermediate recombination**.

Empirical studies showed best results for discrete recombination on objective parameters and intermediate recombination on strategy parameters and, additionally, the necessity of recombination on strategy parameters for a well-performing ES (Bäck and Schwefel 1993).

Mutation

The mutation process modifies the offspring generated by one of the recombination mechanisms. Thereby, several methods can be applied which all use a specific number of endogenous strategy parameters. The simplest mutation operation functions with only one strategy parameter σ for the mutation strength. The objective parameters are then modified by

$$x' = x + z \quad \text{Eq. 3-13}$$

$$z = \sigma \cdot \mathcal{N}, \qquad \mathcal{N} = (\mathcal{N}_1(0,1), \mathcal{N}_2(0,1), \ldots, \mathcal{N}_n(0,1)) \quad \text{Eq. 3-14}$$

where $\mathcal{N}_i(0,1)$ is a normally distributed random number. Hence, the mutation operator favors small changes which depend on the choice of the mutation strength σ. The samples are **isotropically** distributed around the parental parameter set. This method has the advantage that only one endogenous strategy parameter is needed for mutation control and is sufficient for objective functions which have spherical surfaces. However, if the surface of an objective function is ellipsoidal, it is beneficial to have mutation vectors whose surfaces of constant density are also ellipsoidal (Beyer and Schwefel 2002). The simplest form of ellipsoidal mutation is the **axes-parallel ellipsoid mutation** which requires n endogenous strategy parameters. These are n standard deviations $s = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ each associated with one component of the objective parameter vector to represent an own mutation strength for each parameter

$$z = \left(\sigma_1 \mathcal{N}_1(0,1), \sigma_2 \mathcal{N}_2(0,1), \ldots, \sigma_n \, \mathcal{N}_n(0,1)\right). \qquad \text{Eq. 3-15}$$

The mutation process can be further modified by rotating the mutation ellipsoid arbitrarily in the search space, called **rotated ellipsoid mutation**

$$\begin{aligned} z &= M \cdot \left(\sigma_1 \mathcal{N}_1(0,1), \sigma_2 \mathcal{N}_2(0,1), \ldots, \sigma_n \, \mathcal{N}_n(0,1)\right)^t = MD\mathcal{N}(O,I) \\ &= \mathcal{N}(O, MD^2M^t) = D\mathcal{N}(0,C) \end{aligned} \qquad \text{Eq. 3-16}$$

where M is a rotation matrix which represents the correlations between the components of z, D is a diagonal matrix with the standard deviations σ_i on its diagonal, and C is the covariance matrix. The usage of this mutation mechanism requires $n(n+1)/2$ strategy parameters.

All introduced mutation mechanisms require an appropriate adaptation of the endogenous strategy parameters. This can be achieved by the so-called **self-adaptation** (Schwefel 1987). Therefore, the strategy parameters may undergo the recombination process and always the mutation process. The mutated strategy parameters are then used to control the mutation process of the objective parameters as previously mentioned.

One single strategy parameter is then mutated isotropically by

$$\sigma' = \sigma \cdot exp\left(\tau \cdot \mathcal{N}(0,1)\right), \qquad \text{Eq. 3-17}$$

where τ is the so-called **learning parameter** which determines the rate and precision of self-adaption. Theoretical and experimental results (Beyer 1995, Schwefel 1975) suggest to choose

$$\tau \propto \frac{1}{\sqrt{n}} \qquad \text{Eq. 3-18}$$

(the first guess might be $\tau = 1/\sqrt{n}$; in highly multimodal fitness landscapes smaller learning rates should be tried, e.g. $\tau = 1/\sqrt{2n}$ (Beyer and Schwefel 2002)).

This technique can be extended for axes-parallel ellipsoid mutation with one strategy parameter for each objective parameter (Schwefel 1977)

$$\sigma_i{}' = \sigma_i \cdot exp\left(\tau' \cdot \mathcal{N}(0,1) + \tau \cdot \mathcal{N}_i(0,1)\right), \quad i = 1, 2, \ldots, n \qquad \text{Eq. 3-19}$$

Thereby, a general mutation is combined with a mutation for each component. The following values are recommended for the learning parameters τ' and τ (Schwefel 1977)

$$\tau' \propto \frac{1}{\sqrt{2n}} \qquad \tau \propto \frac{1}{\sqrt{2\sqrt{n}}} \tag{Eq. 3-20}$$

Usually, the proportional constant is chosen to be 1 (Schwefel 1995).

If the rotated ellipsoidal mutation is applied, $n(n-1)/2$ additional elements of the matrix M have to be modified by

$$m_j' = m_j + \beta \cdot N_j(0,1), \;\; j = 1,2,\dots,n(n-1)/2, \tag{Eq. 3-21}$$

whereby $\beta \approx 0.0873$ (Schwefel 1977). The mutation of the standard deviation is performed in the same manner as mentioned in Eq. 3-19.

The recombination of the endogenous strategy parameters can be performed by the mechanisms in Eq. 3-12. Thereby, the intermediate recombination is highly recommended (Beyer and Schwefel 2002).

Selection

The selection process is a complete deterministic process with two possible strategies (Bäck and Schwefel 1993)

- (μ,λ)-selection (**comma-selection**): the μ best individuals out of the set of λ offspring individuals are selected.
- $(\mu+\lambda)$-selection (**plus-selection**): the μ best individuals out of the union of parents and offspring are selected.

The disadvantage of the plus-selection is the inability to react on changing environmental conditions. Additionally, it hinders the self-adaption mechanism efficiently working because mismatched strategy parameters may survive for many generations. The capability of the comma-selection to forget good solutions allows, in principle, the leaving of local minima and is, therefore, advantageous for multimodal objective functions (Bäck 1996). The (μ,λ)-selection method is recommended today and experiments indicate that the optimal ratio of the number of parents and offspring is $\mu/\lambda = 1/7$ (Schwefel 1987).

Based on the introduced recombination, mutation, and selection mechanisms as well as the self-adaption process of the endogenous strategy parameters, the mentioned algorithm can be more specified. This conceptual algorithm can be found in Algorithm A4 (Appendix A1).

Thereby, the algorithm could be terminated when the distance between the best parameter sets found in subsequent iterations fall below a determined boundary

$$\iota\big(A(t)\big) = \begin{cases} true & \|x_{min}(g) - x_{min}(g-1)\| \leq \varepsilon \\ false & otherwise \end{cases} \qquad \text{Eq. 3-22}$$

but this criterion guarantees no sufficient convergence because small steps occur not only if the current parameter set is near the minimum but also if the search is moving through a narrow valley (Schwefel 1995). It is better to compare the objective function values. Then the algorithm ends when the difference between the worst objective function value of the parent population and the best objective function value so far Q_{min} becomes arbitrarily small (Bäck 1996)

$$\iota\big(A(g)\big) = \begin{cases} true & max\left\{Q\big(x_1(g)\big), Q\big(x_2(g)\big), \ldots, Q\left(x_\mu(g)\right)\right\} - Q_{min} \leq \varepsilon \\ false & otherwise \end{cases} \qquad \text{Eq. 3-23}$$

In this manner the objective function values of the parents in a generation must fall closely together before the convergence is accepted.

Another aspect is global convergence and the convergence order, i.e. the time complexity of ES. To prove global convergence, it has to be shown that the probability of reaching a specific neighborhood of the global optimum $Q^* = Q(x^*)$ by a sequence of search points is one if the number of generations goes to infinity

$$\forall \varepsilon > 0\text{:}\ \lim_{g\to\infty} Pr\big(Q\big(x(g)\big) - Q^* \leq \varepsilon\big) = 1. \qquad \text{Eq. 3-24}$$

A sketched proof of Eq. 3-24 for $(1+1)$-ES with constant mutation strength can be found in (Rechenberg 1971) which should be easily accomplished for $(\mu+\lambda)$-ES according to (Beyer 2001). However, (μ,λ)-ES are generally not convergent. They require an appropriate control of the mutation strength to converge at least locally. The global convergence property of an ES is good to know but no one can wait indefinitely so the relevance is rather low. More important is the convergence order. Rechenberg calculates the convergence order exemplary for two basic functions by applying the $(1+1)$-ES (Rechenberg 1971). General considerations about global convergence and convergence order are rather difficult and, according to current knowledge, not yet published.

Hereafter, an extension of this basic ES procedure is introduced. The covariance matrix adaptation evolution strategy (CMAES) modifies the update of the covariance matrix C in case of the rotated ellipsoidal mutation in Eq. 3-16.

Covariance Matrix Adaptation Evolution Strategy

CMAES was introduced by Hansen and Ostermeier (Hansen and Ostermeier 2001) to update the covariance matrix of the multivariate normal distribution for the mutation process in Eq. 3-16. The covariance matrix describes the pairwise dependencies between the objective parameters, hence, the adaptation process is similar to learning about the second order information of the underlying objective function. The update method is, additionally, improved by cumulation, i.e. evolution paths are utilized instead of single search steps.

CMAES starts with the mutation process to generate offspring by sampling a multivariate normal distribution

$$\begin{gathered} x_k(g+1) = m(g) + z_k(g) \\ z_k(g) = \sigma(g) \cdot \mathcal{N}\big(0, C(g)\big), \qquad k = 1, 2, \dots, \lambda, \end{gathered} \qquad \text{Eq. 3-25}$$

whereby $\sigma(g)$ is the “overall” standard deviation (step size) at generation g.

Afterwards, the μ best individuals of the λ offspring individuals are selected for the recombination process, i.e. $x_{k:\lambda}(g+1)$ is the kth best parameter set such as

$$Q\big(x_{1:\lambda}(g+1)\big) \leq Q\big(x_{2:\lambda}(g+1)\big) \leq \cdots \leq Q\big(x_{k:\lambda}(g+1)\big) \leq \cdots \leq Q\big(x_{\lambda:\lambda}(g+1)\big).$$

The recombination is performed by the so-called **weighted intermediate recombination**, i.e. the weighted average of the μ best individuals

$$\begin{gathered} m(g+1) = \sum_{k=1}^{\mu} w_k x_{k:\lambda}(g+1) \\ \sum_{k=1}^{\mu} w_k = 1, \;\; w_1 \geq w_2 \geq \cdots \geq w_\mu > 0. \end{gathered} \qquad \text{Eq. 3-26}$$

The measure

$$\mu_{eff} = \left(\frac{\|w\|_1}{\|w\|_2}\right)^2 = \frac{\sum_{k=1}^{\mu} |w_k|}{\sum_{k=1}^{\mu} (w_k)^2} = \frac{1}{\sum_{k=1}^{\mu} (w_k)^2} \qquad \text{Eq. 3-27}$$

is called **variance effective selection mass** with $1 \leq \mu_{eff} \leq \mu$. Usually, $\mu_{eff} \approx \lambda/4$ indicates a reasonable setting of w_k, and typical values are $w_k \propto (\mu - k + 1)$ and $\mu \approx \lambda/2$ (Hansen 2006).

The next step is to adapt the covariance matrix by applying two methods, rank-one-update and rank-μ-update, which are combined to use the advantages of both. In this manner the information of the current population is exploited efficiently by the rank-μ-update and the rank-one-update considers the correlations between the generations

$$C(g+1) =$$

$$\left(1-c_1-c_\mu\right)C(g)+c_1 q_c(g+1)q_c(g+1)^t+c_\mu\sum_{k=1}^{\mu}w_k y_{k:\lambda}(g+1)y_{k:\lambda}(g+1)^t \quad \text{Eq. 3-28}$$

$$q_c(g+1)=(1-c_c)q_c(g)+\sqrt{c_c(2-c_c)\mu_{eff}}\cdot\left(\frac{m(g+1)-m(g)}{\sigma(g)}\right)$$

where $y_{k:\lambda}(g+1)=\frac{x_{k:\lambda}(g+1)-m(g)}{\sigma(g)}$, $c_\mu \leq 1$ and $c_1 \leq 1$ are the learning parameters, and q_c is the **evolution path**, also called **cumulation**, which is expressed by a sum of consecutive steps.

The last step is the control of the overall step size σ. This is also done by constructing an evolutionary path

$$q_\sigma(g+1)=(1-c_\sigma)q_\sigma(g)+\sqrt{c_\sigma(2-c_\sigma)\mu_{eff}}\cdot C(g)^{-\frac{1}{2}}\cdot\left(\frac{m(g+1)-m(g)}{\sigma(g)}\right), \quad \text{Eq. 3-29}$$

where c_σ is the learning parameter. The length of the evolution path provides information for the update of the step size. If the evolution path is short, single steps cancel out each other and the step size should be decreased. However, if the evolutionary path is long, single steps point to similar directions and the step size should be increased. To decide whether the evolutionary path is long or short, it is compared with its expected length under random selection $E(\|N(0,I)\|)$

$$\sigma(g+1)=\sigma(g)\,exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|q_\sigma(g+1)\|}{E(\|N(0,I)\|)}-1\right)\right) \quad \text{Eq. 3-30}$$

$$E(\|N(0,I)\|)\approx\sqrt{n}\left(1-\frac{1}{4n}+\frac{1}{21n^2}\right),$$

where $d_\sigma$is the damping parameter.

The mentioned mechanisms lead to the algorithm outlined in Algorithm A5 (Appendix A1) which is taken from (Hansen 2006). The default values for the exogenous strategy parameters of the algorithm are also given in Table A1 (Appendix A1). Hansen does not recommend changing the default values of the exogenous strategy parameters because they are particularly chosen to be a robust setting (Hansen 2006). The only exception is the population size λ which can be increased due to its significant influence on the global search performance (Hansen and Kern 2004). Increasing the population size usually causes an improvement of the global search capability and the robustness of CMAES but, additionally, the convergence speed is reduced.

Thereby, it is assumed that the algorithm converge to a minimum when the difference between the worst objective function value of the parent population and the best objective function value so far Q_{min} fall below a determined accuracy

$$a_1 = \begin{cases} true & max\left\{Q\big(x_1(g)\big), Q\big(x_2(g)\big), \dots, Q\left(x_\mu(g)\right)\right\} - Q_{min} \leq \varepsilon \\ false & otherwise \end{cases} \qquad \text{Eq. 3-31}$$

However, according to present knowledge no global convergence results are proven till now; but experimental studies have shown that a large class of function converges fast to the global optimum. Some functions converge with the probability of one, independent of the initial settings. But others have a probability less than one which normally depends on the initial values for m^0 and σ^0. Additionally, no general results of the convergence order have been published thus far. Furthermore, the step-size control in Eq. 3-30 prevents the algorithm from converging prematurely but it does not avoid allowing the search to end up in a local minimum. Large populations help avoid local minima but with slower convergence rate. For a detailed description of CMAES and performance considerations, refer to (Hansen 2006), (Hansen and Ostermeier 2001), and (Hansen and Kern 2004).

3.2.3 Hybrid Optimization Methods

The main drawback of ES approaches is the high computation cost due to the slow convergence rates. They work generally well to explore the parameter space but they are slow in accurately finding the minimum of the objective function. However, local methods are much faster in finding a minimum once in the right neighborhood but they can be entrapped in a local minimum. Hence, the optimization process can be enhanced by combining a global and a local method to a so-called **hybrid method**. The purpose of hybrid methods is to speed up the convergence rate while retaining the ability to avoid being easily entrapped at a local minimum. Thereby, the ES method localizes a **promising region** within the parameter space and the local optimizer reaches the best solution in this region accurately and quickly. Yen et al. divided hybrid methods into four groups (Yen et al. 1998):

1. **Pipeline hybrids**: ES and a local method are applied sequentially. They can be further classified to (see Figure 3.9):
    a. **Preprocessor**: ES is applied at first and afterwards the local method.
    b. **Primary**: The local method is applied at first and afterwards ES.

   c. **Staged pipeline**: ES and the local method are interleaved with each other.

1. **Asynchronous hybrids**: ES and the local method proceed and cooperate asynchronously using a shared population.
2. **Hierarchical hybrids**: ES and the local method operate on different level on the optimization problem.
3. **Additional operators**: An additional reproduction operator is introduced that perform the local search. There are two architectures to integrate this new operator:
   a. **Partition-based hybrid architecture**: the current population is divided into several disjoint subgroups. Each subgroup can reproduce their offspring by the conventional mechanisms mentioned previously or by the new operator. Thereby, the new operator is selected with a specific probability.
   b. **Elite-based hybrid architecture**: the new operator is applied to the top-ranking individuals to generate a portion of offspring.

This study focuses on pipeline hybrids especially those with ES and CMAES, respectively, as preprocessor. Thereby, the general hybrid algorithm is performed in two phases: **diversification** and **intensification** (Chelouah and Siarry 2003). In the diversification phase the mechanism of ES are applied repeatedly: recombination, mutation, and selection. It stops when one of the following abort criteria is fulfilled:

S1 A given number of generations (g_{max}) is reached.

S2 A given accuracy which corresponds to the objective function values is reached, i.e. the difference between the worst objective function value of the current population and the best objective function value so far Q_{min} is smaller than a given accuracy ε

$$\max\left\{\mathrm{Q}\big(x_1(g)\big), \mathrm{Q}\big(x_2(g)\big), \dots, \mathrm{Q}\left(x_\mu(g)\right)\right\} - \mathrm{Q}_{min} \leq \varepsilon$$

S3 A given accuracy which corresponds to the localization of the objective parameters in the current population is reached, i.e. the average distance between the best parameter set up till now x_{min} and the remaining parameter sets of the population is smaller than a given neighborhood radius ρ

$$\frac{1}{\mu}\sum_{k=1}^{\mu} \|x_k - x_{min}\| \leq \rho.$$

Then it is assumed that the individuals are all in the same area, called **promising area**. If the diversification phase stops with individuals all in the promising area, the parameter set of the best individual is the initial value x^0 of the local method. In this study the Nelder-Mead simplex method and the Hooke-Jeeves method serve as local optimizer (see Section 3.2.1). The hybrid approach is outlined in Algorithm A6 (Appendix A1).

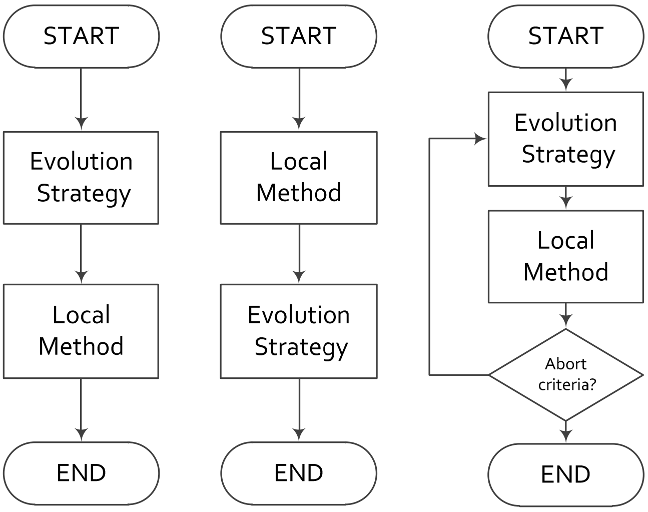


Figure 3.9: The classes of pipeline hybrids; left: preprocessor, middle: primary, and right: staged pipelining (Yen et al. 1998)

3.3 Sensitivity Analysis Methods

The sensitivity analysis evaluates the contribution of each model parameter to the variation of the model output, i.e. it reveals the effects of changes in the model parameters. Saltelli et al. give the following definition (Saltelli et al. 2004): *The study of how uncertainty in the output of a model (numerical or otherwise) can be apportioned to different sources of uncertainty in the model input***.**

Within this study, the inputs are a subset of the model parameters x, either the structure parameters $x = p$ or the process parameters $x = z$ (see Section 3.2), and the vector $y(t, x)$ comprises the considered model outputs. The model output and, thus, the sensitivity information have also been summarized by the objective function value $Q(x)$ in Eq. 3-6.

The sensitivities of model parameters which correspond to the model output can either be evaluated by local or global methods. **Local methods** carry out the local impact of model parameters on the model output at some fixed point in the parameter space.

However, **global methods** offer possibilities to obtain parameter sensitivities which are valid for the entire considered parameter range. Additionally, no assumptions according to continuity and differentiability of the model output have to be made. Global methods are usually performed by the following steps (see Figure 3.10):

1. Specify the model outputs of interest and possibly an objective function to summarize the sensitivities.
2. Assign a probability density function to each model parameter of interest which covers its uncertainty range in Eq. 3-6. This range is determined by experimental measurements, theoretical assumptions, e.g. physical laws, scientific literature, or expert judgment.
3. Generate a sample according to the defined probability density functions in step 2.
4. Simulate the model for all parameter sets of the sample generated in step 3 (and calculate the corresponding value of the objective function in Eq. 3-6.).
5. Explore the impact of each model parameter on the objective function by an appropriate method.

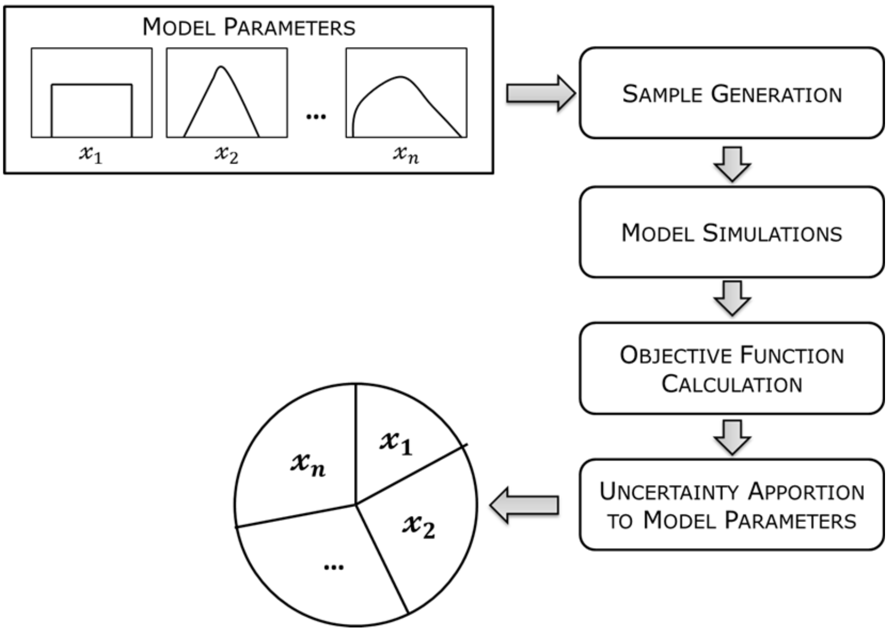


Figure 3.10: General scheme of a global SA (Saltelli et al. 1999)

Several global SA methods are available. An overview of most methods can be found in (Saltelli et al. 2000). Generally, they can be classified into two groups:

- **Sample-based methods**: based on a generated sample of the model parameter. The sensitivities are determined by specific coefficients. Possible coefficients are the standardized regression coefficient, (partial) correlation coefficient, or their rank-transformed variants (Saltelli et al. 2000).
- **Variance-based methods**: aim at decomposing the variance of the model output as the sum of contributions of each model parameter and parameter combinations. Sometimes, they are also called ANOVA (**ANalysis Of VAriance**) methods. The FAST method (**Fourier Amplitude Sensitivity Test**) (Cukier et al. 1973) and **Sobol's method** (Sobol 1993) are possible techniques.

According to Saltelli et al., an ideal global SA method has to fulfill four properties (Saltelli et al. 2004, Saltelli et al. 2000):

1. **The inclusion of influence of scale and shape**: The sensitivity estimates of individual model parameters should incorporate the effect of the range and shape of the assigned probability density functions, i.e. it matters if the underlying probability density function of a model parameter corresponds to a uniform or to a normal distribution and how the corresponding distribution parameters are.
2. **Multidimensional averaging**: The sensitivity estimates of individual model parameters are evaluated by varying all other model parameters as well. In contrast, local methods are based on the OAT (one-at-a-time) approach, i.e. only one parameter is varying at a time while the others are kept constant at their actual values.
3. **Model independence**: The method should work regardless of the characteristics of the underlying objective function and the model, respectively. These characteristics are
   a. **Linearity**: The model output $\mathcal{Y}$ depends linearly on the input parameters x
   $$\mathcal{Y} = c_0 + \sum_{i=1}^{n} c_i x_i.$$
   b. **Additivity**: The model output $\mathcal{Y}$ is additive if
   $$\mathcal{Y} = f(x_1, x_2, \dots, x_n) = \sum_{i=1}^{n} f_i(x_i).$$
   c. **Monotony**: The model output $\mathcal{Y}$ depends monotonically on the input parameter x_i
   $$\mathcal{Y}_k \geq \mathcal{Y}_{k+1} \quad if\ x_i^k \geq x_i^{k+1}\ or\ \mathcal{Y}_k \leq \mathcal{Y}_{k+1} \quad if\ x_i^k \leq x_i^{k+1}.$$
4. **Parameter groups**: The method should treat grouped model parameters as if they were single model parameters. This is important for interpreting of the results.

The sample-based methods fulfill only property 1 and 2 but they are not model independent; they are only applicable when model input and output behave monotonically and, in addition, the parameters cannot be summarized into groups. However, variance-based methods satisfy all four properties and are, thus, ideal global SA techniques.

Within this study the relationships between the objective function in Eq. 3-6 and the (structure or process) parameters of an xHPN model should be analyzed. These relationships are usually non-monoton and, hence, only variance-based methods are considered hereafter.

Variance-based methods decompose the variance of the objective function to the model parameters. Therefore, the objective function is rewritten as a HDMR (**High Dimensional Model Representation**) (Sobol 1993)

$$Q(x) = q_0 + \sum_{i=1}^{n} q_i(x_i) + \sum_{i=1}^{n} \sum_{j>i} q_{ij}(x_i, x_j) + \cdots + q_{12\ldots n}(x_1, x_2, \ldots, x_n)$$

$$q_i(x_i) = E(Q|x_i) - q_0, \qquad q_0 = E(Q) \tag{Eq. 3-32}$$

$$q_{ij}(x_i, x_j) = E(Q|x_i, x_j) - q_i - q_j - q_0.$$

This is a decomposition of the objective function into terms of increasing dimensionality, whereby $E(Q|x_i) = E(Q|x_i = x_i^*)$ is the conditional expectation value, i.e. the expectation value of $Q(x)$ if the parameter x_i is fixed at x_i^*.

If the model parameters are independent of each other, it is possible to decompose the variance of $Q(x)$ also into terms of increasing dimensionality

$$Var(Q) = \sum_{i=1}^{n} V_i + \sum_{i=1}^{n} \sum_{j>i} V_{ij} + \cdots + V_{12\ldots n}$$

$$V_i = Var\big(q_i(x_i)\big) = Var(E(Q|x_i) - q_0) = Var\big(E(Q|x_i)\big) \tag{Eq. 3-33}$$

$$V_{ij} = Var\left(q_{ij}(x_i, x_j)\right) = Var\big(E(Q|x_i, x_j) - q_i - q_j - q_0\big)$$

$$= Var\left(E(Q|x_i, x_j)\right) - V_i - V_j,$$

where $V_i, V_{ij}, \ldots, V_{12\ldots n}$ are called **partial variances** and $V_i = Var\big(E(Q|x_i)\big)$ is variance taken over all possible values of x_i^*. It expresses the expected amount of variance that would be removed from the total variance $Var(Q)$ if the true value of the model parameter x_i would be known, called **main effect** (Saltelli et al. 2008), hence, a large value of V_i implies that x_i is an important parameter.

Dividing both sides of Eq. 3-33 by the variance of the objective function leads to

$$1 = \sum_{i=1}^{n} S_i + \sum_{i=1}^{n} \sum_{j>i} S_{ij} + \cdots + S_{12\ldots n} \tag{Eq. 3-34}$$

$$S_i = \frac{Var\big(E(Q|x_i)\big)}{Var(Q)}$$

$$S_{ij} = \frac{Var\left(E(Q|x_i, x_j)\right) - V_i - V_j}{Var(Q)},$$

where $S_i, S_{ij}, \ldots, S_{12\ldots n}$ are called **first-order sensitivity indices**, **second-order sensitivity indices**, …,and $\boldsymbol{nth}$**-order sensitivity indices**, respectively. The first order sensitivity index represents the main effect contribution of each model parameter to the variance of the objective function; thus, it is a measure of the relative importance of an individual parameter x_i and can be used to rank the parameters (Saltelli et al. 2008, Sobol 1993).

The **total effect** of a parameter x_i comprises its first-order effect and, additionally, all higher-order effects due to interactions. This is associated with the fact that the variance is composed of

$$Var(Q) = E\big(Var(Q|x_{\sim i})\big) + Var\big(E(Q|x_{\sim i})\big) \qquad \text{Eq. 3-35}$$

according to the law of total variance and $x_{\sim i}$ is comprised of all parameters expect x_i, hence, $E\big(Var(Q|x_{\sim i})\big) = Var(Q) - Var\big(E(Q|x_{\sim i})\big)$ is the expected amount of variance that would remain unexplained if only x_i were left to vary within its uncertainty range and all other parameters are known. The **total sensitivity index** is then given by

$$S_{Ti} = \frac{E\big(Var(Q|x_{\sim i})\big)}{Var(Q)} = \frac{Var(Q) - Var\big(E(Q|x_{\sim i})\big)}{Var(Q)} = 1 - \frac{Var\big(E(Q|x_{\sim i})\big)}{Var(Q)}. \qquad \text{Eq. 3-36}$$

First-order sensitivity coefficients can be estimated by using methods such as FAST (Fourier Sensitivity Amplitude Test) (Cukier et al. 1973) or the method of Sobol (Sobol 1993). To carry out the total sensitivity indices Sobol's method can also be used or an extended version of the FAST method called eFAST (**extended FAST**) introduced by Saltelli et al. (Saltelli et al. 1999). This study concerns the FAST and eFAST method.

Hereafter the local approach for calculating sensitivities of the model parameters with respect to the model outputs $\mathcal{y}(x,t)$ and objective function value $Q(x)$ is introduced first. Afterwards the FAST and eFAST method are described in detail.

3.3.1 Local approach

A local SA is usually performed by computing partial derivatives of the model outputs with respect to the model parameters and, thus, it is only applicable when the model outputs are differentiable in a neighborhood of the actual value x^*. The partial derivatives of the model output $y_j(t, x^*),\ \ j = 1, \dots, n_y$ with respect to the model parameters x_i^*

$$\frac{\partial y_j(x^*, t)}{\partial x_i} \qquad \text{Eq. 3-37}$$

$$i = 1, \dots, n, j = 1, \dots, n_y$$

are called **first-order sensitivity coefficients** and form the **Jacobian**, also called **sensitivity matrix**, in this context

$$S(x^*, t) = \begin{pmatrix} \dfrac{\partial y_1(x^*, t)}{\partial x_1} & \cdots & \dfrac{\partial y_1(x^*, t)}{\partial x_n} \\ \dfrac{\partial y_2(x^*, t)}{\partial x_1} & \cdots & \dfrac{\partial y_2(x^*, t)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial y_{n_y}(x^*, t)}{\partial x_1} & \cdots & \dfrac{\partial y_{n_y}(x^*, t)}{\partial x_n} \end{pmatrix}. \qquad \text{Eq. 3-38}$$

If the analytical solution of the model output $y(x^*, t)$ is known, the sensitivity matrix S can be obtained by differentiation. This is usually not the case and, therefore, numerical methods have to be applied. The simplest method to calculate local sensitivities numerically is the **finite-difference approximation**. Thereby, one parameter is changed slightly at a time while the others are held fixed and the model is rerun

$$\frac{\partial y_j(x^*, t)}{\partial x_i} \approx \frac{y_j(x^* + he_i, t) - y_j(x^*, t)}{h} \qquad \text{Eq. 3-39}$$

$$i = 1, \dots, n, j = 1, \dots, n_y,$$

where e_i is the ith unit vector. This approximation is called **forward difference**. It is also possible to approximate the partial derivatives by **backward** or **central differences**, respectively:

$$\frac{\partial y_j(x^*, t)}{\partial x_i} \approx \frac{y_j(x^*, t) - y_j(x^* - he_i, t)}{h} \qquad \text{Eq. 3-40}$$

$$i = 1, \dots, n, \qquad j = 1, \dots, n_y$$

$$\frac{\partial y_j(x^*, t)}{\partial x_i} \approx \frac{y_j(x^* + he_i, t) - y_j(x^* - he_i, t)}{2h} \qquad \text{Eq. 3-41}$$

$$i = 1, \dots, n, \qquad j = 1, \dots, n_y$$

The calculation of the sensitivity matrix requires $n + 1$ simulations of the model if forward or backward differences are applied and $2n$ simulations in the case of central differences. The advantage of this method is that no extra code beyond the original ODE-solver is needed for the calculation of sensitivities; but on the other hand, it is difficult to find the right level of the parameter change h because the accuracies of the sensitivities depend on it. If the change is too small, the difference between original and perturbed solution is too small and the round-off error is too high and if it is too large, the linear approximation fails. To consider different magnitudes of the parameters, it is advisable to choose the parameter change as a percentage from the actual value

$$h_i = r \cdot x_i^* \qquad \text{Eq. 3-42}$$

where $0 < r < 1$.

To compare the local sensitivities, the entries of the sensitivity matrix have to be normalized. One possible way to do this, is to evaluate the partial derivatives of the logarithm of the model outputs and multiply them with the actual values

$$\tilde{S}(x^*, t) = \left\{ {x^*}_i \cdot \frac{\partial\, ln\left(\left|y_j(x^*, t)\right| + 1\right)}{\partial x_i} \right\} \qquad \text{Eq. 3-43}$$

$$i = 1, \dots, n, \qquad j = 1, \dots, n_y$$

Thereby, the model output is shifted by one to avoid negative values of the logarithm if the model output is less than one.

However, a practical difficulty of the sensitivity matrix is often its size. If a model may consist of 25 variables and 20 parameters, the matrix has 500 elements and if, in addition, 100 time points are studied, then 5000 sensitivities have to be compared. Thus, it is necessary to summarize the sensitivity information. This can be achieved by using an objective function (Eq. 3-6) which converts the multivariate output of a model to a single value. The partial derivatives of the objective function, with respect to the model parameters, are then given by

$$\nabla Q(x^*) = \left\{ \frac{\partial Q(x^*)}{\partial {x^*}_i} \right\} \qquad \text{Eq. 3-44}$$

$$i = 1, \dots, n$$

which is also known as the gradient of $Q(x^*)$.

First-order local sensitivity coefficients of the objective function only give information about the change of single parameters; but it is also important to analyze the effect of changing several parameters, simultaneously. The **principal component analysis** is based on local sensitivities and can be used to estimate the effect of simultaneous parameter changes (Vajda et al. 1985, Bard 1974).

Assuming that $Q(x)$ is twice continuous differentiable in an open neighborhood $\mathcal{N}$ of x^*, then $Q(x)$ can be expanded to its Taylor series at the actual point x^*

$$Q(x) \approx Q(x^*) + \nabla Q(x^*)\Delta x + \frac{1}{2}\Delta x^t \nabla^2 Q(x^*)\Delta x \qquad \text{Eq. 3-45}$$

where $\Delta x = x - x^*$, $\nabla Q(x^*)$ is the gradient of $Q(x)$ at x^*, and $\nabla^2 Q(x^*)$ is the Hessian of $Q(x)$ at x^*. If x^* is a minimum of $Q(x)$, then is $\nabla Q(x^*) = 0$ (see Theorem 3.1) and the objective function can be approximated by

$$\tilde{Q}(x) = Q(x^*) + \frac{1}{2}\Delta x^t \nabla^2 Q(x^*)\Delta x. \qquad \text{Eq. 3-46}$$

Hereafter it is simplified assuming that $Q(x^*) = 0$ if x^* is a minimizer of Q. The expression in Eq. 3-46 is then a quadratic approximation of the real shape of the objective function so that at any fixed $\varepsilon > 0$ the inequality

$$0 \leq \tilde{Q}(x) = \frac{1}{2}\Delta x^t \nabla^2 Q(x^*)\Delta x \leq \varepsilon \qquad \text{Eq. 3-47}$$

defines an ellipsoid in the parameter space with the principal axes defined by the Hessian (see Figure 3.11). The orientation of the ellipsoid with respect to the parameter axes is defined by the eigenvectors of the Hessian $H = \nabla^2 Q(x^*)$ while the relative lengths of the axes are revealed by the eigenvalues of this matrix. The function $\tilde{Q}(x)$ is most sensitive to a change in x along the principal axis corresponding to the largest eigenvalue and is least sensitive to a change in x along the principal axis corresponding to the smallest eigenvalue. If all principal axes of the ellipsoid are parallel to the axes of the parameter space, there is no synergistic effect among the parameters.

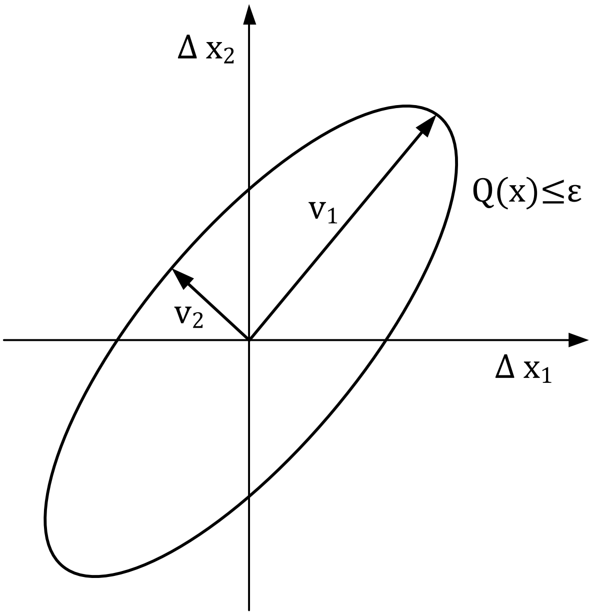


Figure 3.11: An approximated region defined by $\tilde{Q}(x) \leq \varepsilon$

This interpretation can be concretized by using the term **principal component**. A principal component is a new parameter obtained via linear combination of the original parameters

$$\psi = U^t x, \qquad \text{Eq. 3-48}$$

where U is the matrix of normalized eigenvectors obtained by diagonalization (eigenvalue-eigenvector decomposition) of the Hessian H:

$$H = U\Lambda U^t \qquad \text{Eq. 3-49}$$

with the diagonal matrix Λ formed by the eigenvalues of H. The objective function $Q(x)$ can be rewritten in dependency of the new parameters ψ:

$$\tilde{Q}(x) = \Delta x^t H \Delta x = \Delta x^t U\Lambda U^t \Delta x = \Delta\psi^t \Lambda \Delta\psi = \sum_{i=1}^{n} \lambda_i \Delta\psi_i^2 = \tilde{Q}(\psi). \qquad \text{Eq. 3-50}$$

This equation provides another explanation of why the eigenvectors of the matrix H reveal the related parameters and why the corresponding eigenvalues show the relative weights of these parameter groups. It clearly displays the inverse relationship between the lengths of the axes and the square roots of the eigenvalues. The largest eigenvalue indicates the parameter group with the largest sensitivity to a change in ψ corresponding to $\tilde{Q}(\psi)$ and the smallest eigenvalue indicates the parameter group with the smallest sensitivity to a change in ψ corresponding to $\tilde{Q}(\psi)$.

Example 3.16 (Bard 1974)

Consider, the two-dimensional case with

$$H = \begin{pmatrix} 0.505 & -0.495 \\ -0.495 & 0.505 \end{pmatrix}$$

and $\varepsilon = 1$. Thereby, Eq. 3-47 reduces to

$$\tilde{Q}(x) = 0.505\Delta x_1^2 - 0.99\Delta x_1 \Delta x_2 + 0.505\Delta x_2^2 \leq 1. \qquad \text{Eq. 3-51}$$

The normalized eigenvectors of the matrix H are $v_1 = (0.7071 \quad -0.7071)^t$ and $v_2 = (0.7071 \quad 0.7071)^t$ with the corresponding eigenvalues $\lambda_1 = 1$ and $\lambda_2 = 0.01$. From this follows

$$\Delta\psi = U^t \Delta x = \begin{pmatrix} 0.7071 & -0.7071 \\ 0.7071 & 0.7071 \end{pmatrix} \cdot \begin{pmatrix} \Delta x_1 \\ \Delta x_2 \end{pmatrix} = \begin{pmatrix} 0.7071(\Delta x_1 - \Delta x_2) \\ 0.7071(\Delta x_1 + \Delta x_2) \end{pmatrix}$$

$$\tilde{Q}(\psi) = \sum_{i=1}^{2} \lambda_i \Delta\psi_i^2 = \Delta\psi_1^2 + 0.01\Delta\psi_2^2$$

Hence, the principal axes have the lengths $\sqrt{\varepsilon/\lambda_1} = 1$ and $\sqrt{\varepsilon/\lambda_2} = 10$. It is clear from Figure 3.12 that if the parameters are changed simultaneously, large changes can be made before the boundary of $\tilde{Q}(x) = 1$ is exceeded. In fact, $\Delta x_1 = \Delta x_2 = 7.701$ satisfies Eq. 3-51.

On the other hand, if the changes in Δx_1 and Δx_2 are taken in opposite directions, the boundary is lower $\Delta x_1 = -\Delta x_2 = 0.701$. Thus, the principal component $\Delta\psi_2$ is very sensitive to changes and $\Delta\psi_1$ is less sensitive.

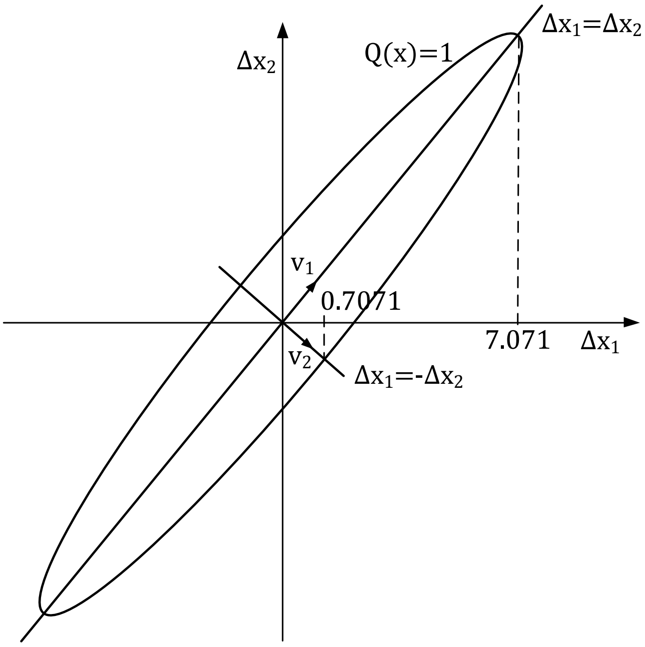


Figure 3.12: The approximated region for the matrix *H* with $\widetilde{Q}(x) = 1$

3.3.2 Global approach: Fourier Amplitude Sensitivity Test

The FAST method was developed by Cukier et al. in the 1970s (Cukier et al. 1973) and provides a way to estimate the first-order sensitivity indices in Eq. 3-34. It follows the general scheme of a global method depicted in Figure 3.10. At first, each considered model parameter x_i is provided with a probability density function $p_i(x_i)$. If the model parameters are not correlated, the probability density function $\mathcal{P}(x)$ is given by

$$\mathcal{P}(x) = \prod_{i=1}^{n} p_i(x_i). \qquad \text{Eq. 3-52}$$

The investigated model output is the value of the objective function in Eq. 3-6. The expected objective function value is then defined as

$$\begin{aligned} E(Q) &= \int \dots \int Q(x_1, x_2, \dots, x_n) \cdot \mathcal{P}(x_1, x_2, \dots, x_n)\, dx_1 \dots dx_n \\ &= \int \dots \int Q(x) \cdot \mathcal{P}(x)\, dx. \end{aligned} \qquad \text{Eq. 3-53}$$

The main idea of the FAST method is to convert this n-dimensional integral into an equivalent one-dimensional integral. Therefore, the parameters are transformed by a set of known transformation functions G, so-called **search curves**

$$x_i = G_i(sin(\omega_i s)), \ \ i = 1, 2, \dots, n, \qquad \text{Eq. 3-54}$$

whereby $\{\omega_i\}$ is a set of frequencies and s is a scalar variable, called **search variable**, which varies over the range $-\infty < s < +\infty$.

According to the ergodic theorem, Weyl shows that the expression for the expected objective function value in Eq. 3-53 is identical to the following one-dimensional integral (Weyl 1938)

$$E(Q) = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} Q\big(x_1(s), x_2(s), \dots, x_n(s)\big)\, ds. \qquad \text{Eq. 3-55}$$

One demand on the search curve is that it is space-filling, i.e. it passes arbitrarily close to any point in the n-dimensional parameter space. To ensure this, the frequencies ω_i have to be selected properly so that they are incommensurate, i.e.

$$\sum_{i=1}^{n} \alpha_i\, \omega_i \neq 0, \qquad \alpha_i - integers. \qquad \text{Eq. 3-56}$$

However, the space-filling curve is only an ideal which cannot be realized numerically due to the fact that the frequencies cannot be truly incommensurate; therefore, at most one of the incommensurate frequencies can be rational and all others have to be irrational. The numerical value of these irrational numbers cannot be stored exactly in a computer and, hence, an approximation of the incommensurate frequencies by a rational number is required. To calculate these frequencies, Schaibly and Shuler propose an appropriate set of integer frequencies which implies that the parameters $x_i, i = 1, 2, \dots, n$ are periodic in s on the finite interval $(-\pi, \pi)$ (Schaibly and Shuler 1973). Hence, Eq. 3-55 can be modified to

$$E(Q) = \frac{1}{2\pi} \int_{-\pi}^{\pi} Q\big(x_1(s), x_2(s), \dots, x_n(s)\big) ds. \qquad \text{Eq. 3-57}$$

and the variance of the objective function is then given by

$$\begin{aligned} Var(Q) &= E(Q^2) - E(Q)^2 \\ &= \frac{1}{2\pi} \int_{-\pi}^{\pi} Q^2\big(x_1(s), x_2(s), \dots, x_n(s)\big)\, ds - E(Q)^2. \end{aligned} \qquad \text{Eq. 3-58}$$

Hereafter $Q\big(x_1(s), x_2(s), \dots, x_n(s)\big)$ is denoted by $Q(s)$.

Frequency Selection

The integer frequencies are selected according to Schaibly and Shuler (Schaibly and Shuler 1973) such that they are **approximately incommensurate to order** M, i.e.

$$\sum_{i=1}^{n} \alpha_i \, \omega_i \neq 0, \;\; \alpha_i - integers$$
$$for \;\; \sum_{i=1}^{n} |\alpha_i| \leq M + 1, \qquad \text{Eq. 3-59}$$

whereby the order M can be determined by the investigator. The larger the chosen value of M, the better the coverage of the space. If $M \rightarrow \infty$, then the frequencies are completely incommensurate.

The frequency set $\{\omega_i\}$ can be calculated by a trial and error procedure (Cukier et al. 1973) (Cukier, et al., 1973). Schaibly et al. present a set of frequencies which are free of interferences to the fourth order for systems with 5 to 19 considered parameters (Schaibly and Shuler 1973). The listed frequencies have the smallest $\omega_{\max}$ that still satisfies the conditions from Eq. 3-59 and are so-called **minimal sets**.

Fourier Expansion

The objective function is expanded to Fourier series

$$Q(s) = \sum_{k=-\infty}^{\infty} A_k \, cos(ks) + B_k \, sin(ks). \qquad \text{Eq. 3-60}$$

The Fourier coefficients A_k and B_k are given by

$$A_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} Q(s) \cdot cos(ks) \, ds$$
$$B_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} Q(s) \cdot sin(ks) \, ds, \qquad \text{Eq. 3-61}$$

whereby $A_{-k} = A_k$ and $B_{-k} = -B_k$.

From Parseval’s theorem, it can be obtained under certain conditions that

$$E(Q^2) = \frac{1}{2\pi} \int_{-\pi}^{\pi} Q^2(s) \, ds = \sum_{k=-\infty}^{\infty} A_k^2 + B_k^2. \qquad \text{Eq. 3-62}$$

The squared mean value of Q can be simplified by using Eq. 3-60

$$E(Q)^2 = \left(\frac{1}{2\pi}\int_{-\pi}^{\pi} Q(s)ds\right)^2$$
$$= \frac{1}{4\pi^2}\left(\int_{-\pi}^{\pi} \sum_{k=-\infty}^{\infty} A_k\, cos(ks) + B_k\, sin(ks)\ ds\right)^2 = A_0^2. \qquad \text{Eq. 3-63}$$

Inserting Eq. 3-62 and Eq. 3-63 into Eq. 3-58 leads to an expression of the variance in terms of the Fourier coefficients

$$Var(Q) = E(Q^2) - E(Q)^2 = \sum_{k=-\infty}^{\infty} A_k^2 + B_k^2 - A_0^2 = 2\sum_{k=1}^{\infty} A_k^2 + B_k^2. \qquad \text{Eq. 3-64}$$

The part of the variance which corresponds to the uncertainty in the ith parameter, the partial variance from Eq. 3-33, can be determined by evaluating the effect of the frequency ω_i on the total variance. Therefore, the Fourier coefficients for the frequency ω_i and its higher harmonics are summed up

$$V_i = 2\sum_{q=1}^{\infty} A_{q\omega_i}^2 + B_{q\omega_i}^2. \qquad \text{Eq. 3-65}$$

The Fourier amplitudes decreases as q increases so that V_i can be approximated by

$$V_i \approx 2\sum_{q=1}^{M} A_{q\omega_i}^2 + B_{q\omega_i}^2, \qquad \text{Eq. 3-66}$$

whereby M is the maximum harmonic that is considered and equals the order chosen for the integer frequencies in Eq. 3-59.

The first-order sensitivity indices of Eq. 3-34 are then given by the ratio

$$S_i = \frac{V_i}{Var(Q)} \approx \frac{\sum_{q=1}^{M} A_{q\omega_i}^2 + B_{q\omega_i}^2}{\sum_{k=1}^{\infty} A_k^2 + B_k^2}. \qquad \text{Eq. 3-67}$$

The FAST method requires one model evaluation for each parameter combination which is the main component of the computational cost. Thus, it is desired to minimize the required number of model evaluations. This can be achieved by utilizing the symmetry properties of the search curve. If the frequency set $\{\omega_i\}$ only comprises odd integers, the search curves $G_i\big(\sin(\omega_i s)\big),\ i = 1, 2, \dots, n$ in Eq. 3-54 become symmetric about $\pm\frac{\pi}{2}$ so that the following symmetry properties hold:

$$Q(\pi - s) = Q(s)$$
$$Q(-\pi + s) = Q(-s) \qquad \text{Eq. 3-68}$$

$$Q\left(\frac{\pi}{2}+s\right)=Q\left(\frac{\pi}{2}-s\right)$$

$$Q\left(-\frac{\pi}{2}+s\right)=Q\left(-\frac{\pi}{2}-s\right).$$

The range of the search variable s can then be restricted to $-\frac{\pi}{2}\leq s\leq\frac{\pi}{2}$ and the Fourier coefficients in Eq. 3-61 are given by

$$A_k=\begin{cases}0 & if\ k\ odd\\ \frac{1}{\pi}\int_0^{\frac{\pi}{2}}\big(Q(s)+Q(-s)\big)\,cos(ks)\,ds & if\ k\ even\end{cases}$$

$$B_k=\begin{cases}0 & if\ k\ even\\ \frac{1}{\pi}\int_0^{\frac{\pi}{2}}\big(Q(s)-Q(-s)\big)\,sin(ks)\,ds & if\ k\ odd\end{cases}$$

Eq. 3-69

Applying these symmetry properties reduces the required model evaluations by one half.

FAST Sampling

The search curve given in equation Eq. 3-54 cannot be utilized in real problems because the number of points on the search curve is infinite. Hence, a finite subset of these points has to be selected. From the Nyquist criteria follows that the minimum number of points N has to be taken corresponding to the maximum frequency ω_{max} of the frequency set $\{\omega_i\}$ (Cukier et al. 1975)

$$N=2M\omega_{max}+1.$$

Eq. 3-70

In this manner, harmonics up to an order of M can be considered by the analysis. The larger the chosen value of the order M, the greater the likelihood that the Fourier amplitude of each input frequency reflects solely the uncertainty of the corresponding parameter. On the other hand, the larger the chosen M, the larger the maximum value ω_{max} of the frequency set which still satisfies Eq. 3-59 and the larger the number of sample points N required for the evaluation of the Fourier amplitudes. The order M is mostly chosen to be 4 or higher. A symmetric and uniformly spaced sample of the search variable s in the interval $\left[-\frac{\pi}{2},\frac{\pi}{2}\right]$, including $s=0$, is achieved by (Cukier et al. 1978)

$$s_j=\frac{\pi}{2}\left(\frac{2j-N-1}{N}\right),\quad j=1,2,\dots,N$$

Eq. 3-71

Search Curve

The search curve in Eq. 3-54 should provide a uniformly distributed sample of the model parameter x_i. Therefore, several transformation functions have been proposed (see Cukier et al. 1973, Koda et al. 1979, Saltelli et al. 1999). Within this work, the search curve suggested by Saltelli et al. is used (Saltelli et al. 1999)

$$x_i = \frac{1}{2} + \frac{1}{\pi} arcsin(sin(\omega_i s)), \qquad i = 1,2,\dots,n \qquad \text{Eq. 3-72}$$

This is a set of straight lines which oscillate between 0 and 1 and the empirical distribution can be regarded as more or less uniform.

Saltelli et al. proposed, additionally, a modification of Eq. 3-72 to obtain more flexible sampling schemes. The disadvantage of all these transformations is that they always return the same points. This can be avoided by a **random phase-shift** φ_i chosen uniformly from the interval $[0, 2\pi[$

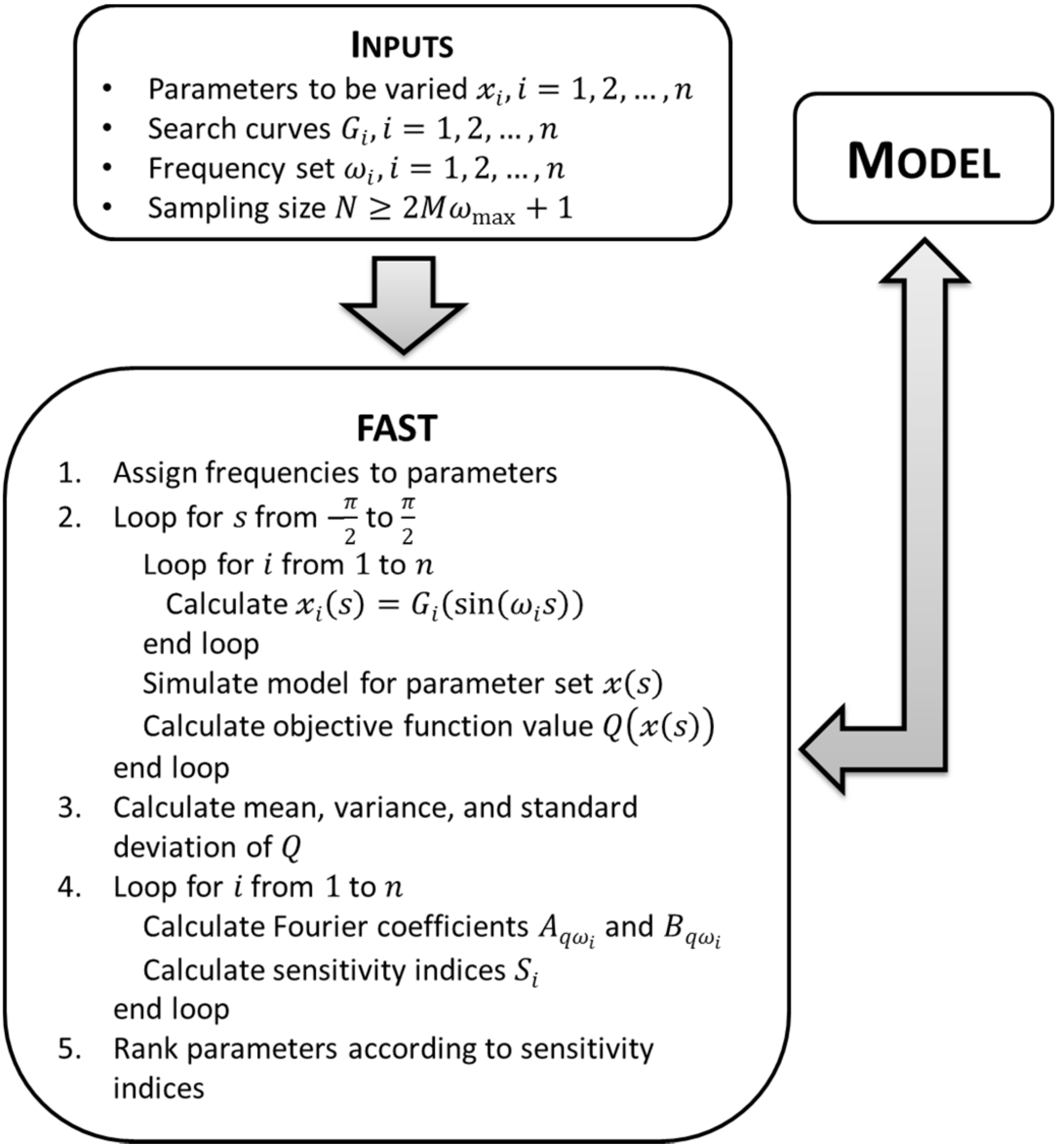
$$x_i = \frac{1}{2} + \frac{1}{\pi} arcsin(sin(\omega_i s + \varphi_i)), \qquad i = 1,2,\dots,n \qquad \text{Eq. 3-73}$$

However, the random shift causes that the symmetry properties of $Q(s)$ in Eq. 3-68 no longer hold. Hence, the search curve has to be sampled over the interval $(-\pi, \pi)$. By selecting various sets $\{\varphi_i\}$, different search curves can be realized. This procedure is called **resampling** and N_r denotes the number of curves used per parameter. The minimum sample size from Eq. 3-70 has to be redefined by

$$N = (2M\omega_{max} + 1)N_r. \qquad \text{Eq. 3-74}$$

Working Equations

For a numerical calculation of the Fourier coefficients, the integrals in Eq. 3-69 have to be approximated by sums to realize their computation. The computational procedure of the FAST method is performed by steps outlined in Figure 3.13 as suggested in (Koda et al. 1979).



INPUTS

- Parameters to be varied $x_i, i = 1, 2, \dots, n$
- Search curves $G_i, i = 1, 2, \dots, n$
- Frequency set $\omega_i, i = 1, 2, \dots, n$
- Sampling size $N \geq 2M\omega_{\max} + 1$

FAST

```
1. Assign frequencies to parameters
2. Loop for $s$ from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$
     Loop for $i$ from 1 to $n$
       Calculate $x_i(s) = G_i(\sin(\omega_i s))$
     end loop
     Simulate model for parameter set $x(s)$
     Calculate objective function value $Q\big(x(s)\big)$
   end loop
3. Calculate mean, variance, and standard
   deviation of $Q$
4. Loop for $i$ from 1 to $n$
     Calculate Fourier coefficients $A_{q\omega_i}$ and $B_{q\omega_i}$
     Calculate sensitivity indices $S_i$
   end loop
5. Rank parameters according to sensitivity
   indices
```

Figure 3.13: Implementation of the FAST-method without random phase-shift

3.3.3 EXTENDED FOURIER AMPLITUDE SENSITIVITY TEST

The FAST method provides a way to estimate the first-order sensitivity indices in Eq. 3-34. But it is also desirable to get knowledge about the residual variance $Var(Q) - \sum_{i=1}^{n} V_i$ the part of the variance that is not caused by first-order effects, i.e. it includes all higher-order effects of the interactions between the parameters. The apportion of this residual variance to various parameter combinations demands the analysis of all linear combinations among the frequencies. The computational complexity of this procedure is the reason why the FAST method has never be exploited for higher-order indices. Instead Saltelli et al. have proposed an extension of the FAST method to evaluate the total sensitivity indices in Eq. 3-36. This total sensitivities give indeed not a full characterization of the underlying system but they allow a full quantification of the impact of each parameter x_i (Saltelli et al. 1999).

The basic idea is to assign a frequency ω_i to a parameter x_i and a set of almost identical frequencies $\omega_{\sim i}$, but different from ω_i, to all other parameters $x_{\sim i}$. The partial variance $V_{\sim i}$ can then be estimated by means of Eq. 3-66

$$V_{\sim i} = 2\sum_{q=1}^{M} A_{q\omega_{\sim i}}^2 + B_{q\omega_{\sim i}}^2. \qquad \text{Eq. 3-75}$$

This partial variance $V_{\sim i}$ comprises all effects of any order that do not involve the parameter x_i. The total sensitivity indices according to Eq. 3-36 are then given by

$$S_{Ti} = 1 - \frac{V_{\sim i}}{Var(Q)} = 1 - \frac{2\sum_{q=1}^{M} A_{q\omega_{\sim i}}^2 + B_{q\omega_{\sim i}}^2}{Var(Q)}. \qquad \text{Eq. 3-76}$$

This approach has the advantage that for each parameter x_i only two frequencies ω_i and $\omega_{\sim i}$ have to be selected. The total number of model evaluations that is needed for a complete sensitivity analysis is given by

$$N = n(2M\omega_{max} + 1)N_r, \qquad \text{Eq. 3-77}$$

where $\omega_{\max} = \max(\omega_i, \omega_{\sim i}) = \omega_i$.

Usually, a high value is assigned to ω_i and a low value to $\omega_{\sim i}$; the best choice would be $\omega_{\sim i} = 1$ but then the curve is very sparse space-filling. Saltelli et al. developed an algorithm to select these frequencies (Saltelli et al. 1999). At first, the maximum allowable frequency of the complementary set $\{\omega_{\sim i}\}$ is set to

$$max\{\omega_{\sim i}\} = \frac{\omega_i}{2M}. \qquad \text{Eq. 3-78}$$

The frequencies of the complementary set are then chosen so that the whole range $[1, \omega_{\sim i}]$ is covered according to the conflicting requirements:

1. the step between two consecutive frequencies must be as large as possible, and
2. the number of parameters assigned with the same frequency must be as low as possible.

It can be shown that the lowest sample size that can be used is 65 with $\omega_i = 8$. Saltelli et al. recommended choosing the values for the frequency ω_i and the resampling size N_r corresponding to Figure 3.14 by a given sample size N (Saltelli et al. 1999). Within the suggested region, the ratio ω_i/N_r varies between 16 and 64 and all possible choices are equivalent for a given N. To take values outside of this suggested region is not recommended because

- if ω_i is low and N_r is high, then the sample over each curve is too sparse, and
- if ω_i is high and N_r is low, then the sample is too dense over a small number of closed paths.

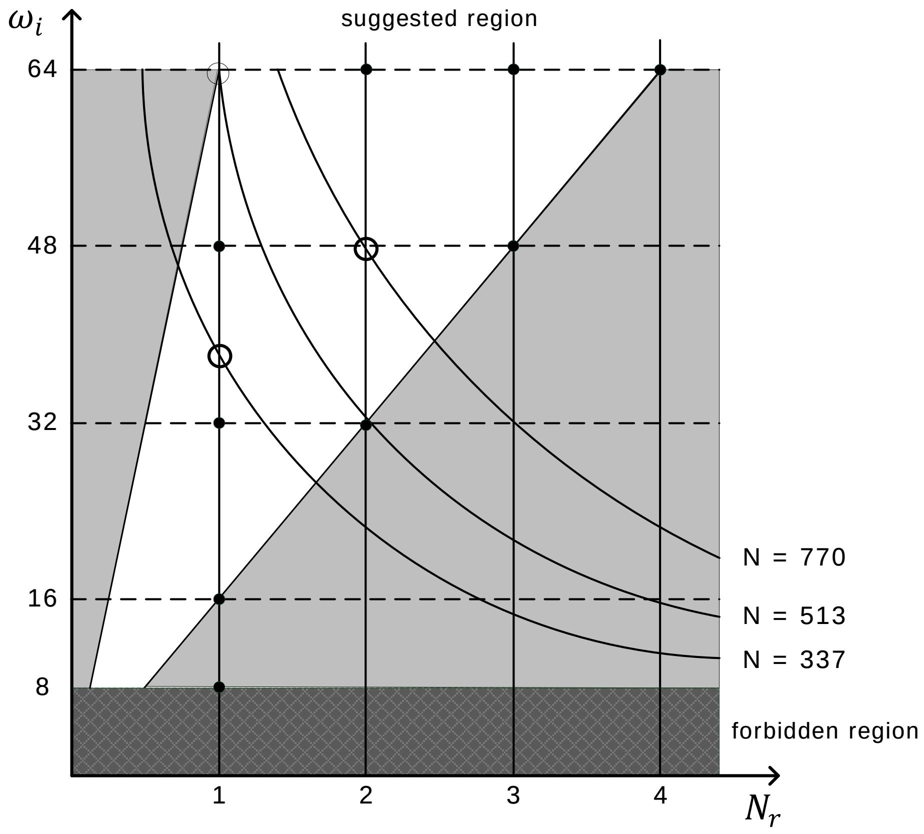


Figure 3.14: Recommended region for choosing the values of the frequency ω_i and the resampling size N_r by a given sample size N (Saltelli et al. 1999)

4 PETRI NETS

The implementation and programming of the Petri net elements by means of the Modelica language demands a clear definition of the underlying formalism and the corresponding processes such as activation, enabling, and firing. Despite several works and publications with Petri net approaches, there is a serious problem according to the lacking unity of concepts, notations, and terminologies. Every author personal definitions which are partly not precise enough, not common, or contradictory. Hence, this section comprises all definitions of the necessary Petri nets concepts, abbreviations, and extensions as well as the definitions of the corresponding processes which are essential for the simulation. Beginning with the Petri net introduced firstly by Carl Adam Petri in 1962 (Petri 1962), the formalism is successively extended and simplified through to e**x**tended **H**ybrid **P**etri **N**ets for biological applications (xHPN) to model nearly all kinds of biological reactions.

Thereby, it is always mentioned who proposed the respective Petri net concept and afterwards the corresponding definitions used in this work are given. These can differ from the original ones to meet the requirements of biologists and retain the basic Petri net logics at every expansion step (see Section 5.2).

4.1 BASIC CONCEPTS

The **Petri net** formalism for graphical modeling and visualization of concurrency, parallelism, synchronization, resource sharing, and non-determinism was first introduced by Carl Adam Petri in 1962 (Petri 1962). A Petri net is mathematically a directed and bipartite graph. The property bipartite implies the division into two unique sets of nodes, called **transitions** and **places**; thereby, places and transitions are connected by arcs. An **arc** is a directed edge that connects either places to transitions or transitions to places; however, connections between transitions or places among each other are not allowed according to the bipartite attribute. Places are graphically represented by circles and the transitions by boxes (see Figure 4.1).

Petri nets can be found in many different application fields like computer communications, business or production processes, operating systems, traffic light crossings, or systems biology. In this connection, a place models a state, for example, of an object or a condition while a transition models the change of several states, for example, activities or events.

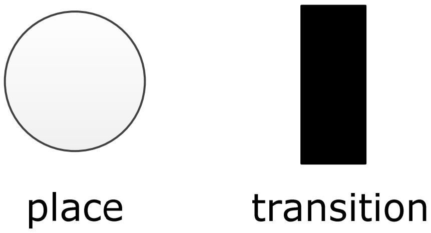


Figure 4.1: Graphical representation of a place (left) and a transition (right)

This section serves as a basis for the implementation of the ordinary Petri net concept and its abbreviations and extensions which are necessary to model biological processes. A**bbreviations** simplify the Petri net representation but can always be transformed to an ordinary Petri net. On the other hand, **extensions** cannot be represented by ordinary Petri nets and allow the usage of the Petri net formalism in a wider range of applications.

Definition 4.1 (net)

A **net** is the tuple (P, T, F, G) of a finite set of places $P = \{p_1,\ p_2, \dots, p_p\}$, a finite set of transitions $T = \{t_1,\ t_2, \dots, t_t\}$, where $P \cap T = \emptyset$, a set $F \subseteq (P \times T)$ of arcs from places to transitions, and a set $G \subseteq (T \times P)$ of arcs from transitions to places.

Definition 4.2 (Petri net)

The tuple (P, T, F, G, f) is a **Petri net** if (P, T, F, G) is a net and $f\colon (F \cup G) \to \mathbb{N}_0$ is an arc weight function which assigns each arc a non-negative integer, whereby $(p_i \to t_j)$ denotes the arc from a place $p_i \in P$ to a transition $t_j \in T$ and $f(p_i \to t_j)$ is the corresponding weight and $(t_j \to p_i)$ denotes the arc from t_j to p_i with the weight $f(t_j \to p_i)$.

Every place in a Petri net can contain an integer number of **tokens**. These tokens are graphically represented by little black dots or numbers in the places (see Figure 4.2). A concrete determination of the token number of a place is called **marking** of the place and a concrete determination of the token numbers of all places in a Petri net is called marking of the Petri net. If a Petri net contains only marked places, then it is called initially marked Petri net or following just Petri net.

Definition 4.3 (marking)

The tuple (P, T, F, G, f) is a Petri net. A map $m: P \rightarrow \mathbb{N}_0$ is called **marking of the Petri net** and assigns each $p_i \in P$ a concrete token number $m(p_i)$, called **marking of the place** p_i. The map $m_0: P \rightarrow \mathbb{N}_0$ is the **initial marking of the Petri net** and the tuple (P, T, F, G, f, m_0) is called **(initially marked) Petri net.**

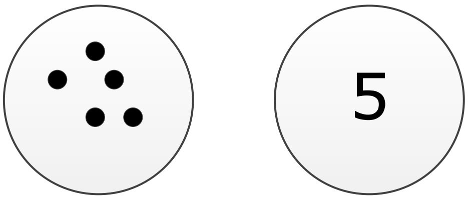


Figure 4.2: Tokens can be graphically represented by dots (left) or by numbers (right) in the places

Definition 4.4 (input, output)

The **set of inputs** of a Petri net element $x \in (P \cup T)$ are defined as $Y_{in}(x) \coloneqq \{y \in (P \cup T) | (y \rightarrow x) \in F \cup G\}$ and the **set of outputs** are $Y_{out}(x) \coloneqq \{y \in (P \cup T) | (x \rightarrow y) \in F \cup G\}$. The set of all input places of a transition $t_j \in T$ is noted with $P_{in}(t_j) \subseteq P$ and the set of all output places is noted with $P_{out}(t_j) \subseteq P$. Similarly, the set $T_{in}(p_i) \subseteq T$ contains all input transitions of a place $p_i \in P$ and the set $T_{out}(p_i) \subseteq T$ contains all output transitions (see Figure 4.3). The number of inputs is noted by n_{in} and the number of outputs by n_{out}.

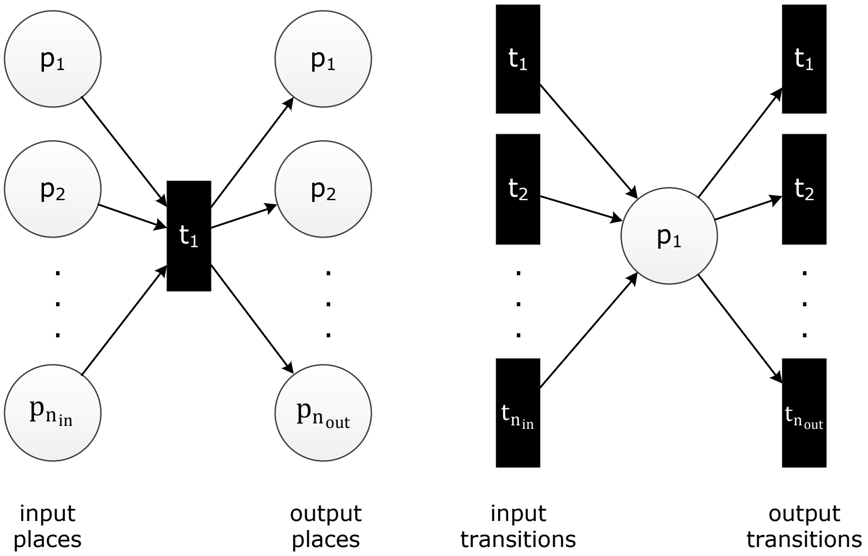


Figure 4.3: Input and output places, and input and output transitions

A transition is said to be active with regard to a concrete marking if all input places have at least as many tokens as their arc weights.

Definition 4.5 (activation Petri net)

The tuple (P, T, F, G, f, m_0) is a Petri net. A transition $t_j \in T$ is **active** with regard to a concrete marking m if and only if

$$\forall p_i \in P_{in}(t) : \; m(p_i) \geq f(p_i \rightarrow t_j).$$

The set of all active input transitions of a place $p_i \in P$ is denoted by $TA_{in}(p_i)$ and the set of all active output transitions is denoted by $TA_{out}(p_i)$.

Active transitions can be enabled by connected places; if a transition is enabled by all its input places, it is **firable** and can fire tokens. Possibly, a place has not enough tokens to enable all output transitions. This is the case when its token number is less than the sum of output arc weights and a general conflict arises that has to be resolved.

Different possible approaches are available; two of them, **priority** and **probabilistic choice** are used within this work which accordingly have been defined and adapted based on the suggestions in (David and Alla 2005).

Definition 4.6 (general conflict)

The tuple (P, T, F, G, f, m_0) is a Petri net. A place $p_i \in P$ has a **general conflict** with regard to a concrete marking m if

$$m(p_i) < \sum_{t_j \in TA_{out}(p_i)} f(p_i \rightarrow t_j),$$

whereby the set $TA_{out}(p_i)$ contains two or more active output transitions.

Example 4.1

The places P_1 and P_2 of the Petri net in Figure 4.4 have a general conflict. P_1 can either enable T_1 or T_2 and P_2 can either enable $T2$ or $T3$. Transition $T4$ is not an option because it is not active due to missing tokens in P_3.

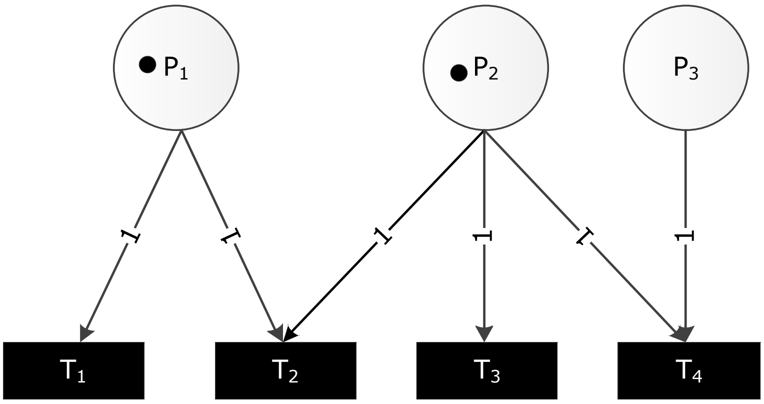


Figure 4.4: General conflicts of discrete places (Example 4.1)

Definition 4.7 (Petri net with priorities)

The tuple $(P, T, F, G, f, m_0, \mathcal{u})$ is a **Petri net with priorities** if (P, T, F, G, f, m_0) is a Petri net and $\mathcal{u}: F \rightarrow \{1,2, \dots, \max(n_{out})\}$ is a priority function which assigns every arc from a place $p_i \in P$ to a transition $t_j \in T_{out}(p)$ an enabling priority $\mathcal{u}(p_i \rightarrow t_j)$ under the condition that each priority from 1 to n_{out} is only used once for each place

$$\mathcal{u}(p_i \rightarrow t_j) \neq \mathcal{u}(p_i \rightarrow t_k)\ \forall t_j, t_k \in T_{out}(p_i)$$

and that 1 is the best priority and n_{out} is the worst priority.

Definition 4.8 (Petri net with probabilities)

The tuple $(P, T, F, G, f, m_0, \mathcal{w})$ is a **Petri net with probabilities** if (P, T, F, G, f, m_0) is a Petri net and $\mathcal{w}: F \rightarrow [0,1]$ is a probability function which assigns every arc from a place $p_i \in P$ to a transition $t_j \in T_{out}(p_i)$ an enabling probability $\mathcal{w}(p_i \rightarrow t_j)$ under the condition that the sum of probabilities is equal to one

$$\sum_{t_j \in T_{out}(p_i)} \mathcal{w}(p_i \rightarrow t_j) = 1 \quad \forall p_i \in P\,.$$

The deterministic conflict resolution with priorities requires that all output transitions of a place are provided with a priority from 1 to n_{out}. The transition with the priority 1 is proven at first. If it is active and the arc weight does not exceed the marking of the place, it is enabled by the place. Then the second prioritized transition is checked; if it is active and the arc weight sum is not greater than the marking, it is also enabled. The next transition proven is the one provided with the third priority and so on.

Definition 4.9 (enabling process Petri net with priorities)

The tuple $(P, T, F, G, f, m_0, \mathcal{u})$ is a Petri net with priorities. A place $p_i \in P$ **enables** with regard to a concrete marking m a subset, denoted by $TE_{out}(p_i)$, of the transitions in $TA_{out}(p_i)$. If p_i has no general conflict according to Definition 4.6, it enables all transitions in $TA_{out}(p_i)$, i.e.

$$TE_{out}(p_i) \equiv TA_{out}(p_i).$$

Otherwise, p_i enables as many transitions of the set $TA_{past}(p_i)$ as possible according to the priorities $\mathcal{u}(p_i \rightarrow t_j)$ so that the condition

$$m(p_i) \geq \sum_{t_j \in TE_{out}(p_i)} f(p_i \rightarrow t_j)$$

is still fulfilled.

The conflict resolution by probabilities proceeds non-deterministically. A probability is assigned to every output transition of a place. The enabled transitions are selected by a random process with due regard to the probabilities; thereby, it always has to be ensured that the current arc weight sum does not exceed the token number of the place.

Definition 4.10 (enabling process Petri net with probabilities)

The tuple $(P, T, F, G, f, m_0, \mathscr{w})$ is a Petri net with probabilities. A place $p_i \in P$ **enables** with regard to a concrete marking m a subset, denoted by $TE_{out}(p_i)$, of the transitions in $TA_{out}(p_i)$. If p_i has no general conflict according to Definition 4.6, it enables all transitions in $TA_{out}(p_i)$, i.e.

$$TE_{out}(p_i) \equiv TA_{out}(p_i).$$

Otherwise, p_i enables randomly as many transitions of the set $TA_{out}(p_i)$ as possible according to the probabilities $\mathscr{w}(p_i \to t_j)$ so that the condition

$$m(p_i) \geq \sum_{t_j \in TE_{out}(p_i)} f(p_i \to t_j)$$

is still fulfilled.

Remark 4.1

All output transitions of a place are provided with a probability and the sum is equal to one. If not all output transitions are active, the probabilities have to be scaled by the following expression

$$\tilde{\mathscr{w}}(p_i \to t_j) = \frac{\mathscr{w}(p_i \to t_j)}{\sum_{t_k \in TA_{out}(p_i)} \mathscr{w}(p_i \to t_k)}.$$

It is also possible to mix the two concepts; then one part of the places enables their output transitions deterministically according to the assigned priorities and the other part randomly according to the assigned probabilities. Petri nets with priorities and probabilities are called simplified resolved Petri nets hereafter.

Definition 4.11 (resolved Petri net)

The tuple $(P, T, F, G, f, e, \mathscr{p}, m_0)$ is a Petri net with priorities and probabilities, simplified called **resolved Petri net** hereafter, if (P, T, F, G, f, m_0) is a Petri net, $e: P \to \{prio, prob\}$ is the resolution function that assigns every place either the resolution type priority or probability, and $\mathscr{p}: F \to \{\{1,2, \ldots, \max(n_{out})\}: e(p_i) = prio,\ [0,1]: e(p_i) = prob\}$ is an enabling function which assigns every arc from a place $p_i \in P$ to a transition $t_j \in T_{out}(p)$ either an enabling priority or a probability $\wp(p_i \rightarrow t_j)$ according to the resolution type of the place and under the condition that each priority from 1 to n_{out} is only used once for each place p_i with $e(p_i) = prio$

$$\wp(p_i \rightarrow t_j) \neq \wp(p_i \rightarrow t_k) \, \forall t_j, t_k \in T_{out}(p_i)$$

and that 1 is the best priority and n_{out} is the worst priority and that the sum of probabilities is equal to one for each place p_i with $e(p_i) = prob$

$$\sum_{t_j \in T_{out}(p_i)} \wp\big(p_i \rightarrow t_j\big) = 1\,.$$

An active transition is firable if it is enabled by all input places. It fires by removing as much tokens as the arc weights from all input places and by adding as many tokens as the arc weights to all output places.

Definition 4.12 (firability Petri net)

The tuple $(P, T, F, G, f, e, \wp, m_0)$ is a resolved Petri net. An active transition $t_j \in T$ is **firable** if and only if

$$\forall\, p_i \in P_{in}\big(t_j\big){:}\;\; t_j \in TE_{out}(p_i).$$

When one or more transitions of a Petri net fire, the markings of the involved places have to be recalculated; thereby, the output firing sum is subtracted from the token number and the input firing sum is added to it.

Definition 4.13 (firing process Petri net)

The tuple $(P, T, F, G, f, e, \wp, m_0)$ is a resolved Petri net. A firable transition $t_j \in T$ **fires** with regard to a concrete marking m by removing as many tokens as the arc weights from all input places

$$m'(p_i) = m(p_i) - f(p_i \rightarrow t_j) \quad \forall\, p_i \in P_{in}\big(t_j\big)$$

and by adding as many tokens as the arc weights to all output places

$$m'(p_i) = m(p_i) + f(p_i \rightarrow t_j) \quad \forall\, p_i \in P_{out}\big(t_j\big).$$

The new marking $m'(p_i)$ of the place $p_i \in P$ is recalculated by the following algebraic equation

$$m'(p_i) = m(p_i) - \sum_{t_j \in TF_{out}(p_i)} f(p_i \rightarrow t_j) + \sum_{t_j \in TF_{in}(p_i)} f(t_j \rightarrow p_i)\,,$$

whereby $TF_{out}(p_i) \subseteq T$ is the set of all firing output transitions and $TF_{in}(p_i) \subseteq T$ is the set of all firing input transitions. The sums $\sum_{t_j \in TF_{out}(p_i)} f(p_i \to t_j)$ and $\sum_{t_j \in TF_{in}(p_i)} f(t_j \to p_i)$ are called **output firing sum** and **input firing sum**, respectively.

Example 4.2

Figure 4.5 displays an example of a Petri net with seven places $\{P1, P2, P3, P4, P5, P6, P7\}$ and four transitions $\{T1, T2, T3, T4\}$. The numbers at the arcs are the weights and the black dots in the places are the tokens. Place $P1$ has two tokens ($m_0(P1) = 2$), $P2$ has five tokens ($m_0(P2) = 5$), $P4$ has one token ($m_0(P4) = 1$), and $P7$ has two tokens ($m_0(P7) = 2$). All others have no tokens ($m_0(P3) = m_0(P5) = m_0(P6) = 0$) and, hence, the initial marking of this Petri net is $m_0 = (2,5,0,1,0,0,2)$.

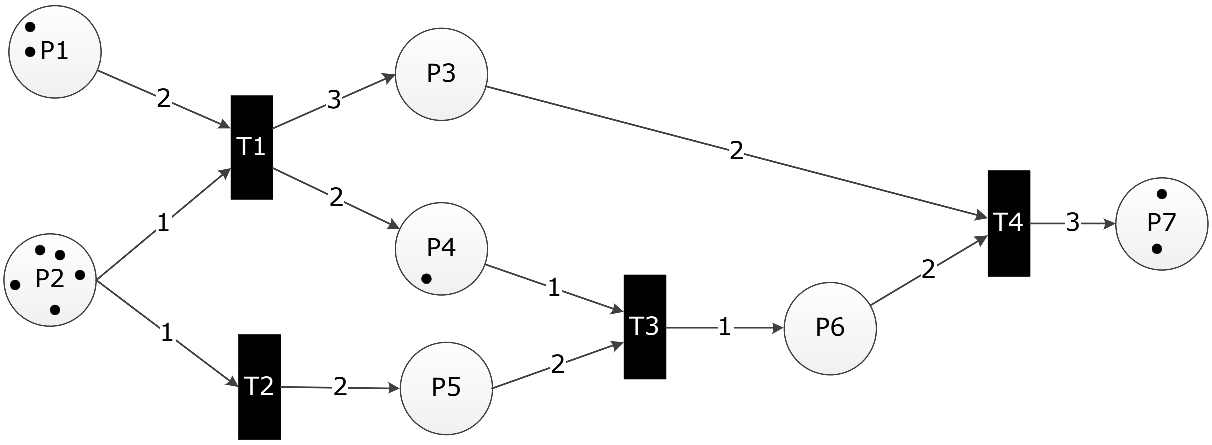


Figure 4.5: Petri net (Example 4.2)

Activation:

- $T1$: The input places are $P1$ and $P2$. $P1$ must have at least two tokens (weight of the arc from $P1$ to $T1$ ($f(P1 \to T1) = 2$)) which it actually has and $P2$ must have at least one token ($f(P2 \to T1) = 1$) and it has five; hence, transition $\boldsymbol{T1}$ **is active**.
- $T2$: The only input place is $P2$. $P2$ must have at least one token ($f(P2 \to T2) = 1$) which is fulfilled; hence, transition $\boldsymbol{T2}$ **is active**.
- $T3$: The input places are $P4$ and $P5$. $P4$ must have at least one token ($f(P4 \to T3) = 1$) which it actually has but $P5$ must have at least two tokens and it has none; hence, transition $\boldsymbol{T3}$ **is not active**.
- $T4$: The input places are $P3$ and $P6$. Both places have no tokens; hence, transition $\boldsymbol{T4}$ **is not active**.

Enabling:

- $P1$ enables its only output transition $T1$.

- $P2$ has enough tokens to enable both output transitions $T1$ and $T2$ $(m(P2) = 5 \geq f(P2 \rightarrow T1) + f(P2 \rightarrow T2) = 2)$. There is no general conflict.

Firing: T1 and T2 are firable and fire by

- Removing 2 $(= f(P1 \rightarrow T1))$ tokens from $P1$
- Removing 2 $(= f(P2 \rightarrow T1) + f(P2 \rightarrow T2))$ tokens from $P2$
- Adding 3 $(= f(T1 \rightarrow P3))$ tokens to $P3$
- Adding 2 $(= f(T1 \rightarrow P4))$ tokens to $P4$
- Adding 2 $(= f(T2 \rightarrow P5))$ tokens to $P5$.

The new marking of the Petri net is $m' = (0,3,3,3,2,0,2)$ and it is displayed in Figure 4.6. Now the transitions $T2$ and $T3$ are active and can be enabled by all their input places. Hence, $T2$ and $T3$ are firable and the firing of $T2$ and $T3$ leads to the new marking $m'' = (0,2,3,2,2,1,2)$.

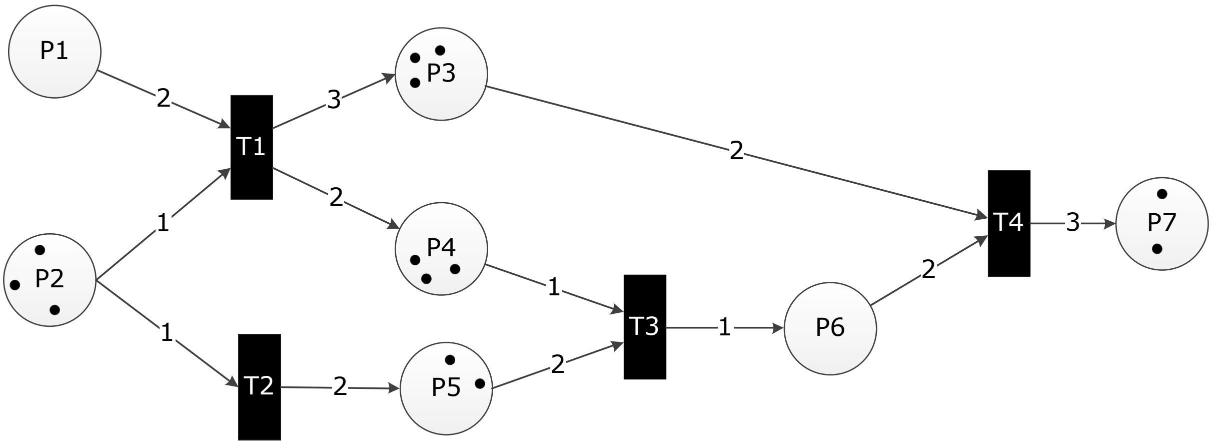


Figure 4.6: Petri net of Figure 4.5 after firing $T1$ and $T2$ (Example 4.2)

Example 4.3

Both transitions of the Petri net example in Figure 4.7 are active because of

$$m(P1) = 2 \geq f(P1 \rightarrow T1) = 1$$

and

$$m(P1) = 2 \geq f(P1 \rightarrow T2) = 2.$$

But $P1$ can only enable one of them due to its token number $(m(P1) = 2 \ngeq f(P1 \rightarrow T1) + f(P1 \rightarrow T2) = 3)$. Hence, $P1$ has a general conflict. If the enabling is performed by priorities and $T2$ has priority 1 $(\wp(P1 \rightarrow T2) = 1)$ and $T1$ has priority 2 $(\wp(P1 \rightarrow T1) = 2)$, $P1$ enables $T2$. $T2$ is firable and fires by removing two tokens from $P1$ and adding one to $P3$. If the enabling is performed at random, one of the transitions is enabled according to their assigned probabilities, e.g. $T1$ has the probability $\wp(P1 \rightarrow T1) = 0.7$ and $T2$ has the probability $\wp(P1 \rightarrow T2) = 0.3$.

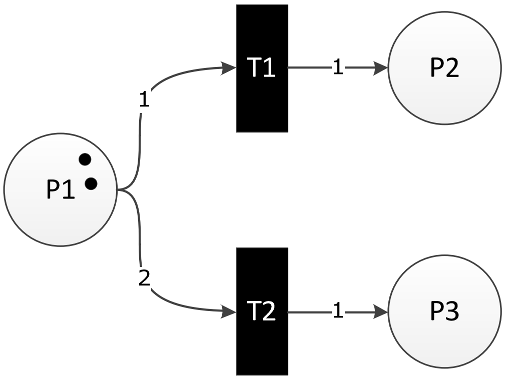


Figure 4.7: Petri net with a general conflict (Example 4.3)

Example 4.4

All transitions of the Petri net in Figure 4.8 are active but $P1$ and $P2$ can only enable one of their output transitions due to their token numbers. Four cases can occur if probabilistic enabling is applied:

- Case 1: $P1$ enables $T1$ and $P2$ enables $T3$

  $T1$ and $T3$ are firable and fire by removing one token from $P1$ and $P2$ and adding one to $P3$ and $P5$. The new marking of the Petri net is $m' = (0,0,1,0,1)$.
- Case 2: $P1$ enables $T1$ and $P2$ enables $T2$

  Only $T1$ is firable and fires by removing one token from $P1$ and adding one to $P3$. The new marking of the Petri net is $m' = (0,1,1,0,0)$.
- Case 3: $P1$ enables $T2$ and $P2$ enables $T3$

  Only $T3$ is firable and fires by removing one token from $P2$ and adding one to $P5$. The new marking of the Petri net is $m' = (1,0,0,0,1)$.
- Case 4: $P1$ enables $T2$ and $P2$ enables $T2$

  $T2$ is firable and fires by removing one token from $P1$ and $P2$ and adding one to $P4$. The new marking of the Petri net is $m' = (0,0,0,1,0)$.

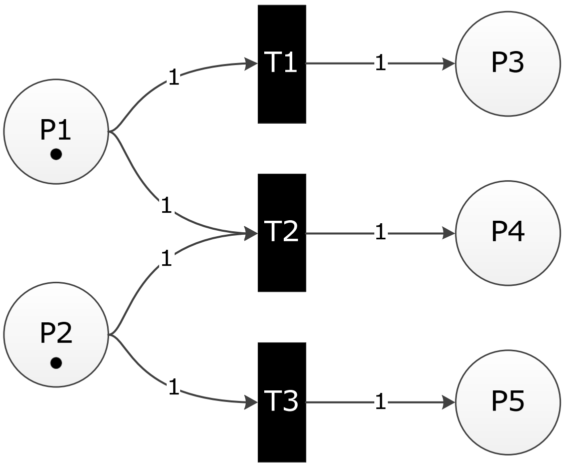


Figure 4.8: Petri net with general conflicts (Example 4.4)

4.2 Abbreviations and Extensions of the Basic Concepts

Abbreviations and extensions of the mentioned Petri net concepts are introduced and defined to use the formalism within a wider range of biological applications and to simplify the modeling process and the visualization. Thereby, **abbreviations** simplify the graphical representation of a Petri net, i.e. they can be also modeled with the basic concept but improve the visualization. **Extensions**, however, enhance the basic concept in order to extend the application field (cf. (David and Alla 2005)).

4.2.1 Capacitive Petri Nets

The basic Petri net concept is modified such that every place is provided with a lower and upper limit of tokens that it can contain. By this modification, the firing of transitions is only possible if these minimum and maximum capacities are not infringed upon. All capacitive Petri nets can be converted to basic Petri nets but the introduction of capacities simplifies the representation of the Petri net model and is, thus, an abbreviation of the basic concept (cf. (David and Alla 2005)).

This abbreviation offers, for example, a simpler way for integrating biological knowledge about minimum and maximum quantities of biological compounds.

Definition 4.14 (capacitive Petri net)

The tuple $(P, T, F, G, f, c_l, c_u, m_0)$ is called **capacitive Petri net** if (P, T, F, G, f, m_0) is Petri net, the map $c_l: P \to \mathbb{N}_0$ assigns a **minimum capacity** $c_l(p_i)$ to every place $p_i \in P$ and the map $c_u: P \to \mathbb{N}_0$ assigns a **maximum capacity** $c_u(p_i)$ to every place $p_i \in P$, whereby the initial marking m_0 must satisfy the condition

$$c_l(p_i) \leq m_0(p_i) \leq c_u(p_i) \quad \forall\, p_i \in P.$$

Capacitive Petri nets require a redefinition of the basic activation process (Definition 4.5) to take the capacities into account. A transition of a capacitive Petri net is active if the token numbers of all input places do not fall below their minimum capacities after removing the arc weights. Additionally, the maximum capacities of all output places may not be exceeded by adding the arc weights.

Definition 4.15 (activation capacitive Petri net)

The tuple $(P, T, F, G, f, c_l, c_u, m_0)$ is a capacitive Petri net. A transition $t_j \in T$ is **active** with regard to a concrete marking m if and only if

$$\forall\, p_i \in P_{in}(t_j):\ m(p_i) - f(p_i \to t_j) \geq c_l(p_i)$$

and

$$\forall\, p_i \in P_{out}(t_j):\ m(p_i) + \ f(t_j \to p_i) \leq c_u(p_i).$$

A place in a capacitive Petri net can have two different conflicts, called general input conflict and general output conflict. A general output conflict occurs when the place has not enough tokens to enable all output transitions due to its minimum capacity and a general input conflict can appear when the place cannot receive tokens from all active input transitions due to its maximum capacity.

Definition 4.16 (general output conflict, general input conflict)

The tuple $(P, T, F, G, f, c_l, c_u, m_0)$ is a capacitive Petri net. A place $p_i \in P$ has a **general output conflict** with regard to a concrete marking m if

$$m(p_i) - \sum_{t_j \in TA_{out}(p_i)} f(p_i \to t_j) < c_l(p_i),$$

whereby the set $TA_{out}(p_i)$ contains two or more active output transitions.

A place $p_i \in P$ has a **general input conflict** with regard to a concrete marking m if

$$m(p_i) + \sum_{t_j \in TA_{in}(p_i)} f(t_j \to p_i) > c_u(p_i),$$

whereby the set $TA_{in}(p_i)$ contains two or more active input transitions.

These conflict situations are resolved in this work by priorities or probabilities as previously mentioned (Definition 4.7 and Definition 4.8). Therefore, the enabling processes of Definition 4.9 and Definition 4.10 have been redefined to include the capacities.

Definition 4.17 (capacitive Petri net with priorities)

The tuple $(P, T, F, \mathrm{G}, f, c_l, c_u, \mathcal{u}, m_0)$ is a **capacitive Petri net with priorities** if $(P, T, F, \mathrm{G}, f, c_l, c_u, m_0)$ is a capacitive Petri net, $\mathcal{u}: (F \cup G) \to \{1,2, \dots, \max(n_{out}, n_{in})\}$ is a priority function which assigns every arc from and to a place $p_i \in P$ an enabling priority $\mathcal{u}(p_i \to t_j)$ and $\mathcal{u}(t_j \to p_i)$, respectively, under the condition that each priority from 1 to n_{out} is only used once for the output transtions of a place

$$\mathcal{u}(p_i \to t_j) \neq \mathcal{u}(p_i \to t_k), \qquad \forall t_j, t_k \in T_{out}(p_i),$$

that each priority from 1 to n_{in} is only used once for the input transtions of a place

$$\mathcal{u}(t_j \to p_i) \neq \mathcal{u}(t_k \to p_i), \qquad \forall t_j, t_k \in T_{in}(p_i)$$

and that 1 is the best priority and n_{out} and n_{in}, respectively, is the worst priority.

Definition 4.18 (capacitive Petri net with probabilities)

The tuple $(P, T, F, G, f, c_l, c_u, \mathcal{w}, m_0)$ is a **capacitive Petri net with probabilities** if $(P, T, F, \mathrm{G}, f, c_l, c_u, m_0)$ is a capacitive Petri net, $\mathcal{w}: (F \cup G) \to [0,1]$ is a probability function which assigns every arc from and to a place $p_i \in P$ an enabling probability $\mathcal{w}(p_i \to t_j)$ and $\mathcal{w}(t_j \to p_i)$, respectively, under the condition that the sum of probabilities for the output and input transitions, respectively, is equal to one for every place p_i

$$\sum_{t_j \in T_{out}(p_i)} \mathcal{w}(p_i \to t_j) = 1 \ , \ \sum_{t_j \in T_{in}(p_i)} \mathcal{w}(t_j \to p_i) = 1.$$

In a capacity Petri net with priorities, all output transitions of a place are provided with a priority from 1 to n_{out}. The transition with the priority 1 is proven at first. It has to be active and the arc weight may not infringe the minimum capacity. Then, the second prioritized transition is checked. It has to be active and the removing of the arc weight sum may not violate the minimum capacity. The transition with the third priority is proven next and so on. When all places have enabled their output transitions, they enable a subset of their active input transitions; but only those can be enabled which are already enabled by all input places. All input transitions of a place are provided with a priority from 1 to n_{in}. The transition with the priority 1 is proven at first. It has to be enabled by all input places and the arc weight may not infringe on the maximum capacity. Then, the second prioritized transition is checked. It has to be enabled by all input places and the addition of the arc weight sum may not violate the maximum capacity. Afterwards the transition with the third priority is proven and so on.

Definition 4.19 (enabling process capacitive Petri net with priorities)

The tuple $(P, T, F, G, f, c_l, c_u, \mathcal{u}, m_0)$ is a capacitive Petri net with priorities. A place $p_i \in P$ **enables** with regard to a concrete marking m a subset, denoted by $TE_{out}(p_i)$, of the transitions in $TA_{out}(p_i)$. If p_i has no general output conflict according to Definition 4.16, it enables all transitions in $TA_{out}(p_i)$, i.e.

$$TE_{out}(p_i) \equiv TA_{out}(p_i).$$

Otherwise, p_i enables as many transitions of the set $TA_{out}(p_i)$ as possible according to the priorities $\mathcal{u}(p_i \to t_j)$ so that the condition

$$m(p_i) - \sum_{t_j \in TE_{out}(p_i)} f(p_i \to t_j) \geq c_l(p_i)$$

is still fulfilled.

After the enabling of output transitions, all places $p_i \in P$ enable with regard to a concrete marking m a subset, denoted by $TE_{in}(p_i)$, of their active input transitions restricted to those that are already enabled by all their input places

$$TAe_{in}(p_i) = \{t_j \in TA_{in}(p_i) \ \wedge \ t_j \in TE_{out}(p_k) \ \ \forall \, p_k \in P_{in}(t_j)\}.$$

If p_i has no general input conflict according to Definition 4.16, it enables all transitions in $TAe_{in}(p_i)$, i.e.

$$TE_{in}(p_i) \equiv TAe_{in}(p_i).$$

Otherwise, p_i enables as many transitions of the set $TAe_{in}(p_i)$ as possible according to the priorities $\mathcal{u}(t_j \to p_i)$ so that the condition

$$m(p_i) + \sum_{t_j \in TE_{in}(p_i)} f(t_j \to p_i) \leq c_u(p_i)$$

is still fulfilled.

The enabling process with probabilities is performed in a similar manner as that with priorities. The only difference is that the order by which the transitions are proven is determined at random. Thereby, the transition with the highest probability has the highest chance to be proven first.

Definition 4.20 (enabling process capacitive Petri net with probabilities)

The tuple $(P, T, F, G, f, c_l, c_u, \mathcal{w}, m_0)$ is a capacitive Petri net with probabilities. A place $p_i \in P$ **enables** with regard to a concrete marking m a subset, denoted by $TE_{out}(p_i)$, of the transitions in $TA_{out}(p_i)$. If p_i has no general output conflict according to Definition 4.16, it enables all transitions in $TA_{out}(p_i)$, i.e.

$$TE_{out}(p_i) \equiv TA_{out}(p_i).$$

Otherwise, p_i randomly enables as many transitions of the set $TA_{out}(p_i)$ as possible according to the probabilities $\mathcal{w}(p_i \to t_j)$ so that the condition

$$m(p_i) - \sum_{t_j \in TE_{out}(p_i)} f(p_i \to t_j) \geq c_l(p_i)$$

is still fulfilled.

After the enabling of output transitions, all places $p_i \in P$ enable with regard to a concrete marking m a subset, denoted by $TE_{in}(p_i)$, of their active input transitions restricted to those that are already enabled by all their input places

$$TAe_{in}(p_i) = \{t_j \in TA_{in}(p_i) \ \land \ t_j \in TE_{out}(p_k) \ \forall \, p_k \in P_{in}(t_j)\}.$$

If p_i has no general input conflict according to Definition 4.16, it enables all transitions in $TAe_{in}(p_i)$, i.e.

$$TE_{in}(p_i) \equiv TAe_{in}(p_i).$$

Otherwise, p_i enables randomly as many transitions of the set $TAe_{in}(p_i)$ as possible according to the probabilities $w(t_j \to p_i)$ so that the condition

$$m(p_i) + \sum_{t_j \in TE_{in}(p_i)} f(t_j \to p_i) \leq c_u(p_i)$$

is still fulfilled.

Remark 4.2

All output transitions of a place are provided with a probability and the sum is equal to one. If not all output transitions are active, the probabilities have to be scaled by the following expression

$$\tilde{w}(p_i \to t_j) = \frac{w(p_i \to t_j)}{\sum_{t_k \in TA_{out}} w(p_i \to t_k)}.$$

Similarly, if not all input transitions are enabled by all input places, the probabilities have to be scaled by the following expression

$$\tilde{w}(t_j \to p_i) = \frac{w(t_j \to p_i)}{\sum_{t_k \in TAe_{in}} w(t_k \to p_i)}.$$

It also possible to mix both concepts, enabling by priorities and probabilities, in one Petri net as mentioned before (see Definition 4.11). Then, every place can be assigned with its own resolution type.

Definition 4.21 (resolved capacitive Petri net)

The tuple $(P, T, F, \mathrm{G}, f, c_l, c_u, e, \wp, m_0)$ is a **resolved capacitive Petri net** if $(P, T, F, \mathrm{G}, f, c_l, c_u, m_0)$ is a capacitive Petri net, $e: P \to \{prio, prob\}$ is the resolution function that assigns every place either the resolution type priority or probability, and $\wp\colon (F \cup G) \rightarrow \{\{1,2,\dots,\max(n_{in}, n_{out})\}\colon e(p_i) = prio,\ [0,1]\colon e(p_i) = prob\}$ is an enabling function which assigns every arc from or to a place $p_i \in P$ either an enabling priority or a probability according to the resolution type of the place and with the condition that each priority from 1 to n_{out} for the output transitions and each priority from 1 to n_{in} for the input transitions is only used once for each place p_i with $e(p_i) = prio$

$$\wp(p_i \rightarrow t_j) \neq \wp(p_i \rightarrow t_k), \qquad \forall t_j, t_k \in T_{out}(p_i)$$

$$\wp(t_j \rightarrow p_i) \neq \wp(t_k \rightarrow p_i), \qquad \forall t_j, t_k \in T_{in}(p_i)$$

and that 1 is the best priority and n_{out} and n_{in}, respectively, is the worst priority and that the sum of probabilities for the output and input transitions, respectively, is equal to one for each place p_i with $e(p_i) = prob$

$$\sum_{t_j \in T_{out}(p_i)} \wp(p_i \rightarrow t_j) = 1\ ,\ \sum_{t_j \in T_{in}(p_i)} \wp(t_j \rightarrow p_i) = 1.$$

A transition in a resolved capacitive Petri net is firable if it is enabled by all input and output places. The firing process of an enabled transition can be adopted from the basic concepts (Definition 4.13).

Definition 4.22 (firability capacitive Petri net)

The tuple $(P, T, F, G, f, c_l, c_u, e, \wp, m_0)$ is a resolved capacitive Petri net. An active transition $t_j \in T$ is **firable** if and only if

$$\forall\, p_i \in P_{in}(t_j) : t_j \in TE_{out}(p_i)$$

and

$$\forall\, p_i \in P_{out}(t_j) : t_j \in TE_{in}(p_i).$$

Example 4.5

Figure 4.9 shows a capacitive Petri net with the initial marking $m_0 = (2,3,0)$. All transitions are active

- T1: $m_0(P1) - f(P1 \rightarrow T1) = 2 - 1 \geq c_l(P1) = 1$ and $m_0(P3) + f(T1 \rightarrow P3) = 0 + 1 \leq c_u(P3) = 1$
- T2: $m_0(P2) - f(P2 \rightarrow T2) = 3 - 1 \geq c_l(P2) = 2$ and $m_0(P3) + f(T2 \rightarrow P3) = 0 + 1 \leq c_u(P3) = 1$
- T3: $m_0(P2) - f(P2 \rightarrow T3) = 3 - 1 \geq c_l(P2) = 2$ and $m_0(P3) + f(T3 \rightarrow P3) = 0 + 1 \leq c_u(P3) = 1.$

At first, the places enable as many output transitions as their minimum capacities allow. $P1$ has no general conflict and enables $T1$ and $P2$ has a general output conflict and can either enable $T2$ or $T3$.

- Case 1: $P2$ enables $T2$

  $P3$ has a general input conflict due to its maximum capacity and can either enable $T1$ or $T2$. $T3$ is not an option because it is not enabled by its input place $P2$. If $P3$ enables $T1$, $T1$ is firable and fires by removing one token from $P1$ and adding one to $P3$ and on the other hand, if it enables $T2$, $T2$ is firable and fires by removing one token from $P2$ and adding one to $P3$.
- Case 2: $P2$ enables $T3$

  $P3$ has a general input conflict due to its maximum capacity and can either enable $T1$ or $T3$. $T2$ is not an option because it is not enabled by its input place $P2$. If $P3$ enables $T1$, $T1$ is firable and fires by removing one token from $P1$ and adding one to $P3$ and on the other hand, if it enables $T3$, $T3$ is firable and fires by removing one token from $P2$ and adding one to $P3$.

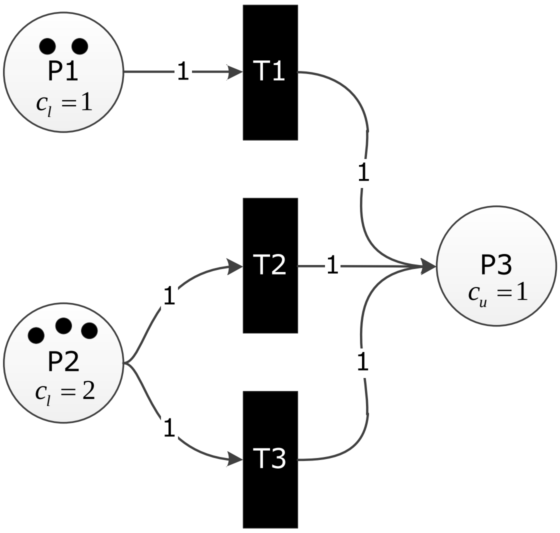


Figure 4.9: Capacitive Petri net with a general output conflict (Example 4.5)

4.2.2 Extended Petri Nets

The basic Petri net concept has been extended by specific arcs as proposed in (Matsuno et al. 2003), (David and Alla 2001), and (David and Alla 2005). In this work three specific arcs are added - **test arc**, **inhibitor arc**, and **read arc** – to accomplish the modeling of, for example, inhibition and activation mechanisms of biological reactions. These arcs connect all places with transitions.

The test arc is represented by a dashed line, the inhibitor arc has a small circle at its end, and the read arc has black square at its end (see Figure 4.10 and Figure 4.11).

If places are connected with test, inhibitor, or read arcs to transitions, their markings are not changed during the firing processes. In the case of test and inhibitor arcs the markings are only read to influence the activation process while read arcs do not influence the activation process nor the firing process. Hence, read arcs have no weights; they only indicate the usage of the token number in the connected transition, for example, for firing conditions or arc weight functions (see Section 4.3).

The same place can be connected with the same transition by a test and normal arc as well as by an inhibitor and normal arc. These arcs are called **double arcs**.

Definition 4.23 (extended Petri net, double arc)

The tuple $(P, T, F, G, \mathcal{T}, \mathcal{I}, \mathcal{R}, f, e, \wp, m_0)$ is an **extended Petri net** if $(P, T, F, G, f, e, \wp, m_0)$ is a resolved Petri net, $\mathcal{T} \subseteq (P \times T)$ is a set of test arcs, $\mathcal{I} \subseteq (P \times T)$ is a set of inhibitor arcs, $\mathcal{R} \subseteq (P \times T)$ is a set of read arcs, and the arc weight function f is modified such that $f: (F \cup G \cup \mathcal{T} \cup \mathcal{I}) \to \mathbb{N}_0$, whereby $f\left(\left(p_i \to t_j\right)_{\mathcal{T}}\right)$ is the weight of the test arc $\left(p_i \to t_j\right)_{\mathcal{T}}$ and $f\left(\left(p_i \to t_j\right)_{\mathcal{I}}\right)$ is the weight of the inhibitor arc $\left(p_i \to t_j\right)_{\mathcal{I}}$. If $\left(p_i \to t_j\right) \in F$ and $\left(p_i \to t_j\right)_{\mathcal{T}} \in \mathcal{T}$ or $\left(p_i \to t_j\right) \in F$ and $\left(p_i \to t_j\right)_{\mathcal{I}} \in \mathcal{I}$ then the arc is called **double arc**.

A transition in an extended Petri net is active if

- all input places connected by normal arcs have at least as many tokens as the arc weights,
- all input places connected by test arcs have more tokens than the arc weights, and
- all input places connected by inhibitor arcs have fewer tokens than the arc weights.

Places connected by read arcs do not influence the activation of a transition. The places connected by test, inhibitor, and read arcs enable all active output transitions because tokens are not changed during the firing process.

Definition 4.24 (activation extended Petri net)

The tuple $(P, T, F, G, \mathcal{T}, \mathcal{I}, \mathcal{R}, f, e, \wp, m_0)$ is an extended Petri net. A transition $t \in T$ is **active** with regard to a concrete marking m if and only if

$$\forall\, p_i \in P_{in}(t_j): \begin{cases} m(p) \geq f\left(p_i \to t_j\right) & if\ \left(p_i \to t_j\right)\ \in F \\ m(p) > f\left(\left(p_i \to t_j\right)_{\mathcal{T}}\right) & if\ \left(p_i \to t_j\right)_{\mathcal{T}} \in \mathcal{T} \\ m(p) < f\left(\left(p_i \to t_j\right)_{\mathcal{I}}\right) & if\ \left(p_i \to t_j\right)_{\mathcal{I}} \in \mathcal{I}. \end{cases}$$

The enabling process by priorities or probabilities, the firability definition, and the firing process are not affected by these extensions and have been adopted from the basic concepts (Definition 4.9, Definition 4.10, Definition 4.12, and Definition 4.13).

Example 4.6

The Petri nets at the top in Figure 4.10 contain test arcs and the Petri nets at the bottom inhibitor arcs. Transition $T1$ is active with regard to a concrete marking m because the token number of $P2$ is above the weight of the test arc $(m(P2) = 3 > f(P2 \to T1) = 2)$. However, $T2$ is not active because the marking of $P5$ is less than the arc weight $(m(P5) = 1 \ngtr f(P5 \to T2) = 2)$. $T3$ is also not active because the token number of $P8$ is greater than the weight of the inhibitor arc $(m(P8) = 3 \nless f(P8 \to T3) = 2)$. However, $T4$ is active because the marking of $P11$ is less than the arc weight $(m(P11) = 1 < f(P11 \to T4) = 2)$.

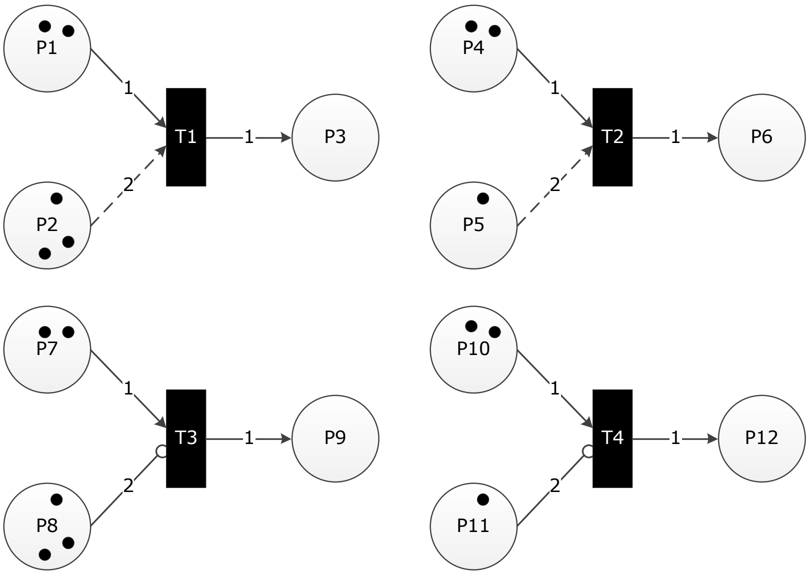


Figure 4.10: Extended Petri nets with test arcs (top) and inhibitor arcs (bottom) (Example 4.6). $T1$ and $T4$ are active and $T2$ and $T3$ are not active.

Example 4.7

Figure 4.11 shows three different biological reactions modeled by extended Petri nets. The first reaction is inhibited by the inhibitor I which is modeled by an inhibitor arc from place I to transition R. This reaction can only proceed when I is less than a specific bound which is represented by the arc weight $f((I \to R)_{\mathcal{I}})$.

The second reaction activated by the activator A which is modeled by a test arc from place A to transition R. This reaction can only proceed when A is greater than a specific bound represented by the arc weight $f((A \to R)_{\mathcal{T}})$.

The third reaction is catalyzed by the enzyme E. This enzyme is not consumed in the reaction but is needed because it influences the amount of substrate molecules which can be converted to product molecules. This is modeled by a read arc to indicate that the token amount of place E is needed for the arc weights of the transition R (these are functional arc weights see also Section 4.3, Example 4.8, and Example 4.9). This arc is only for visualization and does not influence the firing process of the transition R.

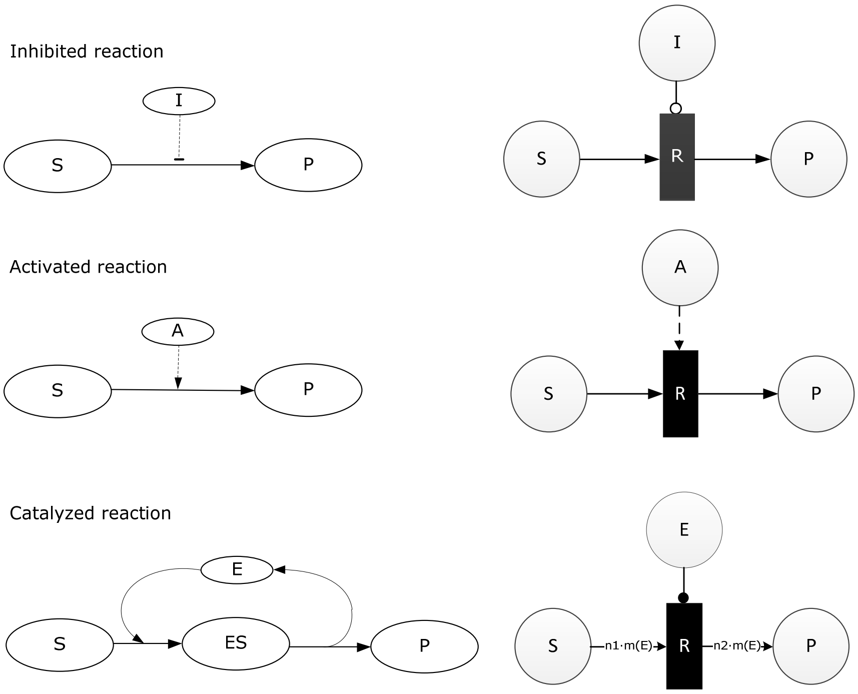


Figure 4.11: Modeling of different biological reactions with extended Petri nets; top: the reaction is inhibited by the inhibitor *I*, it can only proceed when *I* is less than a specific bound, middle: the reaction is activated by the activator *A*, it can only proceed, when *A* is greater than a specific bound, bottom: the reaction is catalyzed by the enzyme *E*, the information about the amount of *E* is needed to determine how many substrates molecules can be converted to product molecules (Example 4.7).

4.2.3 Extended Capacitive Petri Nets

The modeling of biological reactions often requires a combination of the two mentioned modifications, i.e. capacitive Petri nets which are extended by test, inhibitor, and read arcs.

Definition 4.25 (extended capacitive Petri net)

The tuple $(P, T, F, G, \mathcal{T}, \mathcal{I}, \mathcal{R}, f, c_l, c_u, e, \wp, m_0)$ is called **extended capacitive Petri net** if

– $(P, T, F, G, f, c_l, c_u, e, \wp, m_0)$ is a resolved capacitive Petri net,

– $\mathcal{T} \subseteq (P \times T)$ is a set of test arcs,

– $\mathcal{I} \subseteq (P \times T)$ is a set of inhibitor arcs,

– $\mathcal{R} \subseteq (P \times T)$ is a set of read arcs, and

– $f: (F \cup G \cup \mathcal{T} \cup \mathcal{I}) \to \mathbb{N}_0$ is a modified arc weight function.

A transition in an extended capacitive Petri net is active if the token numbers of all input places do not fall below the minimum capacities when the arc weights are removed, and the maximum capacities of all output places may not be exceeded when the arc weights are added. Additionally, the input places connected by test arcs must have more tokens than the arc weights and the places connected by inhibitor arcs must have less tokens than the arc weights; read arc do not influence the activation of a transition.

Definition 4.26 (activation extended capacitive Petri net)

The tuple $(P, T, F, G, \mathcal{T}, \mathcal{I}, \mathcal{R}, f, c_l, c_u, e, \wp, m_0)$ is an extended capacitive Petri net. A transition $t \in T$ is **active** with regard to a concrete marking m if and only if

$$\forall\, p_i \in P_{in}(t_j) : \begin{cases} m(p_i) - f(p_i \to t_j) \geq c_l(p_i) & if\ (p_i \to t_j)\ \in F \\ m(p_i) > f\left((p_i \to t_j)_{\mathcal{T}}\right) & if\ (p_i \to t_j)_{\mathcal{T}} \in \mathcal{T} \\ m(p_i) < f\left((p_i \to t_j)_{\mathcal{I}}\right) & if\ (p_i \to t_j)_{\mathcal{I}} \in \mathcal{I}, \end{cases}$$

and

$$\forall\, p_i \in P_{out}(t_j) :\ m(p_i) + f(p_i \to t_j) \leq c_u(p_i).$$

The enabling can be performed by priorities, probabilities, or by a mix of both. The respective definitions for capacitive Petri net can be adopted (Definition 4.19 and Definition 4.20) as well as the firability definition (Definition 4.22) and the basic firing process (Definition 4.13).

4.2.4 Self-modified Petri Nets

The arc weight function has been modified in order to model dynamic arc weights which depend on the marking of a place, i.e. not only positive integers can be written at the arcs but also the place markings. This Petri net extension is called a self-modified Petri net and was first introduced by Valk in (Valk 1978). Self-modified Petri nets enables, for example, the modeling of biochemical reactions on the molecular level as demonstrated in Example 4.8.

The corresponding definition valid for this work is given below.

Definition 4.27 (self-modified Petri net)

The tuple $(P, T, F, G, f, e, \wp, m_0)$ is called **self-modified Petri net** if $(P, T, F, G, f, e, \wp, m_0)$ is a resolved Petri net and the arc weight function f is modified such that $f\colon (F \cup G, m) \to \mathbb{N}_0$ is an dynamic arc weight function that assigns every arc either a positive integer or a concrete marking $m(p_i)$ of the place p_i.

Example 4.8

Figure 4.12 displays a biochemical reaction; a substrate is converted into a product with the aid of an enzyme. The enzyme is necessary for the reaction but not consumed. After the product formation the enzyme is set free and can convert another substrate molecule into a product. It is assumed that there are more substrate molecules then enzymes.

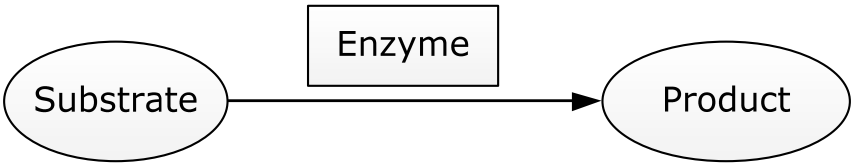


Figure 4.12: Biochemical reaction (Example 4.8)

Figure 4.13 shows the biochemical reaction modeled by a self-modified Petri net; substrate, product, and enzyme are modeled by the places S, P, and E, respectively and the biochemical reaction is modeled by the transition R. The marking of the enzyme place E is written at all arcs. This causes the removing of $m(E)$ tokens from the places S and E and the addition of $m(E)$ tokens to place P and E at each firing of transition R. The transition R is active if E has tokens and S has at least as many tokens as E.

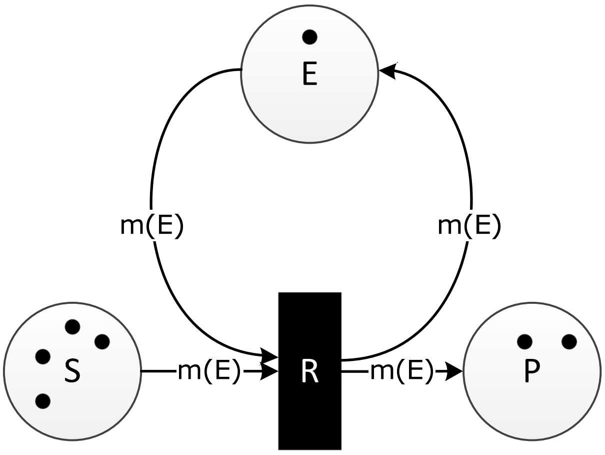


Figure 4.13: Self-modified Petri net (Example 4.8)

The activation definition of a transition in a self-modified Petri net have been adopted from the basic concepts (Definition 4.5) just as well as the enabling process, the firability definition, and the firing process (Definition 4.9, Definition 4.10, Definition 4.12, and Definition 4.13). Self-modified Petri nets can also have capacities or be extended by test, inhibitor, and read arcs or both. Then the respective definitions of the previous sections are valid without exception.

4.2.5 Functional Petri Nets

The self-modified Petri nets have been further extended to functional Petri nets which were firstly introduced by Hofestädt and Thelen in (Hofestädt and Thelen 1998). The arc weights are functions which can depend on the markings of several places. This modification further expands the application of the Petri net formalism in the biological field. The Petri net model of the biochemical reaction in Figure 4.12 can be, for example, modified by integrating specific kinetic effects on the molecular level (see Example 4.9). The corresponding definitions valid for this work are given below.

Definition 4.28 (functional Petri net)

The tuple $(P,T,F,G,f,e,\wp,m_0)$ is called **functional Petri net** if $(P,T,F,\mathrm{G},f,e,\wp,m_0)$ is a resolved Petri net and the arc weight function f is modified such that $f\colon (F \cup G, m) \to \mathbb{N}_0$ is a dynamic arc weight function which assigns every arc a function that depends on a subset of concrete markings m.

The activation definition has been adopted from the basic concepts (Definition 4.5) just as well as the enabling process, the firability definition, and firing process (Definition 4.9, Definition 4.10, Definition 4.12, and Definition 4.13). Functional Petri nets can also have capacities or be extended by test, inhibitor, and read arcs or both. Then the respective definitions of the previous sections are valid without exception.

Example 4.9

In comparison to Example 4.8, the biochemical reaction can be described by linear functions which depend on the available number of enzyme molecules. Figure 4.14 shows the functional Petri net of the biochemical reaction taken from (Hofestädt and Thelen 1998); thereby, the parameter n_1 and n_2 are arbitrary positive integers. At each firing $n_1 \cdot m(E)$ tokens are removed from place S and $m(E)$ from E; additionally, $n_2 \cdot m(E)$ tokens are added to place P and $m(E)$ to E. The transition is active if E has tokens and S has at least $n_1 \cdot m(E)$ tokens.

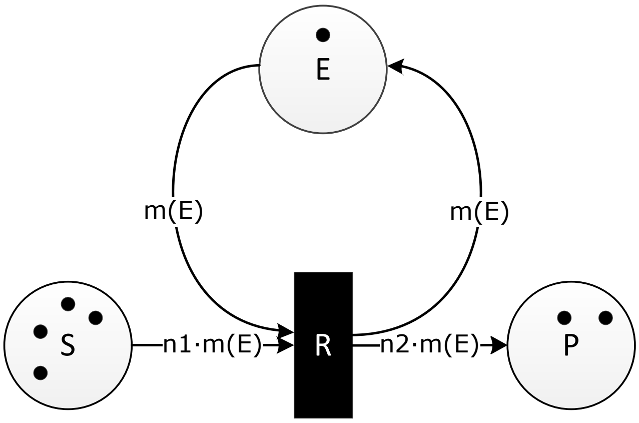


Figure 4.14: Functional Petri net (Example 4.9)

4.3 NON-AUTONOMOUS PETRI NETS

The Petri nets of the previous sections describe systems and processes in a qualitative manner. The firing instants of the transitions are either unknown or not indicated. This kind of Petri net is called **autonomous Petri net**. However, if a Petri net should be analyzed quantitatively, for example, by a simulation, it is necessary to associate events and/or time with its behavior; it is then called **non-autonomous Petri net**. Two different concepts are discussed. The first implies that enabled transitions fire when an associated condition is satisfied (conditional Petri net) and the second involve the firing of enabled transitions when an associated time period is passed (timed Petri net). Both concepts can also be mixed in one Petri net.

4.3.1 CONDITIONAL PETRI NETS

In a conditional Petri net, an event is associated with every transition. An enabled transition fires when the associated condition is fulfilled.

By means of conditional Petri nets, it is, for example, possible to model signal transduction. Thereby, external stimuli are the conditions associated with the transitions. These transitions represent the processes by which the cell reacts to external stimuli.

Definition 4.29 (conditional Petri net)

The tuple $(P,T,F,G,f,e,\wp,s,m_0)$ is a **conditional Petri net** if $(P,T,F,G,f,e,\wp,m_0)$ is a resolved Petri net and $\mathrm{s}:(\mathrm{T},\mathcal{E}) \rightarrow \{\mathrm{true},\mathrm{false}\}$ is a condition function that assigns every transition $t_j \in T$ a condition $s_j = s(t_j, \mathcal{E})$ depending on several environmental factors $\mathcal{E}$ e.g. time.

Definition 4.30 (firing instant, actual firable conditional Petri net)

The tuple $(P, T, F, G, f, e, \wp, \mathrm{s}, m_0)$ is a conditional Petri net. A **firing instant** of a firable transition $t_j \in T$ occurs when the associated condition s_j becomes true. It is said that the transition is **actual firable** and it then fires immediately.

The basic definitions for activation, firability, and firing process keep their validity (Definition 4.5, Definition 4.12, and Definition 4.13). Conditional Petri net can be capacitated, extended, capacitated and extended, self-modified, or functional. Then the respective definitions of Section 4.2 are valid.

Example 4.10

Figure 4.15 shows a conditional Petri net; transition $T1$ is active and firable. It fires when the assigned condition $C1$ becomes true which is represented on the right side of Figure 4.15. At that time, one token is removed from $P1$ and one is added to $P2$. Then $T2$ is firable and fires when the condition $C2$ becomes true by removing one token from $P2$ and adding one to $P1$.

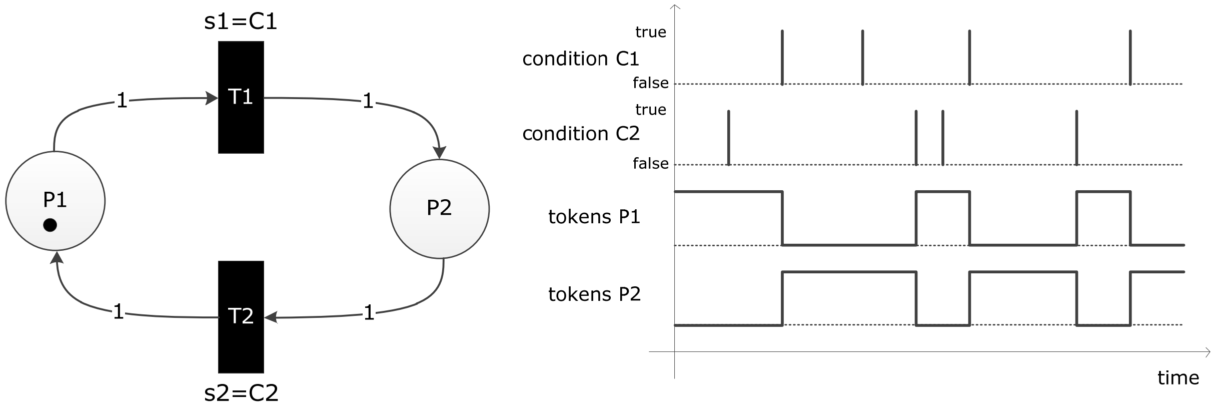


Figure 4.15: Conditional Petri net (Example 4.10)

The enabling of conditional transitions requires the consideration of general conflicts but only those which happen at the same time have to be resolved, called actual conflicts.

Definition 4.31 (actual conflict)

The tuple $(P, T, F, G, f, e, \wp, s, m_0)$ is a conditional Petri net. If a place $p_i \in P$ has a general conflict according to Definition 4.6 and the involved transitions are actual firable (Definition 4.30) at the same time, then the place has an **actual conflict**.

Actual conflicts can be resolved by providing the transitions with priorities or probabilities; thereby, the basic enabling processes can be adopted (Definition 4.9 and Definition 4.10) and the corresponding ones for the Petri net modifications of Section 4.2, respectively.

Example 4.11

Figure 4.16 represents a conditional Petri net. Place $P5$ has a general conflict because it has not enough tokens to enable $T1$ and $T2$, simultaneously. It is also an actual conflict because both transitions are provided with the same condition $C1$.

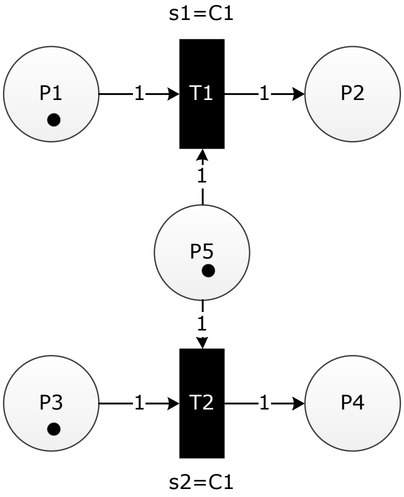


Figure 4.16: Conditional Petri net with an actual conflict (Example 4.11)

4.3.2 TIMED PETRI NETS

In timed Petri nets, a delay is associated with every transition. An enabled transition fires first when the associated delay is passed.

These delays represent the duration of a biological reaction associated with the transition. If, for example, a biochemical reaction is modeled by a transition, then the delay represents the time that is necessary to convert a determined quantity (input arc weights) of substrates (input places) to a determined quantity (output arc weights) of products (output places) (see Example 4.13).

Definition 4.32 (timed Petri net)

The tuple $(P, T, F, G, f, e, \wp, d, m_0)$ is a **timed Petri net** if $(P, T, F, G, f, e, \wp, m_0)$ is a resolved Petri net and $d: T \rightarrow \mathbb{R}_{\geq 0}$ is a delay function that assigns every transition $t_j \in T$ a non-negative real number, whereby $d_j = d(t_j)$ is the delay of transition $t_j \in T$.

Definition 4.33 (firing instant, actual firable timed Petri net)

The tuple $(P, T, F, G, f, e, \wp, d, m_0)$ is a timed Petri net. A **firing instant** of a firable transition $t_j \in T$ occurs when the associated time period d_j is elapsed. It is said that the transition is **actual firable** and it then fires immediately.

The basic definitions for activation, firability, and firing process keep their validity (Definition 4.5, Definition 4.12, and Definition 4.13). Timed Petri nets can be capacitated, extended, capacitated and extended, self-modified, or functional. Then the respective definitions of Section 4.2 are valid.

Example 4.12

Figure 4.17 shows on the left side a timed Petri net and on the right side the corresponding token evolution:

- Time 0: $T1$ and $T3$ become firable.
- Time 2: the delay of $T3$ is elapsed and it fires by removing two tokens from $P4$ and one from $P3$ and by adding two to $P2$; $T2$ becomes firable.
- Time 3: the delay of $T1$ is elapsed and it fires by removing one token from $P1$ and by adding one to $P2$; $T1$ becomes firable again.
- Time 6: the delay of $T1$ is elapsed and it fires by removing one token from $P1$ and by adding one to $P2$; Simultaneously, the delay of $T2$ is elapsed and it fires by removing two tokens from $P2$ and by adding one to $P1$; $T1$ and $T2$ become firable again.

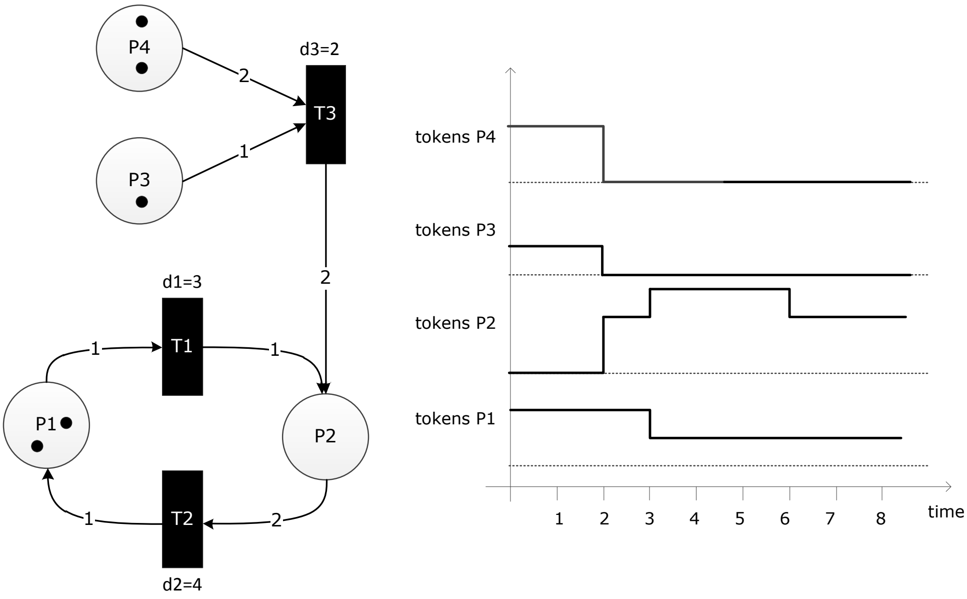


Figure 4.17: Timed Petri net (left) and the token evolution (right) (Example 4.12)

Example 4.13

Figure 4.18 shows a timed Petri net of the biochemical reaction in Figure 4.12. In dR time units, $(n1 \cdot enzyme\, molecules)$ substrate molecules should be converted into $(n2 \cdot enzyme\, molecules)$ product molecules. This is achieved by assigning transition R in Figure 4.14 the delay dR. This causes that the transition R waits dR time units after it becomes firable before $n1 \cdot m(E)$ tokens are removed from S and $n2 \cdot m(E)$ are added to P.

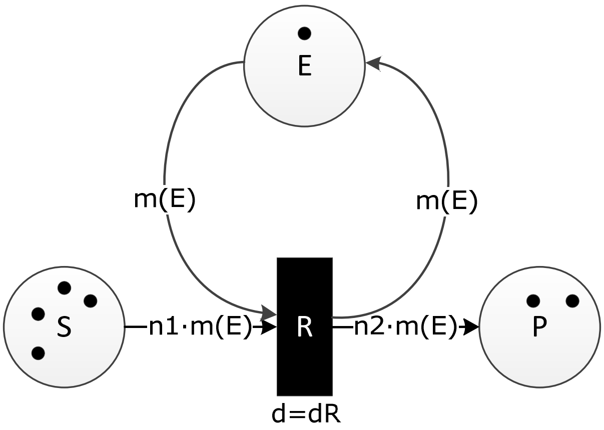


Figure 4.18: A timed Petri net of the biochemical reaction in Figure 4.12 (Example 4.13)

Remark 4.3

If a place in a timed Petri net has a general conflict, at first the involved transition is fired which becomes actual firable according to Definition 4.33. An actual conflict only occurs when two or more involved transitions become actual firable at the same time. Then the conflict can be solved by priorities or probabilities; thereby, the basic enabling processes can be adopted (Definition 4.9 and Definition 4.10) or the respective enabling processes for the Petri net modifications of Section 4.2.

Example 4.14

Figure 4.19 represents two timed Petri nets. Places $P5$ and $P10$ have a general conflict because they have not enough tokens to enable both connected transitions. The conflict of $P5$ is not actual due to the fact that $T1$ becomes firable before $T2$. Thus $T1$ fires and afterwards $T2$ is not active anymore. However, the conflict of $P10$ is actual because the same delay is assigned to both transitions. This actual conflict can be resolved by a deterministic or random process to decide which of the involved transitions, $T3$ or $T4$, gets permission to fire.

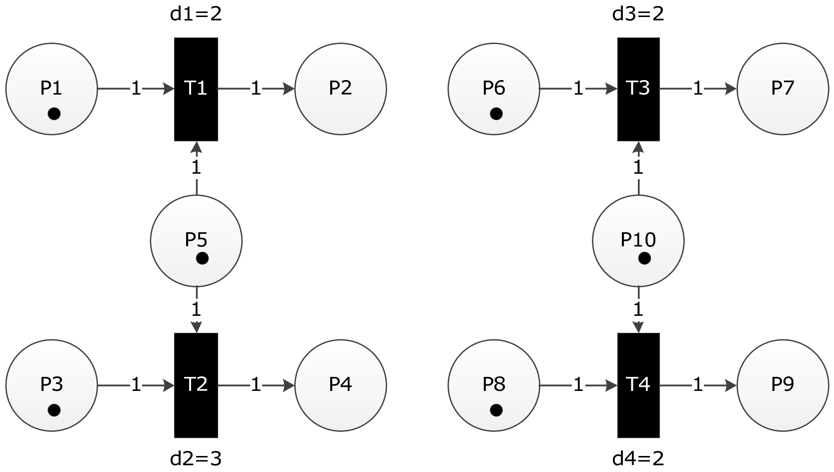


Figure 4.19: Timed Petri nets without actual conflict (left) and with actual conflict (right) (Example 4.14)

4.3.3 Stochastic Petri Nets

The timed Petri net concept can be modified to stochastic Petri nets which were introduced by Goss and Peccoud in (Goss and Peccoud 1998) for quantitative modeling of molecular reaction networks. The stochastic transitions are provided with random delays instead of fixed values and are graphically represented by a black box with a white triangle (see Figure 4.20). The delay is an exponentially distributed random variable $\mathcal{X} = Exp(\lambda)$ with the probability density function

$$f_\lambda(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases},$$

the distribution function

$$F_\lambda(x) = \begin{cases} 1 - e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases},$$

and expectation value

$$E(\mathcal{X}) = \frac{1}{\lambda},$$

whereby $\lambda > 0$ is the **characteristic parameter** of the exponential distribution. Thereby, the exponential distribution is chosen because this distribution covers only real numbers and is memoryless.

The characteristic parameter λ can depend functionally on the markings of several places (Heiner et al. 2008) and is recalculated at each point in time when the respective transition becomes active or when one or more markings of involved places change. The definitions developed within this work are given below.

Definition 4.34 (stochastic Petri net)

The tuple $(P, T, F, \mathrm{G}, f, e, \wp, h, m_0)$ is a **stochastic Petri net** if $(P, T, F, \mathrm{G}, f, e, \wp, m_0)$ is a resolved Petri net and $h: (T, m) \rightarrow \mathbb{R}_{\geq 0}$ is a hazard function which assigns every transition $t_j \in T$ a function $h_j = h(t_j, m)$ that depends on a subset of concrete markings m. Each time when t_j becomes active or involved markings m are changed by firing other transitions, the characteristic parameter $\lambda_j = h_j$ is recalculated to evaluate the next putative firing point in time $\tau_j = time + \mathrm{Exp}(\lambda_j)$.

Definition 4.35 (firing instant, actual firable stochastic Petri net)

The tuple $(P, T, F, G, f, e, \wp, h, m_0)$ is a stochastic Petri net. A **firing instant** of a firable transition $t_j \in T$ occurs when the putative firing time τ_j is reached. It is said that the transition is **actual firable** and it then fires immediately.

The basic definitions for activation, firability, and the firing process keep their validity (Definition 4.5, Definition 4.12, and Definition 4.13). Stochastic Petri nets can be capacitated, extended, capacitated and extended, self-modified, or functional. Then the respective definitions of the Section 4.2 are valid.

Remark 4.4

If a place in a stochastic Petri net has a general conflict, the first transition which becomes actual firable according to Definition 4.35 is fired. Hence, there are no actual conflicts in stochastic Petri nets. But if pseudo random numbers are used, actual conflicts are conceivable.

Example 4.15

Figure 4.20 shows on the left side a stochastic Petri net; transition $T1$ has the fixed delay $d1 = 1$ and $T2$ and $T3$ are stochastic transitions with random putative firing times which are generated according to their hazard functions.

- Time 0: $T1$ becomes firable.
- Time 1: the delay of $T1$ is elapsed and it fires by adding one token to $P1$; $T2$ and $T3$ become firable and evaluate each a putative firing time according to the values of the hazard functions at this time $\lambda 2 = h2 = 0.8$ and $\lambda 3 = h3 = 1.8$. The transition with the smaller putative firing time becomes actual firable and fires. Afterwards, the transitions are not active anymore. They become active again when $T1$ fires and $P1$ gets one token. Then the next token competition between $T2$ and $T3$ occurs.

Figure 4.20 shows on the right side one possible token evaluation of $P2$ and $P3$.

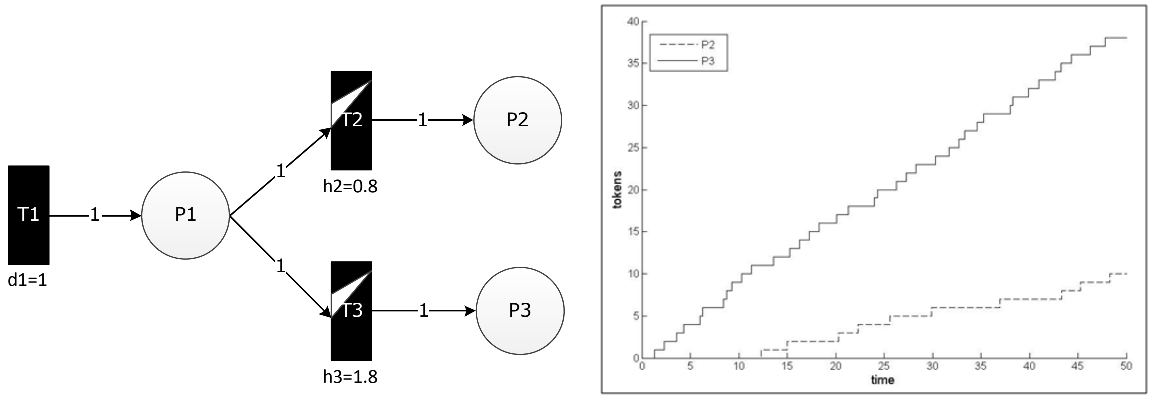


Figure 4.20: Stochastic Petri net (left) and one possible token evolution (right) (Example 4.15)

<u>Example 4.16</u>

Figure 4.21 shows a stochastic Petri net model of protein synthesis taken from (Goss and Peccoud 1998). The place IG represents the inactive gene, AG represents the active gene, and P represents the protein. The stochastic transitions model the processes activation ($T1$), inactivation ($T2$), synthesis ($T3$), and degradation ($T4$). The arc $(AG \to T3)_{\mathcal{T}}$ is modeled by a test arc because the gene is needed for the synthesis but not consumed by it. These processes are modeled by stochastic transitions due to the fact that their durations are not exactly known. How long is the gene active? When does it become active again? Therefore, the parameters $\alpha, \beta, \gamma,$ and δ characterize the exponentially distributed random number generated for the next putative firing time.

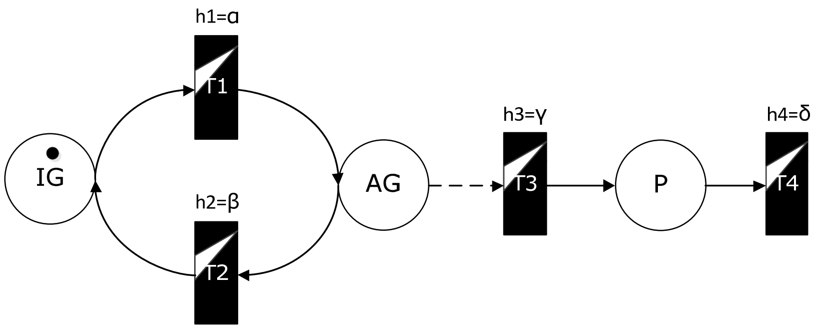


Figure 4.21: A Stochastic Petri net model of protein synthesis (IG = inactive gene, AG = active gene, P = protein, T1 = activation, T2 = inactivation, T3 = synthesis, and T4 = degradation) (Example 4.16)

4.3.4 CONDITIONAL TIMED PETRI NETS

The concepts of conditional and timed Petri nets are combined in one Petri net, called conditional timed Petri net. The transitions can have delays and additional conditions that have to be true so that the transitions become active. If the delays of the transitions are all zero, it is a conditional Petri net (see Definition 4.29) and if there are no additional conditions assigned to the transitions, it is a timed Petri net (see Definition 4.32).

Definition 4.36 (conditional timed Petri net)

The tuple $(P, T, F, G, f, e, \wp, d, s, m_0)$ is a **conditional timed Petri net** if $(P, T, F, G, f, e, \wp, d, m_0)$ is a timed Petri net and $s: (T, \mathcal{E}) \rightarrow \{true, false\}$ is a condition function that assigns every transition $t_j \in T$ a condition $s_j = s(t_j, \mathcal{E})$ depending on several environmental factors $\mathcal{E}$ e.g. time.

Definition 4.37 (activation conditional timed Petri net)

The tuple $(P, T, F, G, f, e, \wp, d, s, m_0)$ is a conditional timed Petri net. A transition $t_j \in T$ is **active** with regard to a concrete marking m if and only if the conditions of Definition 4.5 are fulfilled and, additionally, the condition s_j.

The basic definitions for the firability and the firing process keep their validity (Definition 4.12 and Definition 4.13) as well as the firing instant definition of timed Petri nets (Definition 4.33). Conditional timed Petri nets can be capacitated, extended, capacitated and extended, self-modified, or functional. Then the respective definitions of Section 4.2 are valid.

Remark 4.5

Transitions in a stochastic Petri net can also have additional conditions.

4.4 Continuous Petri Nets

The continuous Petri net concept was introduced by David and Alla in 1987 (David and Alla 1987). Contrary to the previously mentioned Petri net concepts, the token numbers and arc weights of continuous Petri nets are non-negative real numbers. In this juncture, tokens are no longer termed with tokens but rather with **marks** because the term token is commonly used for integer quantities. The firing process takes place as a continuous flow with a maximum speed assigned to every transition. Only continuous Petri nets are considered by which time is associated with the behavior; hereafter the term continuous Petri net is simplified used for timed continuous Petri nets.

The transformation from the discrete Petri net concept to is the continuous one is important for biological applications. Then it is, for example, possible to model the quantities of metabolites as real concentrations instead of integer molecules. The biochemical reaction proceeds then as a continuous flow at an assigned speed.
The definitions for continuous Petri nets developed within this work are given below.

Definition 4.38 (continuous Petri net)
The tuple (P, T, F, G, f, v, m_0) is a **continuous Petri net** if (P, T, F, G) is a net, $f: (F \cup G) \to \mathbb{R}_{\geq 0}$ is an arc weight function which assigns every arc a non-negative real number, $v: T \to \mathbb{R}_{\geq 0}$ is a maximum speed function which assigns every transition $t_j \in T$ a maximum speed $v_j = v(t_j)$, and the map $m_0: P \to \mathbb{R}_{\geq 0}$ is the initial marking.

Continuous places are represented by double circles and continuous transitions by white boxes. The transformation from a (discrete) timed Petri net to a continuous Petri net is displayed in Figure 4.22. Thereby, the discrete firing process with the delay $d_1 = 2$ is replaced by a continuous firing process with the maximum speed $v_1 = 1/d_1 = 1/2$, i.e. the discrete steps of the discrete token evolution are approximated by a line with the slope $1/2$.

Remark 4.6
Hereafter the maximum speed of a transition $t_j \in T$ is denoted by v_j, the instantaneous speed by $\tilde{v}_j$, and the preliminary speed by $\bar{v}_j$.

Definition 4.39 (input and output speed)
The tuple (P, T, F, G, f, v, m_0) is a continuous Petri net. The **input speed** of a place $p_i \in P$ is

$$I_i = \sum_{t_j \in TF_{in}(p_i)} f(t_j \to p_i) \cdot \tilde{v}_j.$$

If $I_i > 0$, place p_i is said to be **fed**.
The **output speed** of a place $p_i \in P$ is

$$O_i = \sum_{t_j \in TF_{out}(p_i)} f(p_i \to t_j) \cdot \tilde{v}_j.$$

If $O_i > 0$, place p_i is said to be **emptied**.

A continuous transition is active, enabled, and firable, simultaneously. Hereafter the term active will be used. It is active if all input places have either a marking greater than zero or are fed by at least one input transition, i.e. the input speed is not zero. If all input places have a non-zero marking the transition is said to be strongly active; otherwise, it is weakly active.

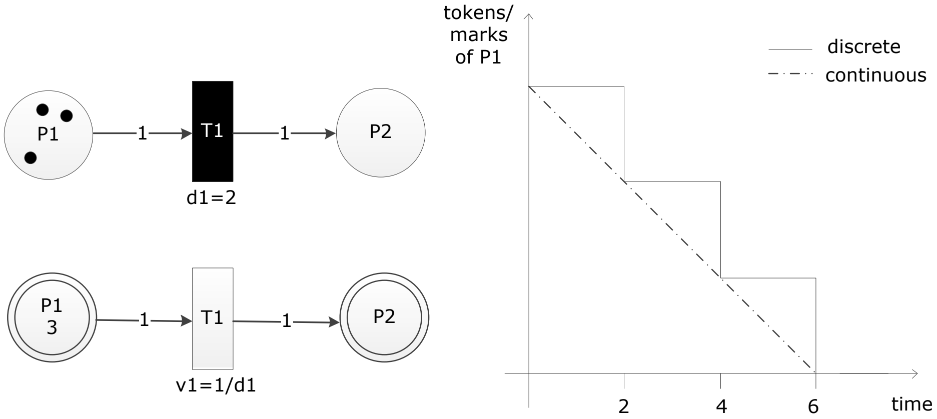


Figure 4.22: Connection between (discrete) timed Petri nets and continuous Petri nets

Definition 4.40 (activation continuous Petri net)

The tuple (P, T, F, G, f, v, m_0) is a continuous Petri net. A transition $t_i \in T$ is **active** if and only if

$$\forall\, p_i \in P_{in}(t_j) : \; m(p_i) > 0 \vee (m(p_i) = 0 \wedge I_i > 0).$$

It is **strongly active** if

$$\forall\, p_i \in P_{in}(t_j) : \; m(p_i) > 0$$

is also satisfied and **weakly active** otherwise, wherby $PI_{in}(t_j)$ ist the set of input places with $m(p_i) = 0 \wedge I_i > 0$.

A strongly active transition fires with the assigned maximum speed. However, the speed of a weakly active transition has to be decreased according to the input speeds of the input places with zero markings.

Definition 4.41 (firing process continuous Petri net)

The tuple (P, T, F, G, f, v, m_0) is a continuous Petri net. A strongly active transition $t_j \in T$ fires with the maximum speed $\tilde{v}_j = v_j$. A weakly active transition $t_j \in T$, not involved in an actual conflict according to Definition 4.42, fires with the **instantaneous speed**

$$\tilde{v}_j = \min\left(\min_{p_i \in PI_{in}(t_j)} \left(\frac{1}{f(p_i \to t_j)} \sum_{t_k \in TF_{in}(p_i)} f(t_k \to p_i) \cdot \tilde{v}_k \right), v_j \right)$$

$$= \min\left(\min_{p_i \in PI_{in}(t_j)} \left(\frac{1}{f(p_i \to t_j)} \cdot I_i \right), v_j \right).$$

The **firing process** is described by a negative mark change of all input places expressed by the following differential equation

$$\frac{dm(p_i)}{dt} = -f\left(p_i \to t_j\right) \cdot \tilde{v}_j \qquad \forall\, p_i \in P_{in}\left(t_j\right)$$

and a positive mark change of all output places expressed by the following differential equation

$$\frac{dm(p_i)}{dt} = f\left(t_j \to p_i\right) \cdot \tilde{v}_j \qquad \forall\, p_i \in P_{out}\left(t_j\right).$$

The **mark change** of the place $p_i \in P$ can be calculated by the following differential equation

$$\frac{dm(p_i)}{dt} = B_i = I_i - O_i,$$

where B_i is the **balance** of place p_i, i.e. the difference between input and output speed.

Example 4.17

Figure 4.23 shows a continuous Petri net without actual conflicts. At time 0, transition $T1$ becomes strongly active and fires with the maximum speed $\tilde{v}_1 = v_1 = 3$. Immediately afterwards, $T2$ becomes strongly active and fires with the maximum speed $\tilde{v}_2 = v_2 = 2$.

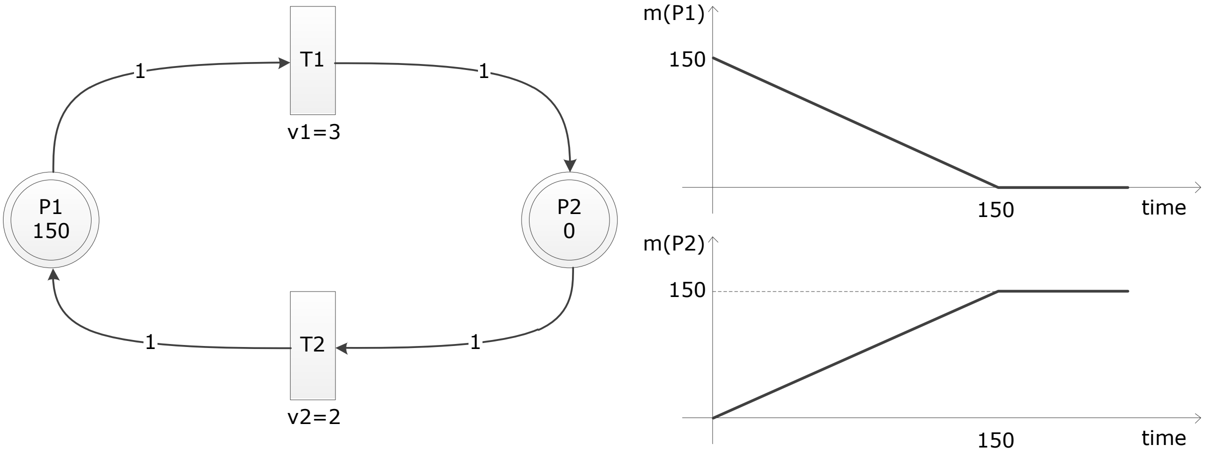


Figure 4.23: Continuous Petri net without actual conflicts (left) and the corresponding mark evolution (right) (Example 4.17

The corresponding mark changes are

$$\frac{dm(P1)}{dt} = 2 - 3 = -1$$

$$\frac{dm(P2)}{dt} = 3 - 2 = 1.$$

At time 150, $P1$ becomes empty $(m(P1) = 0)$ but it is fed $(I_1 = 2 > 0)$, hence, $T1$ is weakly active with the instantaneous speed $\tilde{v}_1 = 2$ so that $I_1 = O_1$. The corresponding mark changes are then both zero

$$\frac{dm(P1)}{dt} = 2 - 2 = 0$$

$$\frac{dm(P2)}{dt} = 2 - 2 = 0.$$

A place in a continuous Petri net has an actual conflict when the input speed is not sufficient for firing all output transitions with the instantaneous speed of Definition 4.41, called **preliminary speed** hereafter.

Definition 4.42 (actual conflict, preliminary speed continuous Petri net)

The tuple (P, T, F, G, f, v, m_0) is a continuous Petri net. A place $p_i \in P$ has an **actual conflict** if

$$m(p_i) = 0$$

and

$$I_i < \sum_{t_j \in TF_{out}(p_i)} f(p_i \to t_j) \cdot \bar{v}_j,$$

whereby

$$\bar{v}_j = \min\left(\min_{p_i \in PI_{in}(t_j)}\left(\frac{1}{f(p_i \to t_j)} \sum_{t_k \in TF_{in}(p_i)} f(t_k \to p_i) \cdot \tilde{v}_k\right), v_j\right)$$

$$= \min\left(\min_{p_i \in PI_{in}(t_j)}\left(\frac{1}{f(p_i \to t_j)} \cdot I_i\right), v_j\right)$$

is said to be the **preliminary speed** of a transition $t_j \in T$.

This conflict can be resolved by priority (Hanzalek 2003, David and Alla 2005) or sharing (David and Alla 2005) under the conditions that the balance of the place is positive and the preliminary speeds of the involved transitions are not exceeded.

Definition 4.43 (feasible solution continuous Petri net)

A solution of an actual conflict of a place $p_i \in P$ which satisfies the following inequality constraints

$$B_i = I_i - O_i \geq 0$$

$$\forall\, t_j \in TF_{out}(p_i) : \tilde{v}_j \leq \bar{v}_j$$

is said to be **feasible;** otherwise, it is **infeasible**.

The set of all feasible solutions is geometrically a convex polyhedron, thus, it usually comprises an infinite number of solutions. One feasible solution can be obtained by sharing the speed proportional to the maximum speeds of the involved transitions as suggested in (David and Alla 2001) which guarantees that the solution lies on a vertex of the polyhedron. This approach has been adapted to the continuous Petri net definitions mentioned before which leads to following definition for the firing process of continuous transitions.

Definition 4.44 (sharing proportional to maximum speeds continuous Petri net)

The tuple (P, T, F, G, f, v, m_0) is a continuous Petri net. An active transition $t_j \in T$, not involved in an actual conflict, fires with the speed of Definition 4.41. If a place $p_i \in P$ has an actual conflict according to Definition 4.42, the speed of the output transitions has to be adapted such that the constraints of Definition 4.43 are satisfied. This is done by **sharing proportional to the maximum speeds** of the involved output transitions. The instantaneous speed of an active transition $t_j \in T$ which has at least one input place with an actual conflict is then given by

$$\tilde{v}_j = \min_{p_i \in P_{in}(t_j)} (D_i) \cdot v_j.$$

Thereby, the expression

$$D_i = \frac{I_i}{\sum_{t_k \in TF_{out}(p_i)} f(p_i \to t_k) \cdot v_k}$$

is called **decreasing factor** of place p_i. This factor causes that the first condition of a feasible solution is satisfied with $B_i = 0$. The maximum speed is scaled by the minimum decrasing factor of all input places so that the condition $B_i \geq 0$ is always satisfied. But the preliminary speed could be exceeded for some transitions. Then the speeds of all transitions $t_l \in TF_{out}(p_i)$ for which $\tilde{v}_l > \bar{v}_l$ are set to the preliminary speed

$$\tilde{v}_l = \bar{v}_l.$$

The subset $\overline{TF}_{out}(p_i)$ contains all these transitions and the decreasing factors have to be modified by

$$D_i = \frac{I_i - \sum_{t_l \in \overline{TF}_{out}(p_i)} f(p_i \to t_l) \cdot \tilde{v}_l}{\sum_{t_k \in TF_{out}(p_i)} f(p_i \to t_k) \cdot v_k - \sum_{t_l \in \overline{TF}_{out}(p_i)} f(p_i \to t_l) \cdot v_l}.$$

This factor guarantees that $B_i \geq 0$ but the premilary speed could be exceeded for some transitions. Then the mentioned procedure has to be performed again.

Example 4.18

Figure 4.24 represents two continuous Petri nets which only differ in the maximum speeds v_2 and v_3 of transitions $T2$ and $T3$. The transitions $T5$ and $T6$ of the left Petri net are weakly active due to $m(P1) = m(P2) = m(P3) = m(P4) = 0$ but they are all fed, i.e. $I_1, I_2, I_3, I_4 > 0$. Hence, the preliminary speeds are given by

$$\bar{v}_5 = \min\left(\frac{1}{f(P1 \to T5)} f(T1 \to P1)v_1, \frac{1}{f(P2 \to T5)} f(T2 \to P2)v_2, \frac{1}{f(P3 \to T5)} f(T3 \to P3)v_3, v_5\right)$$

$$= \min\left(\frac{1}{1} \cdot 1 \cdot 3, \frac{1}{3} \cdot 1 \cdot 10.5, \frac{1}{2} \cdot 1 \cdot 11.7\,, 3\right) = 3$$

$$\bar{v}_6 = \min\left(\frac{1}{f(P2 \to T6)} f(T2 \to P2)v_2, \frac{1}{f(P3 \to T6)} f(T3 \to P3)v_3, \frac{1}{f(P4 \to T6)} f(T4 \to P4)v_4, v_6\right)$$

$$= \min\left(\frac{1}{1} \cdot 1 \cdot 10.5, \frac{1}{3} \cdot 1 \cdot 11.7, \frac{1}{2} \cdot 1 \cdot 1{,}2\right) = \frac{1}{2}.$$

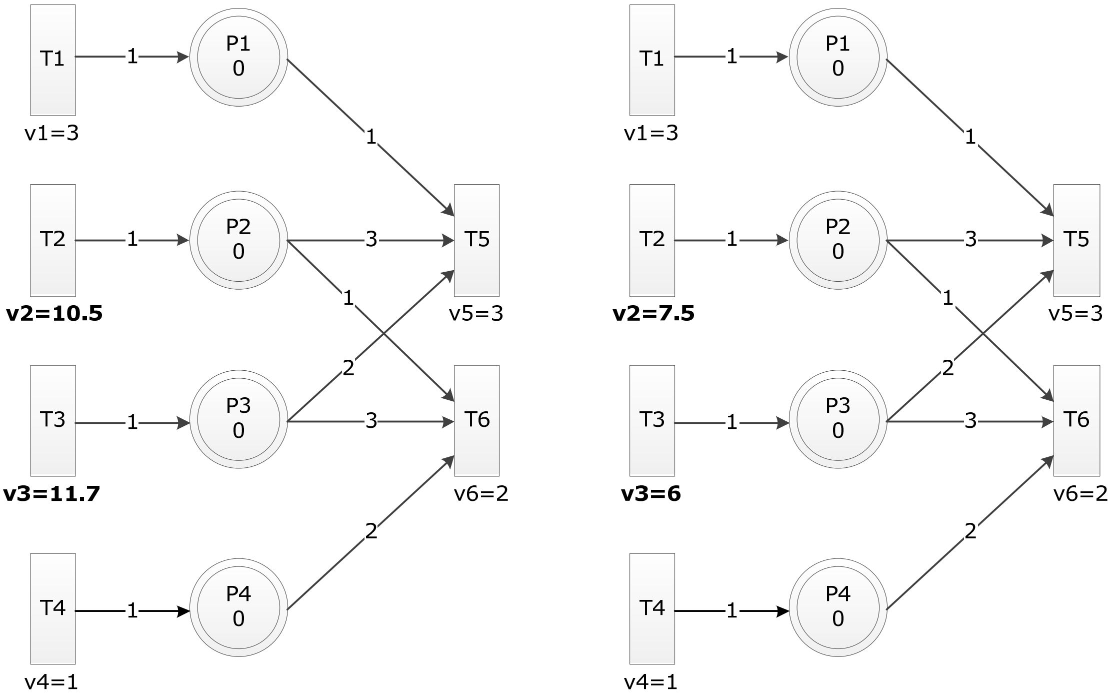


Figure 4.24: Continuous Petri nets without actual conflict (left) and with actual conflict (right) (Example 4.18)

Because neither $P2$ nor $P3$ has an actual conflict

$$I_2 = 10.5 \geq f(P2 \to T5) \cdot \bar{v}_5 + f(P2 \to T6) \cdot \bar{v}_6 = 9\frac{1}{2}$$

$$I_3 = 11.7 \geq f(P3 \to T5) \cdot \bar{v}_5 + f(P3 \to T6) \cdot \bar{v}_6 = 7\frac{1}{2},$$

the instantaneous speeds of $T5$ and $T6$ are equal to the preliminary speeds

$$\tilde{v}_5 = \bar{v}_5 = 3$$

$$\tilde{v}_6 = \bar{v}_6 = \frac{1}{2}.$$

The mark changes are

$$\frac{dm(P1)}{dt} = f(T1 \to P1) \cdot v_1 - f(P1 \to T5) \cdot \tilde{v}_5 = 0$$

$$\frac{dm(P2)}{dt} = f(T2 \to P2) \cdot v_2 - f(P2 \to T5) \cdot \tilde{v}_5 - f(P2 \to T6) \cdot \tilde{v}_6 = 1$$

$$\frac{dm(P3)}{dt} = f(T3 \to P3) \cdot v_3 - f(P3 \to T5) \cdot \tilde{v}_5 - f(P3 \to T6) \cdot \tilde{v}_6 = 4.2$$

$$\frac{dm(P4)}{dt} = f(T4 \to P4) \cdot v_4 - f(P4 \to T6) \cdot \tilde{v}_6 = 0.$$

However, the places $P2$ and $P3$ of the right Petri net have both an actual conflict

$$\bar{v}_5 = \min\left(\frac{1}{1} \cdot 1 \cdot 3, \frac{1}{3} \cdot 1 \cdot 7.5, \frac{1}{2} \cdot 1 \cdot 6\,, 3\right) = 2.5$$

$$\bar{v}_6 = \min\left(\frac{1}{1} \cdot 1 \cdot 7.5, \frac{1}{3} \cdot 1 \cdot 6, \frac{1}{2} \cdot 1 \cdot 1{,}2\right) = \frac{1}{2}$$

$$I_2 = 7.5 < f(P2 \to T5) \cdot \bar{v}_5 + f(P2 \to T6) \cdot \bar{v}_6 = 8$$

$$I_3 = 6 < f(P3 \to T5) \cdot \bar{v}_5 + f(P3 \to T6) \cdot \bar{v}_6 = 6.5.$$

Figure 4.25 shows the space of feasible solutions according to the constraints of Definition 4.43.

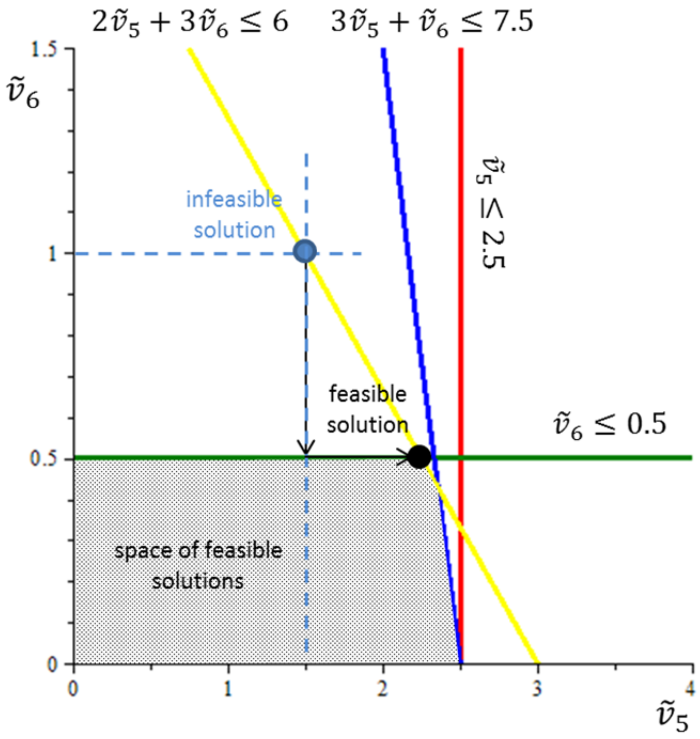


Figure 4.25: Feasible solutions of the actual conflicts of the right Petri net in Figure 4.24

One of these feasible solutions is selected by sharing which is proportional to the maximum speeds of the involved transitions $T5$ and $T6$. At first, the decreasing factors of $P2$ and $P3$ are calculated

$$D_2 = \frac{I_2}{f(P2 \to P5)v_5 + f(P2 \to T6)v_6} = \frac{7.5}{3 \cdot 3 + 1 \cdot 2} = \frac{15}{22}$$

$$D_3 = \frac{I_3}{f(P3 \to P5)v_5 + f(P3 \to T6)v_6} = \frac{6}{2 \cdot 3 + 3 \cdot 2} = \frac{1}{2}.$$

Then the maximum speeds of $T5$ and $T6$ are decreased by this factor

$$\tilde{v}_5 = \min(D_2\,, D_3)\ \cdot v_5 = \frac{1}{2} \cdot 3 = \frac{3}{2} \leq \bar{v}_5$$

$$\tilde{v}_6 = \min(D_2\,, D_3)\ \cdot v_6 = \frac{1}{2} \cdot 2 = 1 \nleq \bar{v}_6$$

But the calculated speed of $T6$ exceeds the preliminary speed marked by a blue dot in Figure 4.25. A feasible solution is achieved if the instantaneous speed is set to the preliminary speed

$$\tilde{v}_6 = \bar{v}_6 = \frac{1}{2}.$$

Additionally, the decreasing factors for the speed of $T5$ have to be recalculated

$$D_2 = \frac{I_2 - f(P2 \to T6)\tilde{v}_6}{f(P2 \to T5)v_5} = \frac{7.5 - 1 \cdot 1/2}{3 \cdot 3} = \frac{7}{9}$$

$$D_3 = \frac{I_3 - f(P3 \to T6)\tilde{v}_6}{f(P3 \to T5)v_5} = \frac{6 - 3 \cdot 1/2}{2 \cdot 3} = \frac{3}{4}$$

$$\tilde{v}_5 = \min(D_2\,, D_3)\ \cdot v_5 = 2\frac{1}{4}.$$

This solution is a vertex of the solution space and marked by a black dot in Figure 4.25. The mark changes are

$$\frac{dm(P1)}{dt} = f(T1 \to P1) \cdot v_1 - f(P1 \to T5) \cdot \tilde{v}_5 = \frac{3}{4}$$

$$\frac{dm(P2)}{dt} = f(T2 \to P2) \cdot v_2 - f(P2 \to T5) \cdot \tilde{v}_5 - f(P2 \to T6) \cdot \tilde{v}_6 = \frac{1}{4}$$

$$\frac{dm(P3)}{dt} = f(T3 \to P3) \cdot v_3 - f(P3 \to T5) \cdot \tilde{v}_5 - f(P3 \to T6) \cdot \tilde{v}_6 = 0$$

$$\frac{dm(P4)}{dt} = f(T4 \to P4) \cdot v_4 - f(P4 \to T6) \cdot \tilde{v}_6 = 0.$$

4.4.1 Continuous Capacitive Petri Nets

Every place in a continuous capacitive Petri net is provided with a lower and upper limit of marks that it can contain, similar to capacitive Petri nets (see Definition 4.14).

Definition 4.45 (continuous capacitive Petri net)

The tuple $(P, T, F, \mathrm{G}, f, v, c_l, c_u, m_0)$ is a **continuous capacitive Petri net** if (P, T, F, G, f, v, m_0) is a continuous Petri net, the map $c_l: P \to \mathbb{R}_{\geq 0}$ assigns a minimum capacity $c_l(p_i)$ to every place $p_i \in P$, the map $c_u: P \to \mathbb{R}_{\geq 0}$ assigns a maximum capacity $c_u(p_i)$ to every place $p_i \in P$, and the initial marking m_0 must satisfy the condition

$$c_l(p_i) \leq m_0(p_i) \leq c_u(p_i) \;\; \forall \, p_i \in P.$$

The continuous capacitive Petri nets require a redefinition of the activation and firing process (Definition 4.40 and Definition 4.41). A transition in a continuous capacitive Petri net is active if all input places have either a marking greater than their minimum capacities or they are fed by at least one input transition, i.e. the input speed is not zero. Additionally, all output places have either a marking less than their maximum capacities or they are emptied by at least one output transition, i.e. the output speed is not zero. If all input places have a marking greater than their minimum capacities and all output places have a marking less than their maximum capacities, the transition is said to be strongly active.

Definition 4.46 (activation continuous capacitive Petri net)

The tuple $(P, T, F, G, f, v, c_l, c_u, m_0)$ is a continuous capacitive Petri net. A transition $t_j \in T$ is **active** if and only if

$$\forall \, p_i \in P_{in}(t_j) : m(p_i) > c_l(p_i) \lor (m(p_i) = c_l(p_i) \land I_i > 0)$$

and

$$\forall \, p_i \in P_{out}(t_j) : \; m(p_i) < c_u(p_i) \lor (m(p_i) = c_u(p_i) \land O_i > 0).$$

It is **strongly input active** if

$$\forall \, p_i \in P_{in}(t_j) : m(p_i) > c_l(p_i)$$

is also satisfied and otherwise it is **weakly input active**. It is **strongly output active** if

$$\forall \, p_i \in P_{out}(t_j) : \; m(p_i) < c_u(p_i)$$

is also satisfied and otherwise it is **weakly output active**. If it is strongly input and output active, it is **strongly active**. If it is weakly input and output active, it is **weakly active**.

Thereby, $PI_{in}(t_j)$ ist the set of input places with $m(p_i) = c_l(p_i) \wedge I_i > 0$ and $PI_{out}(t_j)$ is the set of output places of t_j with $m(p_i) = c_u(p_i) \wedge O_i > 0$.

A strongly active transition fires with maximum speed. However, the speed of a weakly (input/output) active transition has to be decreased according to the input speeds of the input places with markings equal to the minimum capacities and/or according to the output speeds of the output places with markings equal to the maximum capacities.

Definition 4.47 (firing process continuous capacitive Petri net)

The tuple $(P, T, F, G, f, v, c_l, c_u, m_0)$ is a continuous capacitive Petri net. A strongly active transition $t_j \in T$ **fires** with maximum speed $\tilde{v}_j = v_j$. A weakly (input/output) active transition $t_j \in T$, not involved in an actual conflict according to Definition 4.48, fires with the **instantaneous speed**

$$\tilde{v}_j = \min\left\{\min_{p_i \in PI_{in}(t_j)}\left(\frac{1}{f(p_i \to t_j)}\sum_{t_k \in TF_{in}(p_i)} f(t_k \to p_i) \cdot \tilde{v}_k\right),\right.$$

$$\left.\min_{p_i \in PI_{out}(t_j)}\left(\frac{1}{f(t_j \to p_i)}\sum_{t_k \in TF_{out}(p_i)} f(p_i \to t_k) \cdot \tilde{v}_k\right), v_j\right\}$$

$$= \min\left\{\mathit{min}_{p_i \in PI_{in}(t_j)}\left(\frac{1}{f(p_i \to t_j)} \cdot I_i\right), \min_{p_i \in PI_{out}(t_j)}\left(\frac{1}{f(t_j \to p_i)} \cdot O_i\right), v_j\right\}.$$

The firing process of the transitions is described by a negative mark change of all input places expressed by the following differential equation

$$\frac{dm(p_i)}{dt} = -f(p_i \to t_j) \cdot \tilde{v}_j \qquad \forall\, p_i \in P_{in}(t_j)$$

and a positive mark change of all output places expressed by the differential equation

$$\frac{dm(p_i)}{dt} = f(t_j \to p_i) \cdot \tilde{v}_j \qquad \forall\, p_i \in P_{out}(t_j).$$

The **mark change** of the place $p_i \in P$ can be calculated by the following differential equation

$$\frac{dm(p_i)}{dt} = B_i = I_i - O_i,$$

where B_i is the **balance** of place p_i, i.e. the difference between input and output speed.

Example 4.19

Figure 4.26 shows a continuous capacitive Petri net without actual conflicts. Place $P2$ has a maximum capacity of 100 marks. At time 0, transition $T1$ becomes strongly active and fires with the maximum speed $\tilde{v}_1 = v_1 = 3$. Immediately afterwards, $T2$ becomes strongly active and fires with the maximum speed $\tilde{v}_2 = v_2 = 2$. At time 100, the maximum capacity of $P2$ is reached $(m(P2) = 100 = c_u(P2))$ but it is emptied by transition $T2$. Hence, $T1$ is weakly output active with the instantaneous speed $\tilde{v}_1 = 2$ so that $I_2 = O_2$. Then the corresponding mark changes are both zero.

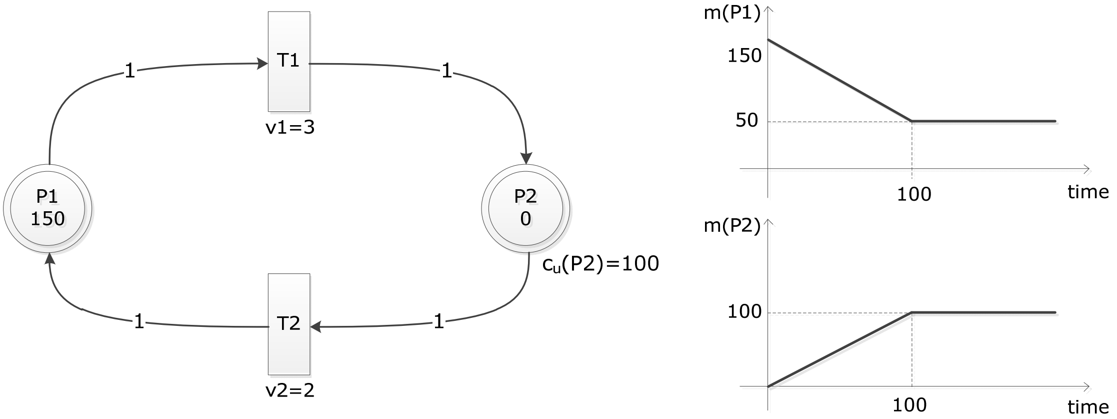


Figure 4.26: Continuous capacitive Petri net without actual conflicts (left) and the mark evolution (right) (Example 4.19)

Example 4.20

Figure 4.27 shows a continuous capacitive Petri net. Place $P1$ has a minimum capacity of 2 marks and $P2$ has a maximum capacity of 10 marks. At time 0, $T2$ is weakly active because the marking of the input place $P1$ is at the minimum capacity and, additionally, the marking of its output place is at the maximum capacity but both places are fed and emptied, respectively.

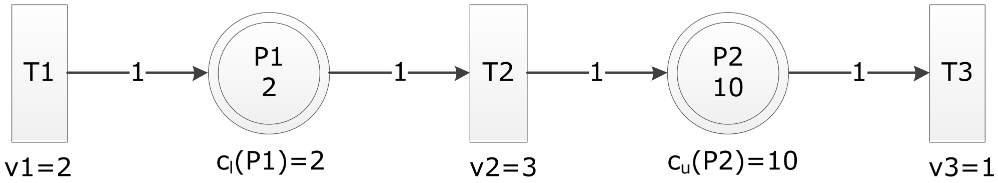


Figure 4.27: Continuous capacitive Petri net with a weakly active transition (Example 4.20)

Hence, the speed of $T2$ has to be slowed down in such a way that

$$I_1 \geq O_1 \text{ and } I_2 \leq O_2.$$

The instantaneous speed of $T2$ is

$$\tilde{v}_2 = \min\left(\frac{1}{f(P1 \to T2)} f(T1 \to P1)v_1, \frac{1}{f(T2 \to P2)} f(P2 \to T3)v_3, v_2\right)$$

$$= \min\ (2, 1,3) = 1$$

and the mark changes are

$$\frac{dm(P1)}{dt} = 2 - 1 = 1$$

$$\frac{dm(P2)}{dt} = 1 - 1 = 0.$$

A place in a continuous capacitive Petri net can have an actual input and output conflict. It has an actual output conflict if the input speed is not sufficient for firing all active output transitions with the instantaneous speed of Definition 4.47. On the other hand, it has an actual input conflict if the output speed is not sufficient for receiving marks from all active input transitions with the instantaneous speed of Definition 4.47. Similar to the continuous Petri nets without capacities, these conflicts can be resolved by priority or sharing. Thereby, it has to be ensured that the solution leads to a positive balance in the case of an output conflict and to a negative balance in the case of an input conflict. Additionally, the preliminary speeds of the involved transitions may not be exceeded.

Definition 4.48 (actual (output/input) conflict, preliminary speed)

The tuple $(P, T, F, G, f, v, c_l, c_u, m_0)$ is a continuous capacitive Petri net. A place $p_i \in P$ has an **actual output conflict** if

$$m(p_i) = c_l(p_i)$$

and

$$I_i < \sum_{t_j \in TF_{out}(p_i)} f(p_i \to t_j) \cdot \bar{v}_j.$$

A place $p_i \in P$ has an **actual input conflict** if

$$m(p_i) = c_u(p_i)$$

and

$$O_i < \sum_{t_j \in TF_{in}(p_i)} f(t_j \to p_i) \cdot \bar{v}_j,$$

whereby

$$\bar{v}_j = \min\left\{ \min_{p_i \in PI_{in}(t_j)} \left(\frac{1}{f(p_i \to t_j)} \sum_{t_k \in TF_{in}(p_i)} f(t_k \to p_i) \cdot \tilde{v}_k \right), \right.$$

$$\left. \min_{p_i \in PI_{out}(t_j)} \left(\frac{1}{f(t_j \to p_i)} \sum_{t_k \in TF_{out}(p)} f(p_i \to t_k) \cdot \tilde{v}_k \, , \right), v_j \right\}$$

$$= \min\left\{ \min_{p_i \in PI_{in}(t_k)} \left(\frac{1}{f(p_i \to t_j)} \cdot I_i \right), \min_{p_i \in PI_{out}(t_k)} \left(\frac{1}{f(t_j \to p_i)} \cdot O_i \right), v_j \right\}$$

is said to be the **preliminary speed** of a transition $t_j \in T$.

Definition 4.49 (feasible solution continuous capacitive Petri net)

A solution of an actual output conflict of a place $p_i \in P$ which satisfies the following conditions

$$B_i = I_i - O_i \geq 0$$
$$\forall\ t_j \in TF_{out}(p_i) : \tilde{v}_j \leq \bar{v}_j$$

is said to be **feasible**; otherwise, it is **infeasible**.

A solution of an actual input conflict of a place $p_i \in P$ which satisfies the following conditions

$$B_i = I_i - O_i \leq 0$$
$$\forall\ t_j \in TF_{in}(p_i) : \tilde{v}_j \leq \bar{v}_j$$

is said to be **feasible*;*** otherwise, it is **infeasible**.

The approach of Definition 4.44 has been adapted for continuous capacitive Petri nets to achieve a feasible solution of actual conflicts.

Definition 4.50 (sharing proportional to maximum speeds continuous cap. Petri net)

The tuple $(P, T, F, G, f, v, c_l, c_u, m_0)$ is a continuous capacitive Petri net. An active transition $t_j \in T$, not involved in an actual conflict, fires with the speed of Definition 4.47. If a place $p_i \in P$ has an actual conflict according to Definition 4.48, the speeds of input and output transitions have to be adapted so that the constraints of Definition 4.49 are satisfied.

This is done by **sharing proportional to the maximum speeds** of the involved input and output transitions. The instantaneous speed of an active transition $t_j \in T$ which has at least one input place with an actual output conflict and no output places with an actual input conflict is then given by

$$\tilde{v}_j = \min_{p_i \in P_{in}(t_j)} (D_i) \cdot v_j.$$

Thereby, the expression

$$D_i = \frac{I_i}{\sum_{t_k \in TF_{out}(p_i)} f(p_i \to t_k) \cdot v_k}$$

is called **output decreasing factor** of place p_i. This factor causes that the first condition of a feasible solution is satisfied with $B_i = 0$. The maximum speed is scaled by the minimum decrasing factor of all input places so that the condition $B_i \geq 0$ is always satisfied. But the preliminary speed could be exceeded for some transitions. Then the speeds of all transitions $t_l \in TF_{out}(p_i)$ for which $\tilde{v}_l > \bar{v}_l$ are set to the preliminary speed

$$\tilde{v}_l = \bar{v}_l.$$

The subset $\overline{TF}_{out}(p_i)$ contains all these transitions and the decreasing factor has to be modified by

$$D_i = \frac{I_i - \sum_{t_l \in \overline{TF}_{out}(p_i)} f(p_i \to t_l) \cdot \tilde{v}_l}{\sum_{t_k \in TF_{out}(p_i)} f(p_i \to t_k) \cdot v_k - \sum_{t_l \in \overline{TF}_{out}(p_i)} f(p_i \to t_l) \cdot v_l}.$$

This factor guarantees that $B_i \geq 0$ but the premilary speed could be exceeded for some transitions. Then the mentioned procedure has to be performed again.

The instantaneous speed of an active transition $t_j \in T$ which has at least one output place with an actual input conflict and no input place with an actual output conflict is then given by

$$\tilde{v}_j = \min_{p_i \in P_{out}(t_j)} (Q_i) \cdot v_j.$$

Thereby, the expression

$$Q_i = \frac{O_i}{\sum_{t_k \in TF_{in}(p_i)} f(t_k \to p_i) \cdot v_k}$$

is called **input decreasing factor** of place p_i. This factor causes that the first condition of a feasible solution is satisfied with $B_i = 0$. The maximum speed is scaled by the minimum decrasing factor of all input places so that the condition $B_i \leq 0$ is always satisfied. But the preliminary speed could be exceeded for some transitions. Then the speeds of all $t_l \in TF_{in}(p_i)$ for which $\tilde{v}_l > \bar{v}_l$ is set to

$$\tilde{v}_l = \bar{v}_l.$$

The subset $\overline{TF}_{in}(p_i)$ contains all these transitions. Then the decreasing factor has to be modified by

$$Q_i = \frac{O_i - \sum_{t_l \in \overline{TF}_{in}(p_i)} f(t_l \to p_i) \cdot \tilde{v}_l}{\sum_{t_k \in TF_{in}(p_i)} f(t_k \to p_i) \cdot v_k - \sum_{t_l \in \overline{TF}_{in}(p_i)} f(t_l \to p_i) \cdot v_l}.$$

This factor guarantees that $B_i \leq 0$ but the premilary speed could be exceeded for some transitions. Then the mentioned procedure has to be performed again.

The instantaneous speed of an active transition $t_j \in T$ which has at least one input place with an actual output conflict and at least one output place with an actual input conflict is then given by

$$\tilde{v}_j = \min\left(\min_{p_i \in P_{in}(t_j)} (D_i), \min_{p_i \in P_{out}(t_j)} (Q_i)\right) \cdot v_j.$$

If this speed exceeds the preliminary speed, the mentioned procedure above has to be performed again.

Example 4.21

Figure 4.28 represents two continuous capacitive Petri nets which only differ in the maximum speed v_4 of transition $T4$. Place $P2$ of the left Petri net has no actual conflict. Transitions $T1$ and $T2$ are weakly output active due to $m(P1) = c_u(P1)$, $m(P2) = c_u(P2)$, $m(P3) = c_u(P3)$ but they are all emptied, i.e. $O_1, O_2, O_3 > 0$. The instantaneous speeds are given by

$$\begin{aligned}\tilde{v}_1 = \bar{v}_1 &= \min\left(\frac{1}{f(T1 \to P1)} f(P1 \to T3) v_{3,} \frac{1}{f(T1 \to P2)} f(P2 \to T4) v_4, v_1\right)\\ &= \min\left(\frac{1}{1} \cdot 1 \cdot 3, \frac{1}{3} \cdot 1 \cdot 10.1, 3\right) = 3\\ \tilde{v}_2 = \bar{v}_2 &= \min\left(\frac{1}{f(T2 \to P2)} f(P2 \to T4) v_{4,} \frac{1}{f(T2 \to P3)} f(P3 \to T5) v_5, v_2\right)\\ &= \min\left(\frac{1}{1} \cdot 1 \cdot 10.1, \frac{1}{2} \cdot 1 \cdot 1{,}2\right) = \frac{1}{2}\end{aligned}$$

and, hence,

$$O_2 = 10.1 \geq f(T1 \to P2) \cdot \tilde{v}_1 + f(T2 \to P2) \cdot \tilde{v}_2 = 9\frac{1}{2}.$$

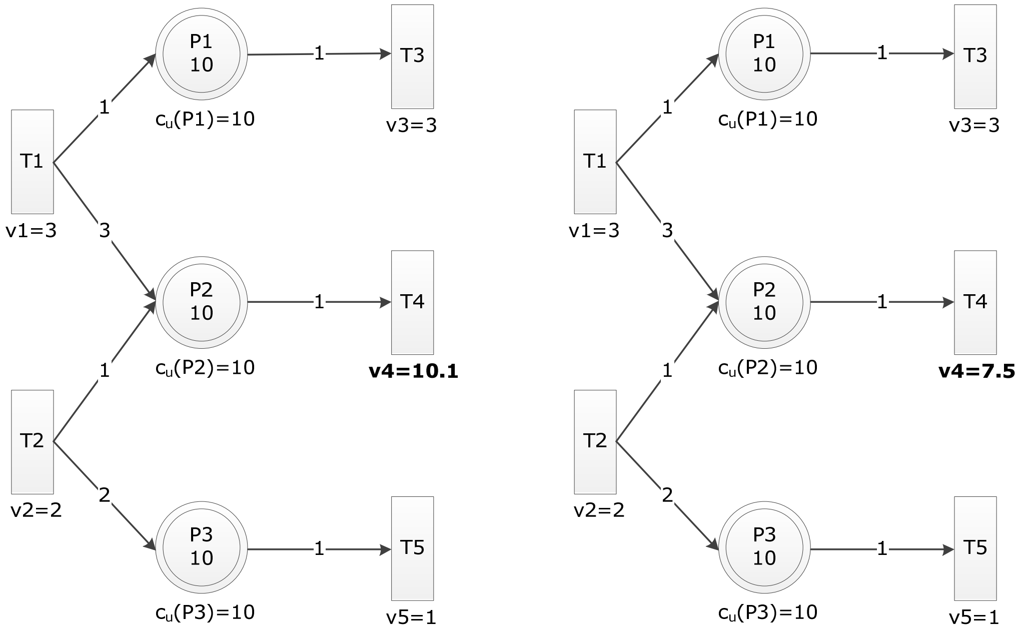


Figure 4.28: Continuous capacitive Petri nets without actual input conflict (left) and with actual input conflict (right) (Example 4.21)

The mark changes are

$$\begin{aligned}\frac{dm(P1)}{dt} &= f(T1 \to P1) \cdot \tilde{v}_1 - f(P1 \to T3) \cdot v_3 = 0\\ \frac{dm(P2)}{dt} &= f(T1 \to P2) \cdot \tilde{v}_1 + f(T2 \to P2) \cdot \tilde{v}_2 - f(P2 \to T4) \cdot v_4 = -0.6\\ \frac{dm(P3)}{dt} &= f(T2 \to P3) \cdot \tilde{v}_2 - f(P3 \to T5) \cdot v_5 = 0.\end{aligned}$$

However, $P2$ of the right Petri net has an actual input conflict due to

$$\bar{v}_1 = \min\left(\frac{1}{f(T1 \to P1)} f(P1 \to T3) v_{3,} \frac{1}{f(T1 \to P2)} f(P2 \to T4) v_4, v_1\right)$$

$$= \min\left(\frac{1}{1} \cdot 1 \cdot 3, \frac{1}{3} \cdot 1 \cdot 7.5, 3\right) = 2.5$$

$$\bar{v}_2 = \min\left(\frac{1}{f(T2 \to P2)} f(P2 \to T4) v_{4,} \frac{1}{f(T2 \to P3)} f(P3 \to T5) v_5, v_2\right)$$

$$= \min\left(\frac{1}{1} \cdot 1 \cdot 7.5, \frac{1}{2} \cdot 1 \cdot 1{,}2\right) = \frac{1}{2}$$

$$O_2 = 7.5 \not\geq 3 \cdot \bar{v}_1 + 1 \cdot \bar{v}_2 = 8.$$

The resolution of this conflict is performed by sharing proportional to the maximum speeds of the involved transitions $T1$ and $T2$. At first, the input decreasing factor of $P2$ is calculated

$$Q_2 = \frac{O_2}{f(T1 \to P2) \cdot v_1 + f(T2 \to P2) \cdot v_2} = \frac{7.5}{3 \cdot 3 + 1 \cdot 2} = \frac{15}{22}.$$

Then the speeds of $T1$ and $T2$ are decreased by this factor

$$\tilde{v}_1 = Q_2 \cdot v_1 = \frac{15}{22} \cdot 3 = 2\frac{1}{22} \leq \bar{v}_1$$

$$\tilde{v}_2 = Q_2 \cdot v_2 = \frac{15}{22} \cdot 2 = 1\frac{4}{11} \not\leq \bar{v}_2.$$

But the calculated speed of $T2$ exceeds the preliminary speed so that a feasible solution is achieved by setting the instantaneous speed to the preliminary speed

$$\tilde{v}_2 = \bar{v}_2 = \frac{1}{2}$$

and the factor for decreasing the speed of $T1$ has to be recalculated by

$$Q_2 = \frac{7.5 - 1 \cdot \tilde{v}_2}{3 \cdot v_1} = \frac{7}{9}$$

$$\tilde{v}_1 = Q_2 \cdot v_1 = 2\frac{1}{3}.$$

The mark changes are

$$\frac{dm(P1)}{dt} = f(T1 \to P1) \cdot \tilde{v}_1 - f(P1 \to T3) \cdot v_3 = -\frac{2}{3}$$

$$\frac{dm(P2)}{dt} = f(T1 \to P2) \cdot \tilde{v}_1 + f(T2 \to P2) \cdot \tilde{v}_2 - f(P2 \to T4) \cdot v_4 = 0$$

$$\frac{dm(P3)}{dt} = f(T2 \to P3) \cdot \tilde{v}_2 - f(P3 \to T5) \cdot v_5 = 0.$$

Example 4.22

Figure 4.29 shows a continuous capacitive Petri net. Transitions $T4$ and $T5$ are weakly active because the input places have markings equal to the minimum capacities and the output places have markings equal to the maximum capacities; but the input places are fed and the output places are emptied.

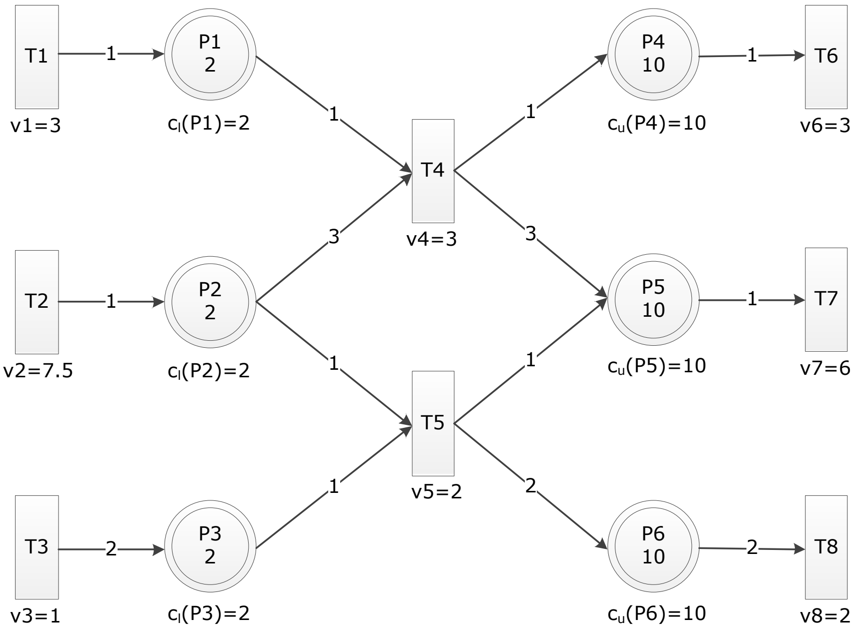


Figure 4.29: Continuous capacitive Petri net with actual input and output conflict (Example 4.22)

The preliminary speeds

$$\bar{v}_4 = \min\left(\min\left(\frac{1}{f(P1 \to T4)} f(T1 \to P1) v_1, \frac{1}{f(P2 \to T4)} f(T2 \right.\right.$$
$$\left.\to P2) v_2\right), \min\left(\frac{1}{f(T4 \to P4)} f(P4 \to T6) v_6, \frac{1}{f(T4 \to P5)} f(P5\right.$$
$$\left.\left.\to T7) v_7\right), v_4\right) = \min(\min(3{,}2.5), \min(3{,}2), 3) = 2$$

$$\bar{v}_5 = \min\left(\min\left(\frac{1}{f(P2 \to T5)} f(T2 \to P2) v_2, \frac{1}{f(P3 \to T5)} f(T3 \right.\right.$$
$$\left.\to P3) v_3\right), \min\left(\frac{1}{f(T5 \to P5)} f(P5 \to T7) v_7, \frac{1}{f(T5 \to P6)} f(P6\right.$$
$$\left.\left.\to T8) v_8\right), v_5\right) = \min(\min(7.5{,}2), \min(6{,}2), 2) = 2$$

reveal that $P2$ has an actual output conflict

$$I_3 = 7.5 < f(P2 \to T4) \cdot \bar{v}_4 + f(P2 \to T5) \cdot \bar{v}_5 = 8$$

and that $P5$ has an actual input conflict

$$O_5 = 6 < f(T4 \to P5) \cdot \bar{v}_4 + f(T5 \to P5) \cdot \bar{v}_5 = 8.$$

The resolution of these conflicts is performed by sharing according to Definition 4.50. The decreasing factors

$$D_2 = \frac{I_2}{f(P2 \to T4) \cdot v_4 + f(P2 \to T5) \cdot v_5} = \frac{15}{22}$$

$$Q_5 = \frac{O_5}{f(T4 \to P5) \cdot v_4 + f(T5 \to P5) \cdot v_5} = \frac{6}{11}$$

lead to the instantaneous speeds

$$\tilde{v}_4 = \min(D_2, Q_5) \cdot v_4 = 1\frac{7}{11} \leq \bar{v}_4$$

$$\tilde{v}_5 = \min(D_2, Q_5) \cdot v_5 = 1\frac{1}{11} \leq \bar{v}_5$$

which is a feasible solution. The corresponding mark changes are

$$\frac{dm(P1)}{dt} = f(T1 \to P1) \cdot v_1 - f(P1 \to T4) \cdot \tilde{v}_4 = 1\frac{4}{11}$$

$$\frac{dm(P2)}{dt} = f(T2 \to P2) \cdot v_2 - f(P2 \to T4) \cdot \tilde{v}_4 - f(P2 \to T5) \cdot \tilde{v}_5 = 1\frac{1}{2}$$

$$\frac{dm(P3)}{dt} = f(T3 \to P3) \cdot v_3 - f(P3 \to T5) \cdot \tilde{v}_5 = \frac{10}{11}$$

$$\frac{dm(P4)}{dt} = f(T4 \to P4) \cdot \tilde{v}_4 - f(P4 \to T6) \cdot v_6 = -1\frac{4}{11}$$

$$\frac{dm(P5)}{dt} = f(T4 \to P5) \cdot \tilde{v}_4 + f(T5 \to P5) \cdot \tilde{v}_5 - f(P5 \to T7) \cdot v_7 = 0$$

$$\frac{dm(P6)}{dt} = f(T5 \to P6) \cdot \tilde{v}_5 - f(P6 \to T8) \cdot v_8 = -1\frac{9}{11}.$$

4.4.2 Continuous Extended Petri Nets

The extended Petri net concept of Definition 4.23 with test, inhibitor, and read arcs has been transferred to the continuous Petri net formalism.

Definition 4.51 (continuous extended Petri net)

The tuple $(P, T, F, G, \mathcal{T}, \mathcal{I}, \mathcal{R}, f, v, m_0)$ is a **continuous extended Petri net** if (P, T, F, G, f, v, m_0) is a continuous Petri net, $\mathcal{T} \subseteq (P \times T)$ is a set of test arcs, $\mathcal{I} \subseteq (P \times T)$ is a set of inhibitor arcs, $\mathcal{R} \subseteq (P \times T)$ is a set of read arcs, and the arc weight function f is modified such that $f: (F \cup G \cup \mathcal{T} \cup \mathcal{I}) \to \mathbb{R}_{\geq 0}$, whereby $f\left(\left(p_i \to t_j\right)_{\mathcal{T}}\right)$ is the weight of the test arc $\left(p_i \to t_j\right)_{\mathcal{T}}$ and $f\left(\left(p_i \to t_j\right)_{\mathcal{I}}\right)$ is the weight of the inhibitor arc $\left(p_i \to t_j\right)_{\mathcal{I}}$. If $\left(p_i \to t_j\right) \in F$ and $\left(p_i \to t_j\right)_{\mathcal{T}} \in \mathcal{T}$ or $\left(p_i \to t_j\right) \in F$ and $\left(p_i \to t_j\right)_{\mathcal{I}} \in \mathcal{I}$ then the arc is called **double arc**.

The activation of transitions in a continuous extended Petri net requires that markings of places connected by test arcs are greater than the arc weights and markings of places connected by inhibitor arcs are less than the arc weights. The markings of these places are not changed by firing so that the firing process and the resolution of actual conflicts have been adopted from Definition 4.41 and Definition 4.44, respectively. It has to be stated that the same place can be connected with the same transition by a test and normal arc as well as by an inhibitor and normal arc. These arcs are called **double arcs**. Then a transition is also weakly active if place and transition are connected by a test and normal arc, the marking is equal to the weight of the test arc, and the place is fed.

Definition 4.52 (activation continuous extended Petri net)

The tuple $(P, T, F, G, \mathcal{T}, \mathcal{I}, \mathcal{R}, f, v, m_0)$ is a continuous extended Petri net. A transition $t_j \in T$ is **active** if and only if

$\forall\, p_i \in P_{in}(t_j)$:

$$\begin{cases} m(p_i) > 0 \vee (m(p_i) = 0 \wedge I_i > 0) & if\ (p_i \to t_j) \in F \\ m(p_i) > f\left((p_i \to t_j)_{\mathcal{T}}\right) & if\ (p_i \to t_j)_{\mathcal{T}} \in \mathcal{T} \wedge (p_i \to t_j) \notin F \\ m(p_i) > f\left((p_i \to t_j)_{\mathcal{T}}\right) & if\ (p_i \to t_j)_{\mathcal{T}} \in \mathcal{T} \wedge (p_i \to t_j) \in F \\ \quad \vee \left(m(p_i) = f\left((p_i \to t_j)_{\mathcal{T}}\right) \wedge I_i > 0\right) & \\ m(p_i) < f\left((p_i \to t_j)_{\mathcal{I}}\right) & if\ (p_i \to t_j)_{\mathcal{I}} \in \mathcal{I}, \end{cases}$$

It is **strongly active** if

$\forall\, p_i \in P_{in}(t_j)$:

$$\begin{array}{ll} m(p_i) > 0 & if\ (p_i \to t_j) \in F \\ \wedge\ m(p_i) > f\left((p_i \to t_j)_{\mathcal{T}}\right) & if\ (p_i \to t_j)_{\mathcal{T}} \in \mathcal{T} \wedge (p_i \to t_j) \in F \end{array}$$

is also satisfied and **weakly active** otherwise, whereby, $PI_{in}(t_j)$ ist the set of input places with

$$\begin{array}{ll} m(p_i) = 0, & if\ (p_i \to t_j) \in F \\ \vee\ m(p_i) = f\left((p_i \to t_j)_{\mathcal{T}}\right) \wedge\ I_i > 0, & if\ (p_i \to t_j)_{\mathcal{T}} \in \mathcal{T} \wedge (p_i \to t_j) \in F\,. \end{array}$$

Example 4.23

Figure 4.30 shows two continuous extended Petri nets. The left Petri net contains a test arc which causes that transition $T2$ fires first when the marking of $P2$ is greater than 3.3. The right Petri net has an inhibitor arc so that transition $T4$ fires first when the marking of $P4$ is less than 3.3.

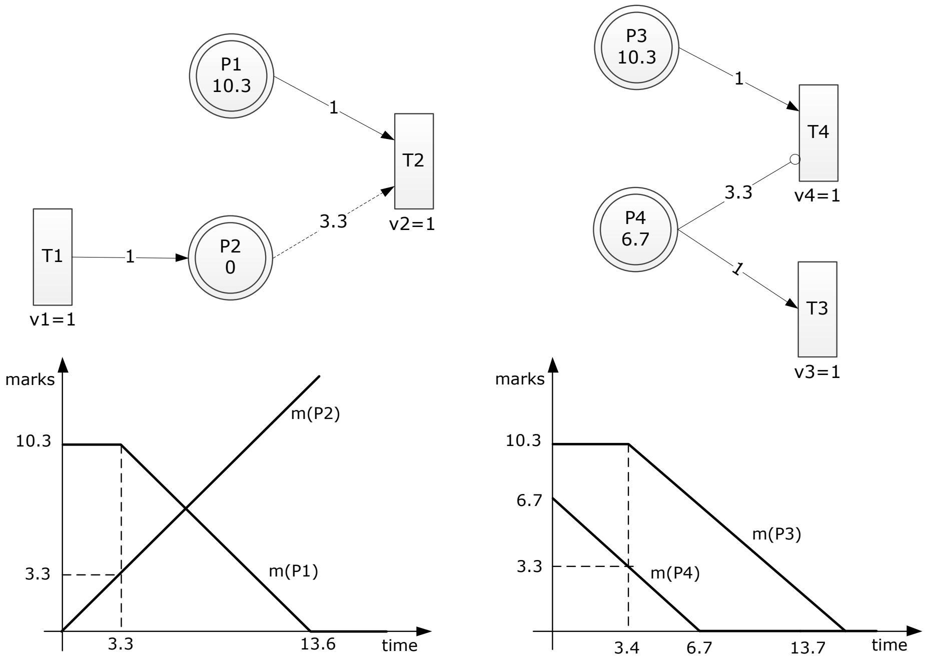


Figure 4.30: Continuous extended Petri nets with a test arc (left) and with an inhibitor arc (right) and the corresponding mark evolutions (Example 4.23)

4.4.3 Continuous Extended Capacitive Petri Nets

The continuous capacitive Petri nets has been also extended by test, inhibitor, and read arcs. Therefore, the activation process has been modified to include that input places connected by test and inhibitor arcs must have appropriate markings. However, the firing process and the resolution of actual conflicts have been adopted from Definition 4.47 and Definition 4.50.

Definition 4.53 (continuous extended capacitive Petri net)

The tuple $(P, T, F, G, \mathcal{T}, \mathcal{I}, \mathcal{R}, f, c_l, c_u, v, m_0)$ is called **continuous extended capacitive Petri net** if

- (P, T, F, G, f, v, m_0) is a continuous Petri net,
- $\mathcal{T} \subseteq (P \times T)$ is a set of test arcs,
- $\mathcal{I} \subseteq (P \times T)$ is a set of inhibitor arcs,
- $\mathcal{R} \subseteq (P \times T)$ is a set read arcs,
- $f\colon (F \cup G \cup \mathcal{T} \cup \mathcal{I}) \to \mathbb{R}_{\geq 0}$ is a modified arc weight function,
- $c_l\colon P \to \mathbb{R}_{\geq 0}$ are the minimum capacities of the places,
- $c_u\colon P \to \mathbb{R}_{\geq 0}$ are the maximum capacities of the places,

whereby the initial marking m_0 must satisfy the condition

$$c_l(p_i) \leq m_0(p_i) \leq c_u(p_i) \quad \forall\, p_i \in P.$$

Definition 4.54 (activation continuous extended capacitive Petri net)

The tuple $(P, T, F, G, \mathcal{T}, \mathcal{I}, \mathcal{R}, f, c_l, c_u, v, m_0)$ is a continuous extended capacitive Petri net. A transition $t_j \in T$ is **active** if and only if

$\forall\, p_i \in P_{in}(t_j)$:

$$\begin{cases} m(p_i) > c_l(p_i) & if\ (p_i \to t_j) \in F \\ \quad \vee\ (m(p_i) = c_l(p_i) \wedge I_i > 0) & \\ m(p_i) > f\left((p_i \to t_j)_{\mathcal{T}}\right) & if\ (p_i \to t_j)_{\mathcal{T}} \in \mathcal{T} \wedge (p_i \to t_j) \notin F \\ m(p_i) > f\left((p_i \to t_j)_{\mathcal{T}}\right) & if\ (p_i \to t_j)_{\mathcal{T}} \in \mathcal{T} \wedge (p_i \to t_j) \in F \\ \ \vee \left(m(p_i) = f\left((p_i \to t_j)_{\mathcal{T}}\right) \wedge I_i > 0\right) & \\ m(p_i) < f\left((p_i \to t_j)_{\mathcal{I}}\right) & if\ (p_i \to t_j)_{\mathcal{I}} \in \mathcal{I}, \end{cases}$$

and

$$\forall\, p_i \in P_{out}(t_j) :\ m(p_i) < c_u(p_i) \vee (m(p_i) = c_u(p_i) \wedge O_i > 0).$$

It is **strongly input active** if

$\forall\, p_i \in P_{in}(t_j)$:

$$\begin{array}{ll} m(p_i) > c_l(p_i) & if\ (p_i \to t_j) \in F \\ \wedge\ m(p_i) > f\left((p_i \to t_j)_{\mathcal{T}}\right) & if\ (p_i \to t_j)_{\mathcal{T}} \in \mathcal{T} \wedge (p_i \to t_j) \in F \end{array}$$

is also satisfied and **weakly input active** otherwise. It is **strongly output active** if

$$\forall\, p_i \in P_{out}(t_j) :\ m(p_i) < c_u(p_i)$$

is also satisfied and **weakly output active** otherwise. If it is strongly input and output active, it is **strongly active**. If it is weakly input and output active, it is **weakly active**. Thereby, $PI_{in}(t_j)$ ist the set of input places with

$$\begin{array}{ll} m(p_i) = c_l(p_i), & if\ (p_i \to t_j) \in F \\ \vee\ m(p_i) = f\left((p_i \to t_j)_{\mathcal{T}}\right) \wedge\ I_i > 0, & if\ (p_i \to t_j)_{\mathcal{T}} \in \mathcal{T} \wedge (p_i \to t_j) \in F\ . \end{array}$$

and $PI_{out}(t_j)$ is the set of output places of t_j with $m(p_i) = c_u(p_i) \wedge O_i > 0$.

4.4.4 Continuous Functional Petri Nets

The considered continuous Petri net concepts so far comprise constant maximum speeds. This has been modified by maximum speed functions that depend on time and/or on markings to enhance the modeling power as suggested in (Dubois et al. 1994) and (David and Alla 2001). Continuous functional Petri nets are useful for modeling the environment or the control of a system and regarding biological systems, it enables the modeling of nearly all kinetics effects (see Example 4.24).

Definition 4.55 (continuous functional Petri net)

The tuple (P, T, F, G, f, v, m_0) is a **continuous functional Petri net** if (P, T, F, G, f, v, m_0) is a continuous Petri net and the maximum speed function is modified such that $v: (T, m, time) \to \mathbb{R}_{\geq 0}$ is a dynamic maximum speed function which assigns every transition $t_j \in T$ a function $v_j = v(t_j, m, time)$ which depends on a subset of concrete markings m and/or on time.

The definition is similar to Definition 4.38 the only difference being that the maximum speed function depends on time and/or markings. Hence, the previously mentioned definitions for the activation and firing process kept their validity and have been adopted (Definition 4.40, Definition 4.41, and Definition 4.44). A continuous functional Petri net can also have capacities or it can be extended by the specific arcs or both. Then the respective definitions of the previous sections are valid.

Example 4.24

Figure 4.31 shows the biochemical reaction of Figure 4.12 modeled by a continuous functional Petri net. The substrate (S), product (P), and enzyme (E) are modeled by continuous places and the reaction (R) between them by a continuous transition.

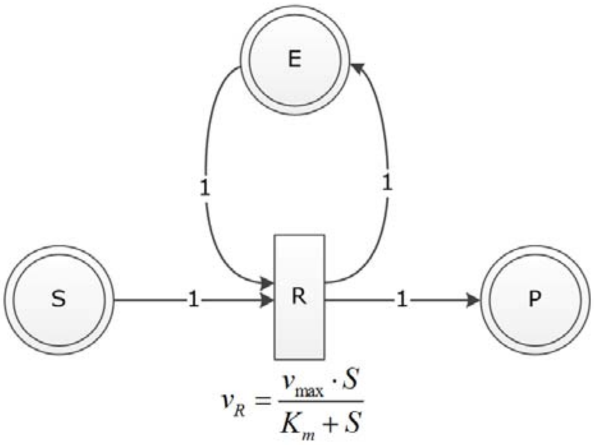


Figure 4.31: Continuous functional Petri net of the biochemical reaction in Figure 4.12 (Example 4.24)

The marks represent continuous concentrations instead of molecules in the discrete model. The speed v of the conversion from substrate to product is described by the Michaelis-Menten-Kinetics

$$v = \frac{v_{max} \cdot S}{K_m + S},$$

where S is the substrate concentration, v_{max} is the maximum reaction rate, and K_m is the Michaelis constant. Hence, this function is the maximum speed of transition R and depends on the marking of place S.

4.4.5 Conditional Continuous Petri Nets

The transitions of the continuous Petri net concepts, introduced in the previous sections, are provided with additional conditions that have to be satisfied so that the transition can become active. The respective activation process has been modified while the firing process and the resolution of actual conflicts remain the same (Definition 4.41 and Definition 4.44).

Definition 4.56 (conditional continuous Petri)

The tuple $(P, T, F, G, f, v, s, m_0)$ is a **conditional continuous Petri net**, if (P, T, F, G, f, v, m_0) is a continuous Petri net and $s: (\mathrm{T}, \mathcal{E}) \rightarrow \{\mathrm{true}, \mathrm{false}\}$ is an condition function that assigns every transition $t_j \in T$ a condition $s_j = s(t_j, \mathcal{E})$ depending on several environmental factors $\mathcal{E}$ e.g. time.

Definition 4.57 (activation conditional continuous Petri)

The tuple $(P, T, F, G, f, v, s, m_0)$ is a conditional continuous Petri net. A transition $t_j \in T$ is **active** if and only if the conditions of Definition 4.40 are fulfilled and, additionally, the condition s_j**.**

4.5 Hybrid Petri Nets

For modeling biological systems it is often necessary to combine discrete and continuous processes. Examples are regulation mechanisms and metabolic reactions of substances which switch from production to consumption or vice versa when specific environmental conditions appear. This is equivalent to suddenly having another Petri net. In order to model this behavior, hybrid Petri nets are introduced which contain discrete places and transitions as well as continuous places and transitions as suggested in (Le Bail et al. 1991), (David and Alla 2001), and (David and Alla 2005).

The markings of discrete places are non-negative integers denoted by tokens and the markings of continuous places are non-negative real numbers called marks. Delays are associated with discrete transitions and maximum speeds with continuous transitions.

Only hybrid Petri nets are considered by which time is associated with the behavior; hereafter the term hybrid Petri net is simplified used for timed hybrid Petri net. The definitions for hybrid Petri nets developed within this work are given below.

Definition 4.58 (hybrid Petri net)

The tuple $(PD, PC, TD, TS, TC, F, G, f, e, \wp, d, h, v, m_0)$ is a **hybrid Petri net** if

- $PD = \{pd_1,\ pd_2, \ldots, pd_{pd}\}$ is a finite set of discrete places,
- $PC = \{pc_1,\ pc_2, \ldots, pc_{pc}\}$ is a finite set of continuous places,
- $TD = \{td_1,\ td_2, \ldots, td_{td}\}$ is a finite set of discrete transitions,
- $TS = \{ts_1,\ ts_2, \ldots, ts_{ts}\}$ is a finite set of stochastic transitions,
- $TC = \{tc_1,\ tc_2, \ldots, tc_{tc}\}$ is a finite set of continuous transitions,
- $PD, TD, TS, PC,$ and TC are pairwise disjoint,
- $F \subseteq (PD \times TD \cup PD \times TS \cup PD \times TC \cup PC \times TC \cup PC \times TD \cup PC \times TS)$ is a set of arcs from places to transitions,
- $G \subseteq (TD \times PD \cup TD \times PC \cup TS \times PD \cup TS \times PC \cup TC \times PC \cup TC \times PD)$ is a set of arcs from transitions to places,
- $f: (F \cup G) \rightarrow \{\mathbb{N}_0: p_i \in PD\ , \mathbb{R}_{\geq 0}: p_i \in PC\}$ is a arc weight function which assigns every arc connected to a discrete place a non-negative integer and all others a non-negative real number,
- if $p_i \in PD$ and $t_j \in TC$ then $(p_i \rightarrow t_j) \in F$ if and only if $(t_j \rightarrow p_i) \in G$ and $f(p_i \rightarrow t_j) = f(t_j \rightarrow p_i)$,
- $e: (PD \cup PC) \rightarrow \{prio, prob\}$ is an enabling function that assigns every place either the resolution type priority or probability,
- $\wp: (F \cup G) \rightarrow \Big\{\{1,2, \ldots, \max(n_{out}, n_{in})\}: e(p_i) = prio \wedge t_j \in TD,\ [0,1]: e(p_i) = prob \wedge t_j \in TD\Big\}$ is an enabling function which assigns every arc connected to a discrete transition either a priority or a probability according to the resolution type,

- if $e(p_i) = prio$ then $\wp(p_i \to t_k) \neq \wp(p_i \to t_l)\ \forall t_k, t_l \in TD_{out}(p_i)$ and $\wp(t_k \to p_i) \neq \wp(t_l \to p_i)\ \forall t_k, t_l \in TD_{in}(p_i)$, if $e(p_i) = prob$ then $\sum_{t_k \in TD_{out}(p_i)} \wp(p_i \to t_k) = 1$ and $\sum_{t_k \in TD_{in}(p_i)} \wp(t_k \to p_i) = 1$,
- $d\colon TD \to \mathbb{R}_{\geq 0}$ is a delay function which assigns every discrete transition a non-negative, real-valued delay,
- $h\colon (TS, m) \to \mathbb{R}_{\geq 0}$ is a hazard function which assigns every stochastic transition a non-negative, real-valued random delay depending on a concrete marking m,
- $v\colon TC \to \mathbb{R}_{\geq 0}$ is a maximum speed function which assigns every continuous transition a non-negative, real-valued maximum speed,
- $m_0\colon \{PD \to \mathbb{N}_0, PC \to \mathbb{R}_{\geq 0}\}$ is the initial marking.

The basic concepts modeled by hybrid Petri nets are **influence** and **conversion**. Figure 4.32 shows at the top two examples of influence. In the left Petri net, the behavior of the continuous part is influenced by the discrete part such that $P3$ must have at least one token to activate $T1$. According to Definition 4.58, discrete input places of continuous transitions must also be their output places and vice versa with arcs of the same weights. Hence, the markings of discrete places are not changed by firing a continuous transition. They can only influence the time when the transition can become active.

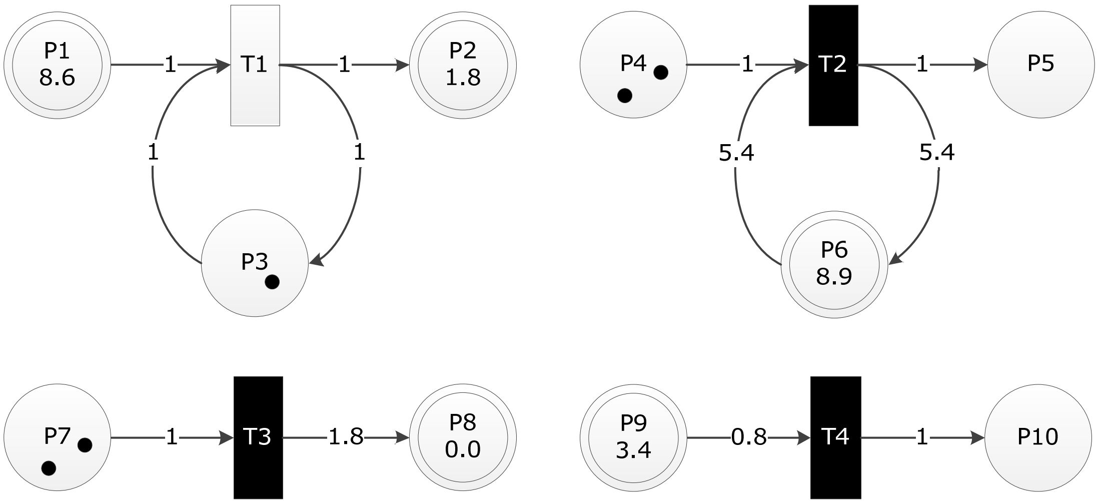


Figure 4.32: Basic concepts of hybrid Petri nets; top: influence of a discrete part on a continuous part (left) and of a continuous part on a discrete part (right); bottom: conversion of a discrete marking to a continuous marking (left) and of a continuous marking to a discrete marking (right)

In the right Petri net, the behavior of the discrete part is influenced by the continuous part. Place $P6$ must have at least 5.4 marks so that $T2$ can become active. Because the weight of the arc from $P6$ to $T2$ equals the weight of the arc from $T2$ back to $P6$, the marking of $P6$

remains the same when $T2$ fires. It only influences the time when the transition can become active.

At the bottom of Figure 4.32 are two examples of conversion. In the left Petri net, a discrete marking is converted to a continuous marking by a discrete transition. The firing process is performed by removing one token from $P7$ and adding 1.8 marks to $P8$.

In the right Petri net, a continuous marking is converted to a discrete marking. When $T4$ fires, 0.8 marks are removed from $P9$ and one token is added to $P10$. Conversions from discrete markings to continuous markings and vice versa can only be performed by discrete transitions.

Example 4.25

A biological example for influence is the protein synthesis which is transformed from a stochastic Petri net (cp. Example 4.16) to a hybrid Petri net depicted in Figure 4.33. The active gene (AG) influences the protein synthesis. Only if the place AG has one token, the protein can be producted by the continuous transition $T3$. But AG is not consumed in this reaction so that it is input and output of $T3$, simultaneously. This can be also modeled by a test arc from AG to $T3$. The activation and inactivation of the gene proceed discretely while the protein is produced and degraded continuously.

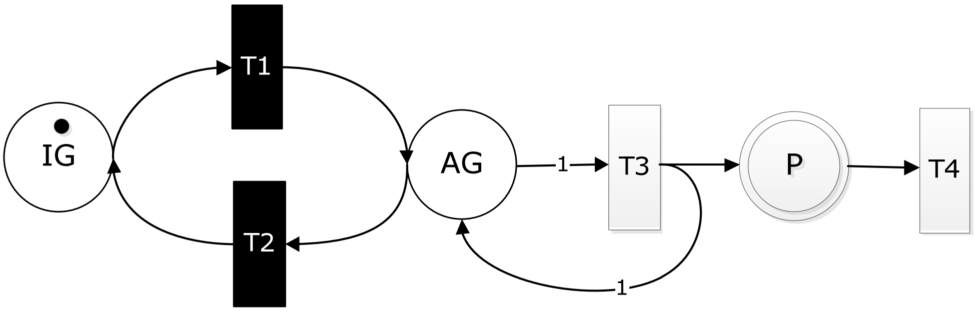


Figure 4.33: Hybrid Petri net model of protein synthesis (IG = inactive gene, AG = active gene, P = protein, T1 = activation, T2 = inactivation, T3 = synthesis, and T4 = degradation) (Example 4.25)

Remark 2.7

Hereafter the set of all input places of a transition t_j is denoted by $P_{in}(t_j)$ and the set of all discrete and continuous input places are denoted by $PD_{in}(t_j)$ and $PC_{in}(t_j)$, respectively. The notations for the set of (discrete/continuous) output places are similar $P_{out}(t_j)$, $PD_{out}(t_j)$, and $PC_{out}(t_j)$, respectively. The set of all input transitions of a place p_i is denoted by $T_{in}(p_i)$ and the set of all discrete/stochastic and continuous input transitions are denoted by $TD_{in}(p_i)$ and $TC_{in}(p_i)$, respectively. The notations for the set of (discrete/ stochastic and continuous) output transitions are similar $T_{out}(p_i)$, $TD_{out}(p_i)$, and $TC_{out}(p_i)$, respectively.

Definition 4.59 (activation hybrid Petri net)

The tuple $(PD, PC, TD, TS, TC, F, G, f, e, \wp, d, h, v, m_0)$ is a hybrid Petri net. A discrete/stochastic transition $t_j \in (TD \cup TS)$ is **active** if and only if

$$\forall\, p_i \in P_{in}(t_j) : \ m(p_i) \geq f(p_i \rightarrow t_j).$$

A continuous transition $t_j \in TC$ is **active** if and only if

$$\forall\, p_i \in PC_{in}(t_j) : \ m(p_i) > 0 \vee (m(p_i) = 0 \wedge I_i > 0)$$

and

$$\forall\, p_i \in PD_{in}(t_j) : \ m(p_i) \geq f(p_i \rightarrow t_j).$$

Four different kinds of actual conflicts can occur in a hybrid Petri net which have to be resolved (David and Alla 2005).

Definition 4.60 (actual conflicts and resolutions hybrid Petri nets)

The tuple $(PD, PC, TD, TS, TC, F, G, f, e, \wp, d, h, v, m_0)$ is a hybrid Petri net:

- **Type-1-conflict**: actual conflict according to Definition 4.6 and Definition 4.31 of a place $p_i \in (PD \cup PC)$ and two or more discrete/stochastic transitions $t_j \in (TD \cup TS)$. It can only occur when all transitions t_j become firable exactly at the same time (Remark 4.3, Remark 4.4). This conflict can be resolved either by priorities or by probabilities (Definition 4.9 and Definition 4.10).
- **Type-2-conflict*:*** actual conflict according to Definition 4.42 of a place $p_i \in PC$ and two or more continuous transitions $t_j \in TC$. It can be resolved by sharing (Definition 4.44).
- **Type-3-conflict*:*** actual conflict of a place $p_i \in (PD \cup PC)$, one or more discrete/stochastic transitions $t_j \in (TD \cup TS)$, and one or more continuous transitions $t_k \in TC$. If there is a conflict between discrete/stochastic and continuous transitions, the discrete/stochastic transitions take priority over the continuous transitions (cp. David and Alla 2005).
- **Type-4-conflict*:*** actual conflict of a discrete place $p_i \in PD$ and two or more continuous transitions $t_j \in TC$. It is solved by priorities (Definition 4.9).

The type-1-conflict is represented in Example 4.1, Example 4.3, Example 4.4, Example 4.11, and Example 4.14. The type-2-conflict is part of Example 4.18.

Example 4.26

Figure 4.34 shows two examples of type-3-conflicts. At time 0, transition $T1$ of the left Petri net becomes active and fires continuously. At time 2, the delay of $T2$ is passed and it becomes firable. At this point in time, $P3$ has an actual conflict because it cannot fire a token in $T1$ and $T2$, simultaneously. Hence, the rule of Definition 4.60 is applied so that $T2$ takes priority over $T1$ and fires. At time 0, transitions $T3$ and $T4$ of the right Petri net fire continuously. At time 1, $m(P4) = 1$ and the delay of $T5$ is passed; hence, $P4$ has an actual conflict. The conflict only occurs when the marking equals the arc weight. If $m(P4) < f(P4 \rightarrow T5)$ or $m(P4) > f(P4 \rightarrow T5)$, there is no conflict. It is solved by the rule of Definition 4.60 so that $T5$ takes priority over $T4$ and fires. This rule is intuitively logical because the firing of a continuous transition is a continuous flow and the firing of a discrete transition is an extreme change of the Petri net marking (David and Alla 2005).

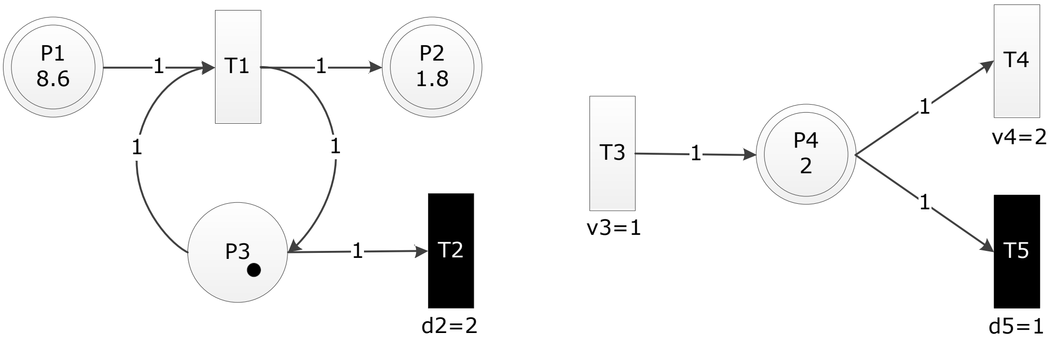


Figure 4.34: Hybrid Petri nets with a type-3-conflict of a discrete place (left) and of a continuous place (right) (Example 4.26)

Example 4.27

Figure 4.35 shows a hybrid Petri net; place $P3$ has a type-4-conflict. At time 0, $P3$ can either activate $T1$ or $T2$ but not both simultaneously.

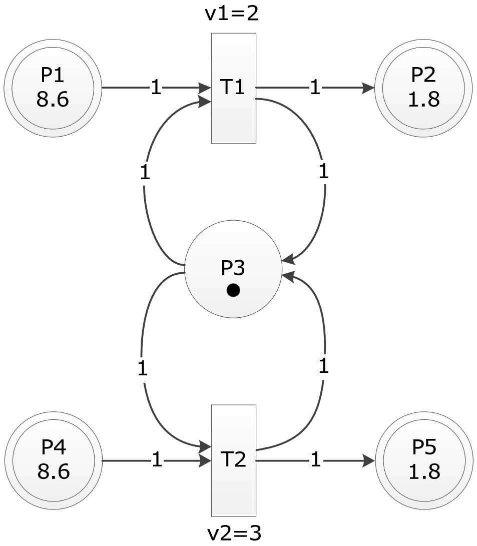


Figure 4.35: Hybrid Petri net with a type-4-conflict (Example 4.27)

This conflict can be solved by prioritization of the transitions. If $T1$ takes priority over $T2$, $T1$ becomes active and fires and if $T2$ takes priority over $T1$, $T2$ becomes active and fires. Therefore, all continuous output transitions of a discrete place have to be provided with priorities.

Discrete and continuous transitions fire in the same manner as determined in Definition 4.13 and Definition 4.41, respectively. The recalculation of a discrete marking is performed as described in the basic concepts. However, the recalculation of a continuous marking has to be modified due to discrete mark changes caused by firing discrete transitions. Thereby, a differential equation describes the flow of the continuous firing and an algebraic equation is used for the firing of the discrete transitions.

Definition 4.61 (firing process hybrid Petri net)

The tuple $(PD, PC, TD, TS, TC, F, G, f, e, \wp, d, h, v, m_0)$ is a hybrid Petri net. The **firing process** of an active continuous transition $t_j \in TC$ is described by a negative mark change of all continuous input places which is expressed by the differential equation

$$\frac{dm(p_i)}{dt} = -f\left(p_i \rightarrow t_j\right) \cdot \tilde{v}_j \qquad \forall\, p_i \in PC_{in}\left(t_j\right)$$

and a positive mark change of all continuous output places which is expressed by the differential equation

$$\frac{dm(p_i)}{dt} = f\left(t_j \rightarrow p_i\right) \cdot \tilde{v}_j \qquad \forall\, p_i \in PC_{out}\left(t_j\right).$$

An active discrete transition $t_j \in TD$ waits d_j time units before it fires and a stochastic transition $t_j \in TS$ fires when the putative firing time τ_j is reached (see Definition 4.34). Both fire by removing the arc weight from all input places

$$m'(p_i) = m(p_i) - f\left(p_i \rightarrow t_j\right)\ \forall\, p_i \in P_{in}\left(t_j\right)$$

and by adding the arc weight to all output places

$$m'(p_i) = m(p_i) + f\left(t_j \rightarrow p_i\right)\ \forall\, p_i \in P_{out}\left(t_j\right).$$

The **marking** of a discrete place $p_i \in PD$ can be recalculated by the following algebraic equation

$$m'(p_i) = m(p_i) + \sum_{t_j \in TDF_{in}(p_i)} f\left(t_j \rightarrow p_i\right) - \sum_{t_j \in TDF_{out}(p_i)} f\left(p_i \rightarrow t_j\right),$$

whereby $TDF_{in}(p_i) \subseteq (TD \cup TS)$ is the set of all discrete/stochastic firing input transitions and $TDF_{out}(p_i) \subseteq (TD \cup TS)$ is the set of all discrete/stochastic firing output transitions.

The continuous **mark change** of a continuous place $p_i \in PC$ is performed with the aid of the following differential

$$\frac{dm(p_i)}{dt} = \sum_{t_j \in TCF_{in}(p_i)} f(t_j \to p_i) \cdot \tilde{v}_j - \sum_{t_j \in TCF_{out}(p_i)} f(p_i \to t_j) \cdot \tilde{v}_j$$

and, in addition, by the following algebraic equation for the discrete mark change caused by firing connected discrete transitions

$$m_{dis}(p_i) = \sum_{t_j \in TDF_{in}(p_i)} f(t_j \to p_i) - \sum_{t_j \in TDF_{out}(p_i)} f(p_i \to t_j)$$

whereby $TCF_{in}(p_i) \subseteq TC$ is the set of all continuous firing input transitions and $TCF_{out}(p_i) \subseteq TC$ is the set of all continuous firing output transitions. At these discrete firing times the continuous marking is reinitialized by

$$m'(p_i) = m(p_i) + m_{dis}(p_i).$$

The hybrid Petri net concept of Definition 4.58 can be further modified by the variations, abbreviations, and extensions mentioned in Sections 4.1 to 4.4.

4.5.1 Variation of Transitions

Discrete, stochastic, and continuous transitions are provided with additional conditions that have to be satisfied so that the transitions can become active. Therefore, the activation definition has to be modified while the firing process remains the same (Definition 4.61).

Definition 4.62 (conditional hybrid Petri net)

The tuple $(PD, PC, TD, TS, TC, F, G, f, e, \wp, d, h, v, \mathrm{s}, m_0)$ is a **conditional hybrid Petri net**. If $(PD, PC, TD, TC, F, G, f, e, \wp, d, v, m_0)$ is a hybrid Petri net and $s: (TD \cup TS \cup TC, \mathcal{E}) \to \{\text{true}, \text{false}\}$ is an condition function that assigns every transition $t_j \in (TD \cup TS \cup TC)$ a condition $s_j = s(t_j, \mathcal{E})$ depending on several environmental factors $\mathcal{E}$, e.g. time.

Definition 4.63 (activation conditional hybrid Petri net)

The tuple $(PD, PC, TD, TS, TC, F, G, f, e, \wp, d, h, v, \mathrm{s}, m_0)$ is a conditional hybrid Petri net. A transition $t_j \in (TD \cup TS \cup TC)$ is **active** if the conditions of Definition 4.59 are fulfilled and, additionally, the condition s_j.

4.5.2 Arc Weight and Maximum Speed Functions Depending on Marking/Time

The constant arc weights and maximum speeds of the basic hybrid Petri net concept are replaced by functions which can depend on a subset of markings and/or on time. This modification is called hybrid functional Petri net and enables the modeling of nearly all kinetic effects of biological reactions.

Definition 4.64 (hybrid functional Petri net)

The tuple $(PD, PC, TD, TS, TC, F, G, f, e, \wp, d, h, v, m_0)$ is a **hybrid functional Petri net** if the arc weights and maximum speeds of Definition 4.58 are modified in the following manner

– $f: (F \cup G, m, time) \rightarrow \{\mathbb{N}_0: p_i \in PD\ , \mathbb{R}_{\geq 0}: p_i \in PC\}$ is an arc weight function which assigns every arc a weight that depends on a subset of markings m and/or on time. It is a non-negative integer if the connected place is discrete and a non-negative real-valued number otherwise.

– $v: (TC, m, time) \rightarrow\ \mathbb{R}_{\geq 0}$ is a maximum speed function which assigns every continuous transition a non-negative, real-valued maximum speed which depends on a subset of concrete markings m and/or on time.

The activation (Definition 4.59) and firing process (Definition 4.61) remains the same.

4.5.3 Capacities and Extensions

Places in a hybrid Petri net can be provided with minimum and maximum capacities. Additionally, hybrid Petri nets can be extended by test, inhibitor, and read arcs to enable the modeling of activation, inhibition, and catalysis of biological reactions.

Definition 4.65 (extended hybrid (functional) Petri net)

The tuple $(PD, PC, TD, TS, TC, F, G, \mathcal{T}, \mathcal{I}, \mathcal{R}, f, e, \wp,\ c_l, c_u, d, h, v, \mathrm{s}, m_0)$ is an **extended hybrid (functional) Petri net** if

– $(PD, PC, TD, TS, TC, F, G, f, e, \wp, d, h, v, \mathrm{s}, m_0)$ is a conditional hybrid (functional) Petri net,

- $\mathcal{T} \subseteq (PD \times TD \cup PD \times TS \cup PD \times TC \cup PC \times TC \cup PC \times TD \cup PC \times TS)$ is a set of test arcs,
- $\mathcal{I} \subseteq (PD \times TD \cup PD \times TS \cup PD \times TC \cup PC \times TC \cup PC \times TD \cup PC \times TS)$ is a set of inhibitor arcs,
- $\mathcal{R} \subseteq (PD \times TD \cup PD \times TS \cup PD \times TC \cup PC \times TC \cup PC \times TD \cup PC \times TS)$ is a set of read arcs,
- $f\colon (F \cup G \cup \mathcal{T} \cup \mathcal{I}, m, time) \to \{\mathbb{N}_0\colon p_i \in PD\ , \mathbb{R}_{\geq 0}\colon p_i \in PC\}$ is an arc weight function,
- $c_l\colon\ \{PD \to \mathbb{N}_0,\ PC \to \mathbb{R}_{\geq 0}\}$ are the minimum capacities of the places,
- $c_u\colon \{PD \to \mathbb{N}_0,\ PC \to \mathbb{R}_{\geq 0}\}$ are the maximum capacities of the places,

whereby the initial marking m_0 must satisfy the condition

$$c_l(p_i) \leq m_0(p_i) \leq c_u(p_i)\ \ \forall\, p_i \in (PD \cup PC).$$

The activation definition (Definition 4.59) has been modified to integrate capacities, test, and inhibitor arcs while the firing process of Definition 4.61 remains valid without exceptions.

Definition 4.66 (activation extended hybrid (functional) Petri net)

The tuple $(PPD, PC, TD, TS, TC, F, G, \mathcal{T}, \mathcal{I}, \mathcal{R}, f, c_l, c_u, e, \wp, d, h, v, s, m_0)$ is an extended hybrid (functional) Petri net. A discrete/stochastic transition $t_j \in (TD \cup TS)$ is **active** if and only if

$$\forall\, p_i \in P_{in}(t_j) : \begin{cases} m(p_i) - f(p_i \to t_j) \geq c_l(p_i) & if\ (p_i \to t_j) \in F \\ m(p_i) > f\left((p_i \to t_j)_{\mathcal{T}}\right) & if\ (p_i \to t_j)_{\mathcal{T}} \in \mathcal{T} \\ m(p_i) < f\left((p_i \to t_j)_{\mathcal{I}}\right) & if\ (p_i \to t_j)_{\mathcal{I}} \in \mathcal{I}, \end{cases}$$

and

$$\forall\, p_i \in P_{out}(t_i) :\ m(p_i) + f(t_j \to p_i) \leq c_u(p_i)$$

and the condition s_j must be fulfilled.

A continuous transition $t_j \in TC$ is **active** if and only if

$\forall\, p_i \in PC_{in}(t_j)$:

$$\begin{cases} m(p_i) > c_l(p_i) & if\ (p_i \to t_j) \in F \\ \qquad \lor (m(p_i) = c_l(p_i) \land I_i > 0) & \\ m(p_i) > f\left((p_i \to t_j)_{\mathcal{T}}\right) & if\ (p_i \to t_j)_{\mathcal{T}} \in \mathcal{T} \land (p_i \to t_j) \notin F \\ m(p_i) > f\left((p_i \to t_j)_{\mathcal{T}}\right) & if\ (p_i \to t_j)_{\mathcal{T}} \in \mathcal{T} \land (p_i \to t_j) \in F \\ \quad \lor \left(m(p_i) = f\left((p_i \to t_j)_{\mathcal{T}}\right) \land I_i > 0\right) & \\ m(p_i) < f\left((p_i \to t_j)_{\mathcal{I}}\right) & if\ (p_i \to t_j)_{\mathcal{I}} \in \mathcal{I}, \end{cases}$$

and

$$\forall\, p_i \in PC_{out}\left(t_j\right):\ m(p_i) < c_u(p_i) \vee \left(m(p_i) = c_u(p_i) \wedge O_i > 0\right).$$

and

$$\forall\, p_i \in PD_{in}\left(t_j\right): \begin{cases} m(p_i) - f\left(p_i \to t_j\right) \geq c_l(p_i) & if\ \left(p_i \to t_j\right) \in F \\ m(p_i) > f\left(\left(p_i \to t_j\right)_{\mathcal{T}}\right) & if\ \left(p_i \to t_j\right)_{\mathcal{T}} \in \mathcal{T} \\ m(p_i) < f\left(\left(p_i \to t_j\right)_{\mathcal{I}}\right) & if\ \left(p_i \to t_j\right)_{\mathcal{I}} \in \mathcal{I}, \end{cases}$$

and

$$\forall\, p_i \in PD_{out}\left(t_j\right):\ m(p_i) + f\left(p_i \to t_j\right) \leq c_u(p_i)$$

and the condition s_j must be fulfilled. It is **strongly input active** if

$\forall\, p_i \in PC_{in}\left(t_j\right):$

$$\begin{aligned} & m(p_i) > c_l(p_i) && if\ \left(p_i \to t_j\right) \in F \\ \wedge\ & m(p_i) > f\left(\left(p_i \to t_j\right)_{\mathcal{T}}\right) && if\ \left(p_i \to t_j\right)_{\mathcal{T}} \in \mathcal{T} \wedge \left(p_i \to t_j\right) \in F \end{aligned}$$

is also satisfied and otherwise it is **weakly input active**. It is **strongly output active** if

$$\forall\, p_i \in PC_{out}\left(t_j\right):\ m(p_i) < c_u(p_i)$$

is also satisfied and otherwise it is **weakly output active**. If it is strongly input and output active, it is **strongly active**. If it is weakly input and output active, it is **weakly active**. Thereby, $PI_{in}\left(t_j\right)$ ist the set of continuous input places with

$$\begin{aligned} & m(p_i) = c_l(p_i), && if\ \left(p_i \to t_j\right) \in F \\ \vee\ & m(p_i) = f\left(\left(p_i \to t_j\right)_{\mathcal{T}}\right) \wedge\ I_i > 0, && if\ \left(p_i \to t_j\right)_{\mathcal{T}} \in \mathcal{T} \wedge \left(p_i \to t_j\right) \in F\ . \end{aligned}$$

and $PI_{out}\left(t_j\right)$ is the set of continuous output places of t_j with $m(p_i) = c_u(p_i) \wedge O_i > 0$.

Furthermore, the conflict types of Definition 4.60 have been modified and expanded in order to cover and resolve all possible situations which can occur in extended hybrid Petri nets.

Definition 4.67 (actual conflicts and resolutions extended hybrid (functional) Petri net)

The tuple $(PD, PC, TD, TS, TC, F, G, \mathcal{T}, \mathcal{I}, \mathcal{R}, f, c_l, c_u, e, \wp, d, h, v, \mathrm{s}, m_0)$ is an extended hybrid (functional) Petri net:

- **Type-1-(input/output)-conflict**: actual (input/output) conflict according to Definition 4.16 and Definition 4.31 of a place $p_i \in (PD \cup PC)$ and two or more discrete/stochastic transitions $t_j \in (TD \cup TS)$. It can only occur when all transitions t_j become firable exactly at the same time (Remark 4.3, Remark 4.4). This conflict can be resolved either by priorities or probabilities (Definition 4.19 and Definition 4.20).

- **Type-2-(input/output)-conflict*:*** actual (input/output) conflict according to Definition 4.48 of a place $p_i \in PC$ and two or more continuous transitions $t_j \in TC$. It can be resolved by sharing (Definition 4.50).
- **Type-3-(input/output)-conflict*:*** actual conflict of a place $p_i \in (PD \cup PC)$, one or more discrete/stochastic transitions $t_j \in (TD \cup TS)$, and one or more continuous transitions $t_k \in TC$. If there is a conflict between discrete/stochastic and continuous transitions, the discrete/stochastic transitions take priority over the continuous transitions (David and Alla 2005).
- **Type-4 -conflict*:*** actual conflict of a place $p_i \in PD$ and two or more continuous transitions $t_j \in TC$. It is solved by priorities (Definition 4.9).

The conflicts of type 1 to 3 can either involve input or output transitions of a place. The type-4-conflict concerns discrete places connected to continuous transitions. Discrete places are always input and output places, simultaneously (Definition 4.58); hence, no differentiation is necessary. Example 4.5 represents a type-1-(input/output)-conflict and Example 4.21 and Example 4.22 represent type-2-(input/output)-conflicts.

Example 4.28

Figure 4.36 shows three examples of type-3-conflicts in extended hybrid Petri nets. At time 0, transition $T1$ of the left Petri net becomes active and fires continuously. At time 2, the delay of $T2$ is passed and it becomes firable. At this time, $P3$ has an actual output conflict due to its minimum capacity of two tokens and it actually has three tokens. If $T1$ and $T2$ fired simultaneously, they would violate the minimum capacity of $P3$. Hence, the rule of Definition 4.67 has to be applied: $T2$ takes priority over $T1$ and fires. Here a discrete place has a type-3-output-conflict. Discrete places can only have type-3-output-conflicts; type-3-input-conflicts cannot occur. However, continuous places can have type-3-output-conflict as well as type-3-input-conflict.

The right Petri net of Figure 4.36 shows at the top a type-3-output-conflict and at the bottom a type-3-input-conflict. At time 0, transitions $T3$ and $T4$ of the top Petri net fire continuously. At time 1, $m(P4) = 3$ and the delay of $T5$ is passed; $P4$ has an actual conflict. The conflict only occurs when $m(P4) - f(P4 \rightarrow T5) = c_l(P4)$. If $m(P4) - f(P4 \rightarrow T5) < c_l(P4)$ or $m(P4) - f(P4 \rightarrow T5) > c_l(P4)$, there is no conflict. It is solved by the rule of Definition 4.67: $T5$ takes priority over $T4$ and fires. The continuous place $P5$ in the bottom Petri net has a type-3-input-conflict at time 1. At that time, $P5$ has five marks and its maximum capacity is six, i.e. $m(P5) + f(T7 \rightarrow P5) = c_u(P5)$. Here the discrete transition $T7$ takes priority over the continuous transition $T6$.

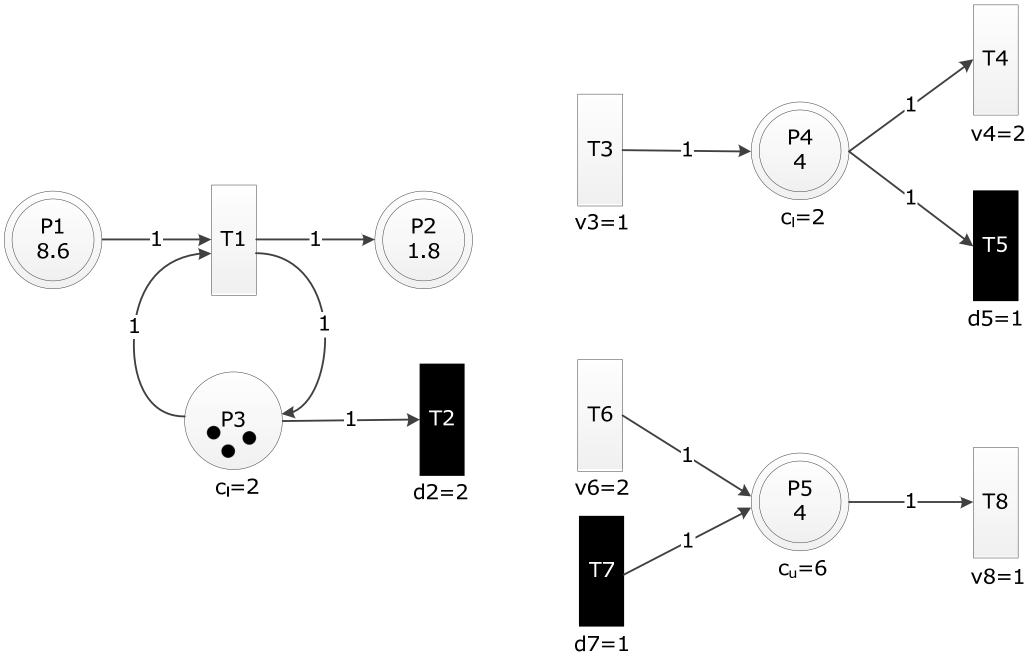


Figure 4.36: Extended hybrid Petri nets with a type-3-conflict of a discrete place (left) and of a continuous place (right), whereby the top Petri net has a type-3-output-conflict and the bottom Petri net has a type-3-input-conflict (Example 4.28)

Example 4.29

Figure 4.37 shows an extended hybrid Petri net with a type-4-conflict of place $P3$.

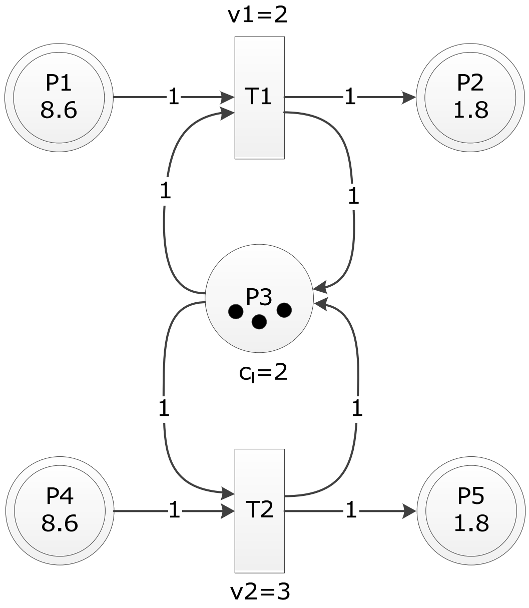


Figure 4.37: Extended hybrid Petri net with a type-4-conflict (Example 4.29)

At time 0, $P3$ can either activate $T1$ or $T2$ but not both simultaneously due to its minimum capacity of two tokens. This conflict can be solved by prioritization of the transitions. If $T1$ takes priority over $T2$, then $T1$ becomes active and fires and if $T2$ takes priority over $T1$, then $T2$ becomes active and fires.

4.5.4 Extended Hybrid Petri Nets

All mentioned variations and extensions presented in this Petri net section are combined in one formalism in order to enable modeling of nearly all kinds of biological processes (see Section 5.2). This formalism is called **xHPN** (e**x**tended **H**ybrid **P**etri **N**et) and the precise definition is given below. The name has been chosen in such a general manner to emphasize that this formalism is useable for biological processes and also for nearly all other processes e.g. production, business, or communication (see Chapter 8). The concrete transformation process of xHPN elements to biological ones is defined in Section 5.2.

Definition 4.68 (xHPN)

The tuple $(PD, PC, TD, TS, TC, F, G, \mathcal{T}, \mathcal{I}, \mathcal{R}, f, c_l, c_u e, \mathscr{p},, d, h, v, s, m_0)$ is a **xHPN** if

- $PD = \{pd_1,\ pd_2, \dots, pd_{pd}\}$ is a finite set of discrete places,
- $PC = \{pc_1,\ pc_2, \dots, pc_{pc}\}$ is a finite set of continuous places,
- $TD = \{td_1,\ td_2, \dots, td_{td}\}$ is a finite set of discrete transitions,
- $TS = \{ts_1,\ ts_2, \dots, ts_{ts}\}$ is a finite set of stochastic transitions,
- $TC = \{tc_1,\ tc_2, \dots, tc_{tc}\}$ is a finite set of continuous transitions,
- $PD, PC, TD, TS,$ and TC are pairwise disjoint,
- $F \subseteq (PD \times TD \cup PD \times TS \cup PD \times TC \cup PC \times TC \cup PC \times TD \cup PC \times TS)$ is a set of arcs from places to transitions,
- $G \subseteq (TD \times PD \cup TD \times PC \cup TS \times PD \cup TS \times PC \cup TC \times PC \cup TC \times PD)$ is set of arcs from transitions to places,
- $\mathcal{T} \subseteq (PD \times TD \cup PD \times TS \cup PD \times TC \cup PC \times TC \cup PC \times TD \cup PC \times TS)$ is a set of test arcs,
- $\mathcal{I} \subseteq (PD \times TD \cup PD \times TS \cup PD \times TC \cup PC \times TC \cup PC \times TD \cup PC \times TS)$ is a set of inhibitor arcs,
- $\mathcal{R} \subseteq (PD \times TD \cup PD \times TS \cup PD \times TC \cup PC \times TC \cup PC \times TD \cup PC \times TS)$ is a set of read arcs,
- $f\colon (F \cup G \cup \mathcal{T} \cup \mathcal{I}, m, time) \rightarrow \{\mathbb{N}_0\colon p_i \in PD, \mathbb{R}_{\geq 0}\colon p_i \in PC\}$ is an arc weight function which assigns every arc connected to a discrete place a non-negative integer and otherwise a non-negative real-valued number depending on a subset of markings m and/or on $time$,

- if $p_i \in PD$ and $t_j \in TC$ then $(p_i \to t_j) \in F$ if and only if $(t_j \to p_i) \in G$ and $f(p_i \to t_j) = f(t_j \to p_i)$,
- $c_l\colon \{PD \to \mathbb{N}_0, PC \to \mathbb{R}_{\geq 0}\}$ are the minimum capacities of the places,
- $c_u\colon \{PD \to \mathbb{N}_0, PC \to \mathbb{R}_{\geq 0}\}$ are the maximum capacities of the places,
- $e\colon (PD \cup PC) \to \{prio, prob\}$ is an enabling function that assigns every place either the resolution type priority or probability,
- $\wp\colon (F \cup G) \to \left\{\{1,2,\dots,max(n_{out}, n_{in})\}\colon e(p_i) = prio \wedge t_j \in TD,\ [0,1]\colon e(p_i) = prob \wedge t_j \in TD\right\}$ is an enabling function which assigns every arc connected to a discrete transition either a priority or a probability according to the resolution type,
- if $e(p_i) = prio$ then $\wp(p_i \to t_k) \neq \wp(p_i \to t_l)\ \forall t_k, t_l \in TD_{out}(p_i)$ and $\wp(t_k \to p_i) \neq \wp(t_l \to p_i)\ \ \forall t_k, t_l \in TD_{in}(p_i)$, if $e(p_i) = prob$ then $\sum_{t_k \in TD_{out}(p_i)} \wp(p_i \to t_k) = 1$ and $\sum_{t_k \in TD_{in}(p_i)} \wp(t_k \to p_i) = 1$,
- $d\colon TD \to \mathbb{R}_{\geq 0}$ is a delay function which assigns every discrete transition a non-negative, real-valued delay,
- $h\colon (TS, m) \to \mathbb{R}_{\geq 0}$ is a hazard function which assigns every stochastic transition a non-negative, real-valued random delay depending on a concrete marking m,
- $v\colon (TC, m, time) \to \mathbb{R}_{\geq 0}$ is a maximum speed function which assigns every continuous transition a non-negative, real-valued maximum speed which depends on a subset of concrete markings m and/or on $time$,
- $s\colon (TD \cup TS \cup TC, \mathcal{E}) \to \{\text{true}, \text{false}\}$ is an condition function which assigns every transition a condition depending on several environmental factors $\mathcal{E}$ e.g. $time$, and
- $m_0\colon \{PD \to \mathbb{N}_0, PC \to \mathbb{R}_{\geq 0}\}$ is the initial marking which must satisfy the condition $c_l(p_i) \leq m_0(p_i) \leq c_u(p_i)\ \ \forall\, p_i \in (PD \cup PC)$.

This xHPN formalism has been transformed to the modeling language Modelica (see Section 3.1) to enable graphical modeling, hybrid simulation, and animation. The concrete transformation process is described in Chapter 6. For the simulation, the following Petri net processes have been used:

- Activation: Definition 4.66
- Firing: Definition 4.61
- Conflict resolution: Definition 4.67,

whereby stochastic transitions are handled as discrete transitions.

5 MODELING PROCESS OF BIOLOGICAL SYSTEMS

This chapter describes the general, universally useable modeling process for biological systems developed in this work which enables the processing of experimental data to usable new insights about the regarded system. This modeling process requires several mathematical methods to achieve a “good working” parameterized model which is able to predict the behavior of the underlying system and forms the basis for optimizing biological processes. Therefore, an environment has been developed which comprises mathematical methods and tools for covering all steps of the established modeling process as depicted in Figure 5.1. The basic concepts of these mathematical methods have been already introduced in Chapter 3 as well as the xHPN modeling concept in Chapter 4 and, hereafter, it is detailed how these methods are particularly adapted to make them useable for biological processes.

The developed modeling process can be divided into the following four phases which are usually rerun several times to establish a reliable model:

1. Preparation
2. Modeling
3. Verification
4. Optimization.

The first phase of the modeling process, called **preparation phase**, is initiated by an observed biological phenomenon which should be investigated. Based on experimental data and prior knowledge about specific structures, functions, and interactions of the investigated system, the phenomenon has to be formulated as hypotheses whose verification is the aim of the modeling process. Furthermore, the available experimental data is preprocessed to remove noise, detect measurement errors, and approximate gaps between two measurements. With the aid of this preprocessed data and non-linear regression methods, relationships of the underlying system are uncovered and analyzed. This procedure is called **preprocessing and relationship analysis** (PRA) and is part of Section 5.1.

The **modeling phase** is initiated by using this knowledge for creating a **mathematical model** (MM) by means of the Modelica language (see Section 3.1). This can either be done textually

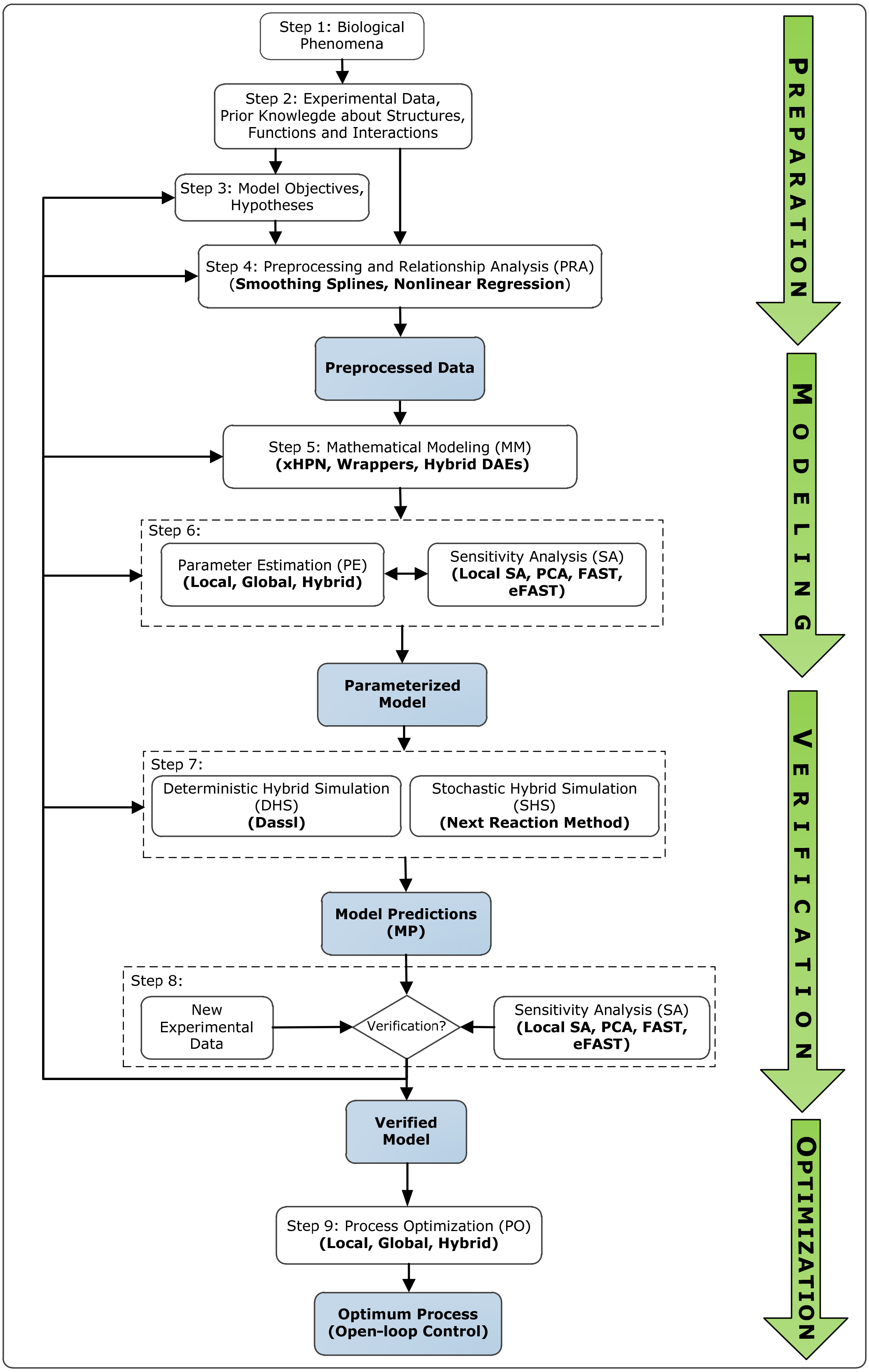

Step 1: Biological Phenomena
Step 2: Experimental Data, Prior Knowlegde about Structures, Functions and Interactions
Step 3: Model Objectives, Hypotheses
Step 4: Preprocessing and Relationship Analysis (PRA) (Smoothing Splines, Nonlinear Regression)
Preprocessed Data
Step 5: Mathematical Modeling (MM) (xHPN, Wrappers, Hybrid DAEs)
Step 6:
Parameter Estimation (PE) (Local, Global, Hybrid)
Sensitivity Analysis (SA) (Local SA, PCA, FAST, eFAST)
Parameterized Model
Step 7:
Deterministic Hybrid Simulation (DHS) (Dassl)
Stochastic Hybrid Simulation (SHS) (Next Reaction Method)
Model Predictions (MP)
Step 8:
New Experimental Data
Verification?
Sensitivity Analysis (SA) (Local SA, PCA, FAST, eFAST)
Verified Model
Step 9: Process Optimization (PO) (Local, Global, Hybrid)
Optimum Process (Open-loop Control)
PREPARATION
MODELING
VERIFICATION
OPTIMIZATION

Figure 5.1: The modeling process: From a biological phenomenon to a verified model which can predict system behavior and forms the basis for process optimization to establish an open-loop control

by a system of hybrid DAEs or graphically with the aid of the xHPN formalism developed in this work to model nearly all kinds of biological reactions. The xHPN formalism is introduced in Section 5.2 based on the terminology and the definitions of Section 4.

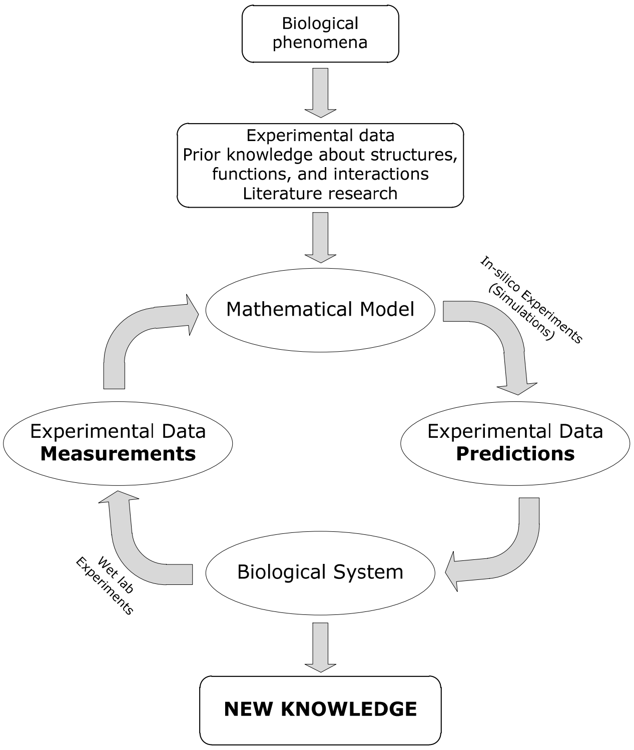

Biological phenomena
Experimental data
Prior knowledge about structures, functions, and interactions
Literature research
Mathematical Model
In-silico Experiments (Simulations)
Experimental Data Measurements
Experimental Data Predictions
Wet lab Experiments
Biological System
NEW KNOWLEDGE

Figure 5.2: The modeling circle (based on (Reiß 2002))

To get further insight into the model and, especially, its parameters, it can be analyzed by means of **parameter estimation** (PE) and **sensitivity analysis** (SA) methods. Thereby, PE aims at adapting the parameters as well as possible to the experimental data with the aid of the optimization methods introduced in Section 3.2; PE is part of Section 5.3. The SA methods introduced in Section 3.3 can be used to reduce the model complexity by revealing unimportant parameters which have little or no influence on the model output and can thus be eliminated from the model. This procedure, also called **model reduction** (MR), can simplify PE due to the fact that the search space of the optimization procedures is reduced. SA for biological processes is described in Section 5.4.

The result of the modeling phase is a **parameterized model** which can be simulated deterministically or stochastically depending on the kind of the Petri net model. Thereby, a **deterministic hybrid simulation** (DHS) can be applied for all kinds of Petri nets while a **stochastic hybrid simulation** (SHS) can only be applied if the Petri net comprises at least one stochastic transition. DHS and SHS are described in Section 5.5.

The next phase of the modeling process is the **verification phase**. A parameterized model can be verified by SA methods which give some indication of the robustness of the found model parameters (see Section 5.4). Additionally, experiments have to be planned based on the results of the parameterized model. This new experimental data gained from wet-lab (**in-vitro** and **in-vivo** experiments) is compared with the **model predictions** (MP) of the system behavior generated by simulations (**in-silico** experiments) to verify the model or to detect a mismatch and return to one of the previous steps (see Figure 5.2). The result of this phase is a **verified model**. MP is part of Section 5.6.

A verified model can be used in the next phase, called **optimization phase,** to optimize one or more underlying processes of the investigated organism. This procedure is called **process optimization** (PO). A PO can be, for example, performed to achieve an **open-loop control** of a fermentation process which maximizes the product yield of the organism. PO is described in Section 5.7.

5.1 Preprocessing and Relationship Analysis

A **relationship analysis** (RA) aims at detecting relationships of the regarded system from experimental data in order to express the discovered relationships in terms of mathematical functions. This is done by nonlinear regression methods. In this way, further knowledge about the structure of the model and the corresponding reaction rates can be obtained.

Therefore, the experimental data has to be **preprocessed** at first due to the fact that the measurements usually contain noise and are only taken at a few points in time. To delete noise and approximate missing data, the experimental data are smoothed. **Smoothing** means that the measurements are approximated by a function that tries to cover all important patterns in the data while no account is taken on noise or other fine-scale structures. Several smoothing methods have been proposed (see e.g. Simonoff 1996). This study concerns the **cubic smoothing spline method** (Reinsch 1967).

A **cubic spline** is a function that interpolates the given data set piecewise by cubic polynomials. However, a **cubic smoothing spline** does not have to go through each given data point; thus, it approximates the data by a function which consists piecewise of cubic polynomials. The cubic smoothing spline f minimizes the sum of weighted squared residuals between the measurement $\hat{y}(t_i)$ and function value $f\,(t_i)$ at time t_i penalized by the roughness measure $\int f''(t)^2\,dt$ which decreases as f get smoother

$$f = arg\,min\left(\sum_{i=1}^{n_t} w_i\big(\hat{y}(t_i) - f\,(t_i)\big)^2 + \lambda\int f''(t)^2\,dt\right). \qquad \text{Eq. 5-1}$$

The smoothing parameter $\lambda \geq 0$ represents the trade-off between accuracy to the measured data and roughness of the function f. If $\lambda = 0$, there is no smoothing and the spline goes through each of the data points $\hat{y}(t_i)$, hence, it is an interpolating cubic spline. If $\lambda \to \infty$, the roughness penalty becomes priority and f converges to the linear least squares regression line. The goodness of a fit can be examined by different statistical coefficients and confidence intervals.

5.2 Mathematical Modeling: xHPN for Biological Applications

The creation of a model for a biological system generally leads to the following advantages (Dunn 2003)

1. **Modeling improves understanding**. The comparison of model predictions and biological behavior leads to an increased understanding of the considered processes. The results of simulations give some indication of the occurrence of observed phenomena that are inexplicable till now. Additionally, the model formulation itself improves the understanding because complex cause-effect sequences and interactions have to be translated into a mathematical formalism.
2. **Modeling supports experimental design**. Experiments have to be designed such that the model can be tested sufficiently. The model itself usually indicates which experimental data are needed to identify the model parameters. Sensitivity analysis can reveal that some parameters have negligible effects on the model and, thus, these effects can be neglected from the model and the experiments while other parameters have a deep impact on the model and, hence, the experiments have to focus on these processes.
3. **Models used for predictions**. Once a model is established and verified, it can be used to predict the behavior of the regarded system under different environmental conditions.

4. **Models used for process optimization**. A verified model can be used for optimizing processes which relate to profits or costs to enable, for example, an open-loop control of these processes.

Numerous model formalisms have been proposed for modeling and simulation of biological systems (see e.g. (Wiechert 2002)). Generally, there needs to be a distinction between qualitative and quantitative approaches. **Qualitative models** represent only the fundamental compounds, their interaction mechanisms, and the relationships amongst them while **quantitative models** describe, in addition, the time-related changes of the components. Hence, a qualitative model is the basis for every quantitative model and the mentioned improved data basis enables us to extend qualitative models to quantitative ones today. Beyond this, quantitative model formalisms can be further divided into discrete, continuous, and hybrid approaches as well as into deterministic and stochastic techniques.

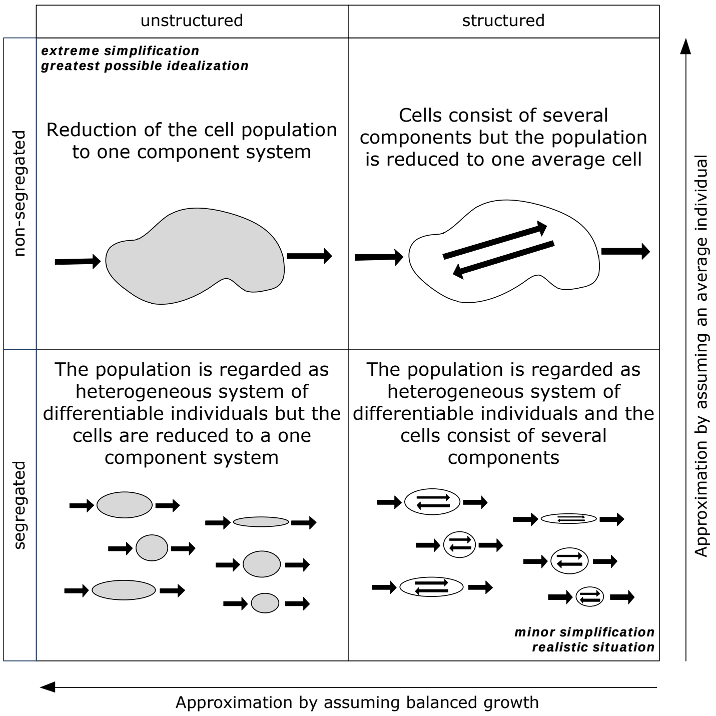

unstructured
structured
non-segregated
segregated
extreme simplification
greatest possible idealization
Reduction of the cell population to one component system
Cells consist of several components but the population is reduced to one average cell
The population is regarded as heterogeneous system of differentiable individuals but the cells are reduced to a one component system
The population is regarded as heterogeneous system of differentiable individuals and the cells consist of several components
minor simplification
realistic situation
Approximation by assuming an average individual
Approximation by assuming balanced growth

Figure 5.3: Different types of cell representation (Chmiel and Briechle 2008, Dunn 2003)

Furthermore, models of biological systems can be classified according to their complexity (Chmiel and Briechle 2008, Dunn 2003). Figure 5.3 gives an overview of the different perspectives to represent a population. On the one hand, models are classified according to the amount of components that are needed to represent the cellular system. If a model consists of several differentiable components, it is called **structured**; otherwise, if the system is represented by one component, it is called **unstructured**. Unstructured models disregard intercellular processes and describe the system only based on changes in its environment. On the other, models can be divided into **segregated models** which regard the system as a heterogeneous collection of differentiable individuals and **non-segregated models** which approximate the system by an average individual. Segregated models consider the heterogeneity of the population concerning cell age, size, growth rate, and physiological state to allow a more precise description of the system.

The decision which modeling approach to use is difficult and strongly influenced by the availability of data. If all kinetic data is known, models consisting of ordinary differential equations are mostly the first choice while in the absence of this kinetic data only qualitative approaches are usable. An additional difficulty arises in the demand of simultaneously having a model which is easy to understand and an abstraction of the real system as well as a detailed and nearly complete description of it. Besides, the modeling process of biological systems is further complicated by incomplete knowledge, noisy and inaccurate data, and different ways of representing data and knowledge.

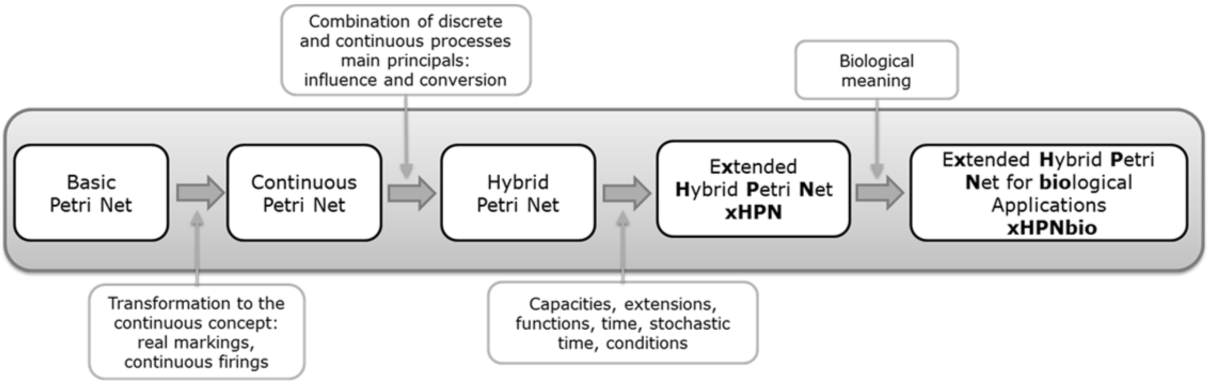

Combination of discrete and continuous processes main principals: influence and conversion
Biological meaning
Basic Petri Net
Continuous Petri Net
Hybrid Petri Net
Extended Hybrid Petri Net xHPN
Extended Hybrid Petri Net for biological Applications xHPNbio
Transformation to the continuous concept: real markings, continuous firings
Capacities, extensions, functions, time, stochastic time, conditions

Figure 5.4: Petri net extensions: From a basic Petri net to an extended hybrid Petri net for biological applications

Petri nets with their various extensions are a universal graphical modeling concept for representing biological systems in nearly all degrees of abstraction. They support both the qualitative and the quantitative modeling approach. Once a qualitative Petri net model has been established, the quantitative data can be added successively. The Petri nets in Section 4.1 and 4.2 are examples for qualitative models due to the fact that no time is associated with the transitions. The arcs can be provided with the stoichiometric of the respective reaction and the tokens represent a discrete quantity of species. The qualitative analysis of such models considers all possible behaviors of the system at any time. Timed, stochastic, continuous, and hybrid Petri nets are examples of quantitative models (see Section 4.3, 4.4, and 4.5). The time is associated with the Petri net behavior by assigning each transition a delay, a hazard, and a maximum speed, respectively. Furthermore, the biological processes can be modeled discretely as well as continuously and, in addition, discrete and continuous processes can also be combined within one Petri net model to so-called hybrid Petri nets (see e.g. (David and Alla 2001)). The Petri net formalism with all its extensions is so powerful that all other formalisms are included and, hence, only one formalism is needed regardless of the approach (qualitative vs. quantitative, discrete vs. continuous, deterministic vs. stochastic) which is appropriate for the respective system. The Petri net formalism is easy to understand for all researchers from different disciplines (biology, mathematics, informatics, and system sciences) which work together in the modeling process and is an ideal way for intuitive representing and communicating experimental data and knowledge of biological systems. Besides, Petri nets allow hierarchical structuring of models and offer the possibility of different detailed views for every observer of the model. For these reasons, the developed xHPN formalism is superior to systems of ordinary differential equations which are mostly the first choice for modeling biological systems.

The Petri net formalism developed and used in this work has been gained from many discussions with biologists and biotechnologist to satisfy all their requirements and to represent with it nearly all kinds of biological reactions and phenomena. The result is called e**x**tended **H**ybrid **P**etri **N**et - abbreviated as **xHPN** and depicted Figure 5.4 - and the precise definition is given in Definition 4.68 (Section 4.5). The abbreviation has been chosen in such a general manner to emphasize that this formalism is not only useable for biological processes but also for nearly all other processes e.g. production, business, or communication (see Chapter 8). This xHPN formalism is extended by providing each Petri net element with a biological meaning. This extension is called **xHPNbio** (e**x**tended **H**ybrid **P**etri **N**ets for **bio**logical applications) and the formal definition is given below.

Definition 5.1 (xHPNbio)

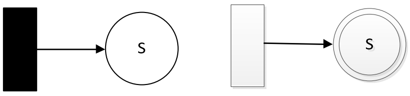
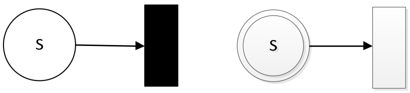
An **xHPNbio** is an xHPN (see Definition 4.68) with a concrete transformation of xHPN elements to biological ones. This transformation is summarized in the following table by mentioning also some examples of the biological meaning.

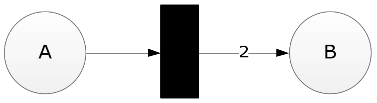
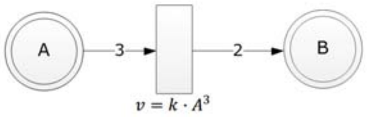
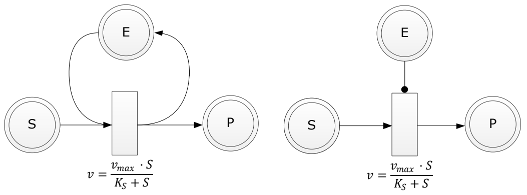
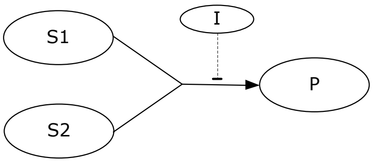
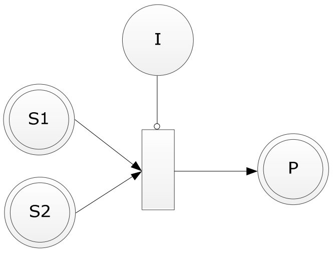
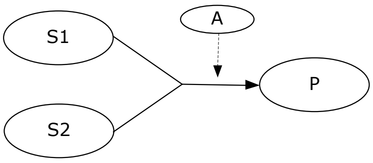
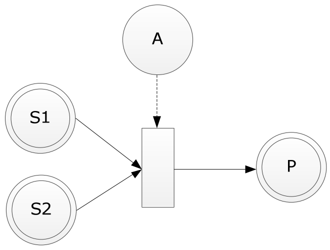
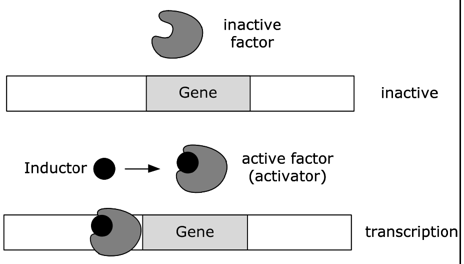
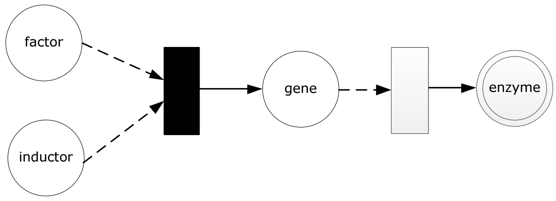
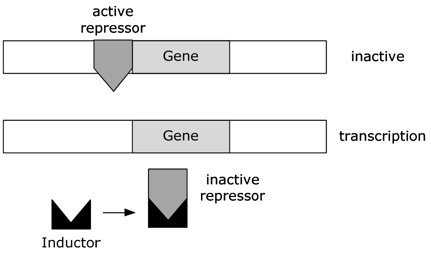
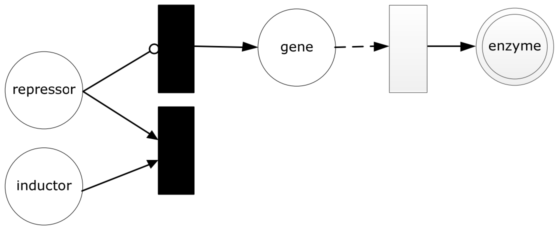
xHPN element	Biological meaning
Places	*Biological compounds*
	metabolites, enzymes, substances, substrates, products, signals, genes, proteins, cells, complexes, activators, inhibitors, repressors, DNA, RNA

Transitions	*Biological processes* biochemical reactions, metabolic reactions, interactions, regulatory reactions, signal transduction reactions, chemical reactions, binding, phosphorylation
Tokens/Marks	*Quantities of biological compounds* molecules, concentrations, cells
Normal arcs	*Connections of biological compounds and processes*
Test arcs	*Activation of biological processes* transcription process, activation in gene regulation, enzyme activity, activation mechanisms
Inhibitor arcs	*Inhibition of biological processes* repression of gene regulation, inhibition mechanisms
Read arcs	*Needs for biological processes* catalysis
Arc weights	*Biological coefficients* stoichiometric coefficients, yield coefficients
Min/max. capacities	*Reasonable biological capacities* biological knowledge
Delays	*Duration of biological processes*
Hazard functions	*Random duration of biological processes* stochastic kinetics
Maximum speeds	*Rate of biological processes* kinetics effects/laws
xHPNbio	*Biological systems* metabolic networks, signal transduction networks, regulatory networks, chemical networks, cell cycle, cell communication, diseases, population dynamics, flux networks, cultivation processes

The following table gives some examples of biological reactions and their modeling with the xHPNbio formalism.

Table 5.1: Examples for modeling biological reactions with the xHPNbio formalism

Type of reaction	**Example**	**Petri net**
Production	→ ***S***	discrete or continuous 
Degradation	***S*** →	discrete or continuous 

Chemical reaction	$A \rightarrow 2B$	
Chemical reaction modeled by mass action kinetics	$3A \overset{v}{\rightarrow} 2B$ $v = k \cdot A^3$	
Biochemical reaction modeled by Michaelis-Menten kinetics	$S + E \overset{v}{\rightarrow} P + E$ $v = \dfrac{v_{max} \cdot S}{K_S + S}$	loop-connection or read arc  E S P $v = \frac{v_{max} \cdot S}{K_S + S}$ E S P $v = \frac{v_{max} \cdot S}{K_S + S}$
Inhibition reaction		
Activation reaction		
Positive gene regulation	 inactive factor Gene inactive Inductor active factor (activator) Gene transcription	
Negative gene regulation	 active repressor Gene inactive Gene transcription inactive repressor Inductor	

5.3 Parameter Estimation

Once a model is constructed using the Modelica language, for example, with the xHPN formalism and the PNlib (see Section 6), the task is to estimate the model parameters. Thereby, the parameters have to be chosen so that the model reproduces the given experimental data in the best possible way. This procedure is called **parameter estimation** or **inverse problem**; thereby, the latter indicates that the model parameters are identified from measurements.

Parameter estimation engenders an optimization problem: Minimize an **objective function** which represents the goodness of a parameter set. This objective function can be formulated mathematically by a non-linear programming problem (NLP) constrained by the hybrid DAEs of the Modelica model (see Section 3.1.7) and upper and lower bounds for every parameter:

$$min\, Q(\mathscr{p}) \quad \mathscr{p} \in \mathcal{P} \subseteq \mathbb{R}^n, Q: \mathcal{P} \to \mathbb{R} \qquad \text{Eq. 5-2}$$

subject to

$$\begin{gathered}\text{Hybrid DAE of the Modelica Model}\\ \text{consisting of a combination of Eq. 3-3, Eq. 3-4, and Eq. 3-5}\end{gathered} \qquad \text{Eq. 5-3}$$

$$\mathscr{p}^l \leq \mathscr{p} \leq \mathscr{p}^u \qquad \text{Eq. 5-4}$$

whereby

- $Q\big(\mathscr{y}(\mathscr{p}, t)\big)$, denoted by $Q(\mathscr{p})$, is the objective function to minimize which depends on a subset of regarded model outputs $\mathscr{y}(\mathscr{p}, t)$ depending on the parameter values $\mathscr{p}$.
- $\mathscr{p} = (\mathscr{p}_1, \mathscr{p}_2, \dots, \mathscr{p}_n)$ are the model parameters to optimize; these are a subset of all parameters p of the Modelica model ($\mathscr{p} \subseteq p$),
- $\mathscr{y}(\mathscr{p}, t) = (\mathscr{y}_1(\mathscr{p}, t), \mathscr{y}_2(\mathscr{p}, t), \dots, \mathscr{y}_{n_y}(\mathscr{p}, t))$ are the regarded model outputs which relate to the measurements $\hat{\mathscr{y}}(t) = \Big(\hat{\mathscr{y}}_1(t), \hat{\mathscr{y}}_2(t), \dots, \hat{\mathscr{y}}_{n_y}(t)\Big)$; these are a subset of state and algebraic output variables of the Modelica model $(\mathscr{y}(\mathscr{p}, \mathrm{t}) \subseteq \big(x(t) \cup y(t)\big)$,
- $\mathscr{p}^l = \big(\mathscr{p}_1^l, \mathscr{p}_2^l, \dots, \mathscr{p}_n^l\big)$ are the lower bounds of the parameters to optimize,
- $\mathscr{p}^u = (\mathscr{p}_1^u, \mathscr{p}_2^u, \dots, \mathscr{p}_n^u)$ are the upper bounds of the parameters to optimize.

The objective function Q measures the goodness of a parameter set. Therefore, this function has to consider the distances between model output and corresponding data points (red lines in Figure 5.5).

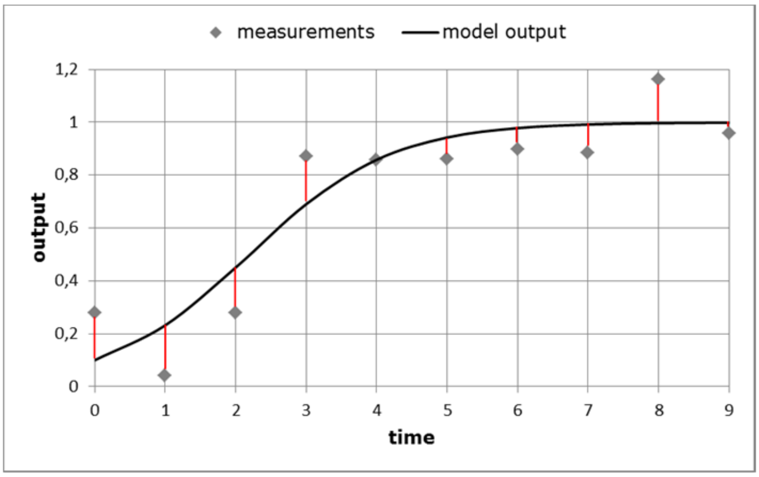


Figure 5.5: Distance between measurements and model output

The **residual matrices** $\mathcal{R}_k(\mathscr{p}), k = 1,2, \ldots, n_d$ are formed by the difference between the **output matrices** $\mathcal{Y}_k(\mathscr{p})$ and **data matrices** $\hat{\mathcal{Y}}_k$; thereby, the columns corresponds to the n_y considered outputs and the rows to the n_t points in time and n_d is the number of considered data sets

$$\mathcal{R}_k(\mathscr{p}) = \mathcal{Y}_k(\mathscr{p}) - \hat{\mathcal{Y}}_k, \qquad k = 1, \ldots, n_d. \qquad \text{Eq. 5-5}$$

The entries, called **residuals**, are given by

$$\bar{\mathscr{r}}_{i,j}^k = \mathscr{y}_{i,j}^k(p) - \hat{\mathscr{y}}_{i,j}^k, \qquad k = 1, \ldots, n_d, i = 1, \ldots, n_t, j = 1, \ldots, n_y \qquad \text{Eq. 5-6}$$

where $\hat{\mathscr{y}}_{i,j}^k = \hat{\mathscr{y}}_j^k(t_i)$ is the measured value of the jth output at time t_i of the kth data set and $\mathscr{y}_{i,j}^k(p) = \mathscr{y}_j^k(\mathscr{p}, t_i)$ is the corresponding model output achieved by the parameter set $\mathscr{p}$.

The residuals can be positive and negative but they should be all positive so that they do not cancel each other out. This can be achieved either by squaring or by calculating the absolute values

$$\mathscr{r}_{i,j}^k = \left(\bar{\mathscr{r}}_{i,j}^k\right)^2, \qquad k = 1, \ldots, n_d, i = 1, \ldots, n_t, j = 1, \ldots, n_y \qquad \text{Eq. 5-7}$$

$$\mathscr{r}_{i,j}^k = \left|\bar{\mathscr{r}}_{i,j}^k\right|, \qquad k = 1, \ldots, n_d, i = 1, \ldots, n_t, j = 1, \ldots, n_y \qquad \text{Eq. 5-8}$$

Afterwards, a one-dimensional result of the objective function can be obtained by either summing up all residuals or by choosing the largest residual

$$Q(\mathscr{p}) = \sum_{k=1}^{n_d} \sum_{i=1}^{n_t} \sum_{j=1}^{n_y} \mathscr{r}_{i,j}^k \qquad \text{Eq. 5-9}$$

$$Q(\mathscr{p}) = max\left\{\mathscr{r}_{i,j}^k, k = 1 \ldots n_d, i = 1 \ldots n_t, j = 1 \ldots n_y\right\} \qquad \text{Eq. 5-10}$$

Additionally, each residual $\mathscr{r}_{i,j}^k$ can be weighted by a factor $\mathscr{w}_{i,j}^k$ to consider, for example, different units or magnitudes. One possible weighting is to divide the residuals by the measured values, i.e. $w_{i,j}^k = \left(1/\hat{y}_{i,j}^k\right)^2$. Summarizing, all the possible combinations lead to the following four objective functions

$$Q_{ss}(p) = \sum_{k=1}^{n_d}\sum_{i=1}^{n_t}\sum_{j=1}^{n_y} w_{i,j}^k\left(y_{i,j}^k(p) - \hat{y}_{i,j}^k\right)^2 \qquad \text{Eq. 5-11}$$

$$Q_{sa}(p) = \sum_{k=1}^{n_d}\sum_{i=1}^{n_t}\sum_{j=1}^{n_y} w_{i,j}^k\left|y_{i,j}^k(p) - \hat{y}_{i,j}^k\right| \qquad \text{Eq. 5-12}$$

$$Q_{ms}(p) = max\left\{w_{i,j}^k\left(y_{i,j}^k(p) - \hat{y}_{i,j}^k\right)^2, k = 1 \dots n_d, i = 1 \dots n_t, j = 1 \dots n_y\right\} \qquad \text{Eq. 5-13}$$

$$Q_{ma}(p) = max\left\{w_{i,j}^k\left|y_{i,j}^k(p) - \hat{y}_{i,j}^k\right|, k = 1 \dots n_d, i = 1 \dots n_t, j = 1 \dots n_y\right\}, \qquad \text{Eq. 5-14}$$

whereby the indices represent the types of the objective functions; the first index can either be s for sum or m for maximum and the second index can either be s for squared or a for absolute.

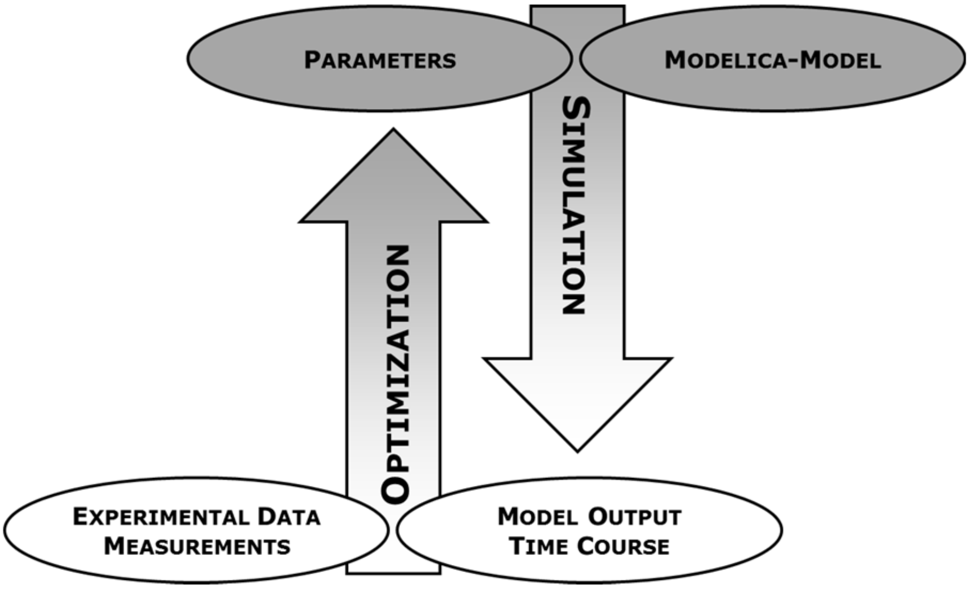


Figure 5.6: The simulation-optimization method for parameter estimation

To find the minimum of the optimization problem in Eq. 5-2 to Eq. 5-4 corresponding to PE, the methods introduced in Section 3.2 have been used. Thereby, the PE process is usually performed using the following steps:

1. Set the model parameters.
2. Simulate the model.
3. Calculate the value of the objective function.
4. Check the abort criteria. If one is fulfilled, stop; otherwise, adjust the model parameters according to the used optimization method of Section 3.2, and go back to step 1.

This procedure requires a combination of optimization routines and simulations, a so-called **simulation-optimization method**, which is displayed in Figure 5.6.

5.4 SENSITIVITY ANALYSIS

A SA can be performed for a number of reasons (see Chan et al. 1997 and Saltelli et al. 2000). This study concentrates on using it for the following aims (see Figure 5.1):

- **Model identification**: determine the model parameters which contribute most to the variability of the model output. These parameters need additional research to get further knowledge about them to reduce the uncertainty in the model output.
- **Model optimization**: determine optimal regions within the parameter space to use them in a subsequent PE process (see Section 5.3).
- **Model reduction**: determine the model parameters which are insignificant to eliminate them from the model. In this manner the subsequent PE can be improved: Fixing less sensitive parameters during the optimization process of the minimization problem in Eq. 5-2 can speed up the convergence rate of the applied method, significantly.
- **Model verification**: determine the robustness of the model parameters after PE to give some indication of the validity of the model toward the reality.

Thereby, local methods are applicable for model verification if it is assumed that the objective function is continuous differentiable in a neighborhood of the found parameter set. On the other hand, the global methods of Section 3.3 are ideal to carry out the impact of each parameter for model identification, optimization, and reduction.

5.5 DETERMINISTIC AND STOCHASTIC HYBRID SIMULATION

The simulation of a **deterministic model**, i.e. no probabilistic behavior is involved, always returns the same results by executing the model with the same input parameters. However, a

stochastic model contains random mechanisms and, hence, the simulation results can change in any run performed with the same input parameters.

Hereafter, the considered stochastic models are stochastic Petri nets (SPNs) already introduced in Section 4.3. The implementation of the stochastic transition by means of the Modelica language is part of Section 6.1.3. The concept is also applicable to an xHPNbio if it contains at least one stochastic transition.

Each transition of a SPN represents a reaction of a biological process and the places are the biological compounds which interact with each other. The tokens quantify the amount of each compound, for example, the number of molecules and the arc weights represent the number of units which are consumed and produced, respectively, in a reaction, for example, the stoichiometric coefficients of a biochemical reaction (see Definition 5.1).

The time when a reaction occurs is a random quantity defined by the hazard function (see Definition 4.34)

$$\tau_j = time + Exp(h_j), \qquad \text{Eq. 5-15}$$

whereby τ_j is the putative firing time of transition t_j which models the reaction, and $Exp(h_j)$ is an exponentially distributed random variable specified by the hazard function h_j. This hazard function can be defined more precisely if a SPN represents a biochemical network. Two possible hazard functions can be applied which depend on reading the tokens as molecules or as levels of concentrations (Heiner et al. 2008).

The first, called **stochastic mass action hazard function**, is directly derived from the mass action kinetics of continuous reactions (Wilkinson 2006). The mass action kinetics has to be modified such that the substances can be represented discretely by molecules instead of continuous concentrations. This general modification process is shown exemplarily by the following types of biochemical reactions.

A biochemical reaction of first-order has the form

$$R\colon\ A \overset{c}{\rightarrow} B,$$

whereby the constant c represents the hazard that a molecule of A will undergo the reaction and there are a molecules of substance A. This leads to the hazard function

$$h_R = c \cdot a. \qquad \text{Eq. 5-16}$$

A second-order biochemical reaction can have the form

$$R:\ A + B \overset{c}{\rightarrow} C,$$

whereby the constant c represents the hazard that a molecule of A and a molecule of B react with each other and there are a molecules of A and b molecules of B. Hence, $a \cdot b$ different pairs are possible so that the hazard function is given by

$$h_R = c \cdot a \cdot b. \qquad \text{Eq. 5-17}$$

It is also possible that a second-order biochemical reaction has the following form

$$R:\ 2A \overset{c}{\rightarrow} B,$$

whereby the constant c represents the hazard that two molecules of substance A react. But only $a(a-1)/2$ pairs are possible so that the hazard function is given by

$$h_R = c \cdot a \cdot \left(\frac{a-1}{2}\right). \qquad \text{Eq. 5-18}$$

This theory can also be applied to higher-order reactions and results in the following general form of the stochastic mass action hazard function regarding that the reaction is modeled by a SPN

$$h_j = c_j \cdot \prod_{p_i \in P_{in}(t_j)} \binom{m(p_i)}{f(p_i \rightarrow t_j)}, \qquad \text{Eq. 5-19}$$

whereby c_j is the transition-specific stochastic rate constant, $m(p_i)$ is the token number of the input place p_i of transition t_j, and $f(p_i \rightarrow t_j)$ is the weight of the arc $(p_i \rightarrow t_j)$.

The second one, called **stochastic level hazard functions**, regards the tokens as levels of concentrations as introduced in (Calder et al. 2006). The concentration of each substance is then transformed to an abstract level. Therefore, it is assumed that the maximum molar concentration is M and the number of the highest level is N so that the amount of levels is $N+1$. Then the abstract levels $0, 1, \dots, N$ represent the concentration intervals

$$0, \left(0, 1 \cdot \frac{M}{N}\right], \left(1 \cdot \frac{M}{N}, 2 \cdot \frac{M}{N}\right], \dots, \left((N-1) \cdot \frac{M}{N}, N \cdot \frac{M}{N}\right].$$

The maximum molar concentration M is always the same for all places in a SPN while the amount of levels $N+1$ can be different for each place. Hence, both parameters M and N can be determined globally in the settings component (see Section 6.2) so that they common to all place components and the parameter N can also be defined locally in the place components (see Section 6.1.2). For the following explanations, it is assumed that the parameter N is defined globally, i.e. it is the same for all places in the SPN.

The derivation of the stochastic level hazard function is shown exemplarily by the following reaction

$$R\colon\ 2A + 3B \overset{k}{\rightarrow} C,$$

where k is the deterministic rate constant. The increase of substance C can be expressed by mass action kinetics

$$\frac{d[C]}{dt} = k[A]^2[B]^3,$$

where $[A]$, $[B]$, and $[C]$ denote the concentrations of substances A, B, and C. The new concentration $[\widetilde{C}]$ can be derived from the current concentration $[C]$ by Euler's method

$$[\widetilde{C}] = [C] + \Delta t \cdot (k[A]^2[B]^3)$$

$$\Leftrightarrow\ \Delta t = \frac{\Delta C}{k[A]^2[B]^3}, \qquad \text{with } \Delta C = [\widetilde{C}] - [C].$$

But the abstract concentrations can only increase by one level, i.e. by a concentration of M/N, when the reaction occurs, hence,

$$\Delta t = \frac{\Delta C}{k[A]^2[B]^3} = \frac{M}{Nk[A]^2[B]^3},$$

whereby Δt is the time that is needed to increase the concentration of substance C by one concentration level M/N. This time is taken as expected value for the occurrence of the reaction and, thus, the hazard function is given by

$$h_R = \frac{1}{\Delta t} = \frac{N}{M}k[A]^2[B]^3 = \frac{N}{M}k\left(\frac{M}{N}\cdot A^d\right)^2\left(\frac{M}{N}\cdot B^d\right)^3,$$

whereby A^d and B^d are the discrete levels which correspond to the concentrations $[A]$ and $[B]$ and, hence,

$$[A] = \frac{M}{N}\cdot A^d \text{ and } [B] = \frac{M}{N}\cdot B^d.$$

The general stochastic level hazard function is then given by

$$h_j = k_j \cdot \frac{N}{M}\prod_{p_i \in P_{in}(t)}\left(\frac{M}{N}\cdot m(p_i)\right)^{f(p_i \to t_j)}, \qquad \text{Eq. 5-20}$$

where k_j is the transition-specific deterministic rate constant. The conversion from the deterministic rate constant k_j to the stochastic rate constant c_j can be found in (Wilkinson 2006).

Several methods have been proposed to perform the stochastic simulation of a SPN model either with stochastic mass action hazard functions or stochastic level hazard functions. The first, called **Gillespie's direct method**, calculates explicitly which reaction occurs next and when it occurs (Gillespie 1977). Thereby, reaction R_k occurs with a probability $h_k/\sum_j h_j$ and the time to the next reaction is the random quantity $Exp\left(\sum_j h_j\right)$. The algorithm is summarized in Algorithm A7 (Appendix A1). This algorithm is direct in the sense that it chooses the reaction R_k and the time when it is occurs τ directly by generating two random numbers at each iteration.

Gillespie also developed an alternative called **first reaction method,** which gets along with one random number per iteration. Thereby, a **putative time** τ_j is generated for each reaction. This is the time when the reaction would occur if no other reaction occured before. Thus, the reaction with the smallest putative time will occur. The method is formalized by Algorithm A8 (Appendix A1).

Gibson and Bruck modified the first reaction method to make it much more effective (Gibson and Bruck 2000). It is called **next reaction method**. They reduced the number of required random numbers by reusing the τ_j. A new putative time is only generated for the reaction that has just occurred. If the hazard function is not changed by this reaction, the τ_j remains unaltered. However, if the hazard function is affected by this reaction, the τ_j is rescaled appropriately. The knowledge about which hazard function is affected by which reaction is attained from a dependency graph. The method can be performed by the following steps.

Algorithm 1

```
1. Set the initial numbers of molecules, set $t = 0$, calculate the hazard function $h_j$, and
   generate a putative time $\tau_j \sim Exp\left(h_j\right)$ for all $j$.
2. Let $R_k$ the reaction whose putative time $\tau_k$ is the smallest.
3. Update the number of molecules affected by reaction $R_k$.
4. Update $h_k$ and generate a new putative time $\tau_k = t + Exp(h_k)$.
5. For each reaction $j \neq k$ whose hazard function is affected by reaction $R_k$:
   a. Set $h_j^{old} = h_j$,
   b. Update $h_j$,
   c. Set $\tau_j = t + \frac{h_j^{old}}{h_j}\left(\tau_j - t\right)$.
6. If $t = T_{max}$ stop; otherwise, set $t = \tau_k$ and go to step 2.
```

The next reaction method is used within this study because the dependency graph is already created by the Modelica-tool to perform the hybrid simulation (see Section 3.1.7). In this manner, the hazard functions are only recalculated when they are changed by firing other transitions. The implementation by the Modelica-language is explained in Section 6.1.3.

5.6 Model Prediction

Once a model is established and the structure parameters are estimated, it can be used to predict the behavior of the system by varying input factors such as substrates, temperature, and stirrer speed. By comparing these predictions with new experimental data, the model can be verified and successively improved (see Figure 5.1). In addition, predictions can expand the knowledge about the studied organism by saving a lot of experiments with different input factors. Usually, saving experiments is equivalent to saving costs which this is an important factor for industry.

5.7 Process Optimization

Moreover, if the model is verified by further experiments and mathematical methods (SA), the modeled processes of the underlying organism can be optimized. Therefore, an objective function has to be defined which depends on several process parameters, e.g. substrates, feeding strategies, temperature, and stirrer speed. Usually, PO aims at maximizing the amount of one or more model outputs at a specific point in time t_f subject to the Modelica model and lower and upper bounds for each process parameter. This can be formalized by the following optimization problem

$$max\, Q(z, t_f) \quad z \in \mathcal{Z} \subseteq \mathbb{R}^n, Q\colon \mathcal{Z} \to \mathbb{R} \tag{Eq. 5-21}$$

subject to

$$\begin{gathered}\text{Hybrid DAE of the Modelica Model}\\ \text{consisting of a combination of Eq. 3-3, Eq. 3-4, and Eq. 3-5}\end{gathered} \tag{Eq. 5-22}$$

$$z^l \leq z \leq z^u, \tag{Eq. 5-23}$$

whereby

- $Q\left(\mathcal{y}\left(z,t_f\right),t_f\right)$, denoted by $Q\left(z,t_f\right)$, is the objective function to maximize which depends on a subset of regarded model outputs $\mathcal{y}\left(z,t_f\right)$ depending on the parameter values z,
- $z=(z_1,z_2,\dots,z_m)$ are a subset of the model parameters ($z \subseteq p$), called **process parameters**, which are varied to maximize the model output,
- $\mathcal{y}\left(z,t_f\right)=\left(\mathcal{y}_1\left(z,t_f\right),\mathcal{y}_2\left(z,t_f\right),\dots,\mathcal{y}_{n_y}\left(z,t_f\right)\right)$ are model outputs to optimize which are a subset of state and algebraic output variables $\left(\mathcal{y}(z,\mathrm{t}) \subseteq \left(x(t) \cup y(t)\right)\right)$,
- t_f is the point in time when the model output is regarded. This time can also be included in the objective function as additional process parameter varied during optimization because the time needed to produce the amount of output correlates usually with its cost,
- $z^l=(z_1^l,z_2^l,\dots,z_n^l)$ are the lower bounds of the process parameters, and
- $z^u=(z_1^u,z_2^u,\dots,z_n^u)$ are the upper bounds of the process parameters.

The following objective functions are usable to maximize one model output by varying several process parameters

$$max\,\mathcal{y}\left(z,t_f^{fix}\right) \qquad z \in \mathcal{Z} \subseteq \mathbb{R}^n \tag{Eq. 5-24}$$

$$max\left(\frac{\mathcal{y}\left(z,t_f\right)}{t_f}\right) \qquad z \in \mathcal{Z} \subseteq \mathbb{R}^n \tag{Eq. 5-25}$$

Thereby, the first one regards only the model output at a fixed point in time t_f^{fix} while the second one includes also the point in time when the maximum model output is reached due to the fact that time is mostly correlated with costs of the regarded process; hence, it tries to find the best compromise of model output and corresponding costs. The arising optimization problem is solved with the methods introduced in Section 3.2.

A PO can be used, for example, to achieve an **open-loop control** of biological processes, i.e. the process parameters are calculated based on the model representing the state of the biological system; no feedback is used to determine if the output has yielded the intended purpose of the adapted process parameters. An open-loop control is an important aspect for industrial fermentation processes: achieving a maximum product yield at minimum costs.

6 The Petri Net Library

This chapter concentrates on the implementation of **extended hybrid Petri nets** (**xHPN**) introduced in Section 4 (see also Section 5.2) in order to model, simulate, and animate these special kinds of Petri nets.

To achieve this aim, an appropriate programming language is needed that is capable of modeling places, transitions, and arcs as objects and meets the following requirements:

- freely available,
- object-oriented,
- equation-based, i.e. support of modeling by discrete, differential, and algebraic equations,
- support of graphical modeling,
- support of hierarchical modeling,
- support of simulation,
- support of animation.

A language that satisfies all these conditions is the non-proprietary, object-oriented, equation-based language **Modelica** (Modelica Association 2011). It is developed and promoted by the Modelica Association since 1996 for modeling, simulation, and programming primarily of physical and technical systems and processes (Modelica Association 2010). Additionally, the Modelica standard library is available from the Modelica Association to model mechanical (1D/3D), electrical (analog, digital, machines), thermal, fluid, control systems, and hierarchical state machines. Furthermore, several libraries have been developed in the last decade for specific applications. An overview can be found on the Modelica homepage (Modelica Association 2011). The development of the language and libraries is ongoing and driven by several European projects (EUROSYSLIB, MODELISAR, OPENPROD, and MODRIO). Since the year 2000, Modelica is used successfully in the industry which is documented in the proceedings of many Modelica conferences and journals.

The Modelica models are described on the textual level by discrete, differential, and algebraic equations and by schematics on the graphical level. A schematic consists of connected **components** which are defined by other components or on the lowest level by equations in the Modelica syntax. The components have connectors which describe the interaction between them. By drawing a line from one component to another, a connection is established to enable interactions. In this manner a model is constructed. Several components can be structured in libraries, called **packages**, which provide hierarchical modeling. Moreover, the **wrapping technique** enables the representation of sub-models consisting of several connected components by a specific adapted icon in order to simplify the modeling process. Then, the sub-models can be used several times in the same or in different models and, in addition, it offers an easy-to-use-model at the top level with an intuitive and familiar adapted view.

For graphical modeling, simulation, and animation an appropriated environment is needed. The following lists some examples of free and commercial Modelica environments and a full list can be found on the Modelica homepage (Modelica Association 2011).

Commercial Modelica environments

– Dymola from Dassault Systemes (www.3ds.com/products/catia/portfolio/dymola)
– MapleSim^{TM} from Maplesoft (www.maplesoft.com/products/maplesim/index.aspx)
– MathModelica System Designer from Wolfram MathCore (www.mathcore.com/products/mathmodelica)

Free Modelica environments

– OpenModelica developed and supported by Linköping University and the Open Source Modelica Consortium (www.openModelica.org)
– JModelica.org maintained and developed by Modelon AB in collaboration with academia (www.jmodelica.org)

Within this study, the commercial tool Dymola – Dynamic Modeling Laboratory – version 7.4 is used.

An introduction to the basic Modelica language constructs and principles which also includes the balanced modeling concept, the hybrid modeling technique, and the description of the process from a Modelica model to executable simulation code has already been given in Section 3.1. The next sections outline the implementation of the Petri net component models by the Modelica language which also comprises an introduction to the Dymola tool and a description of the connection between Dymola and Matlab/Simulink for post-processing of simulation results.

6.1 IMPLEMENTATION OF THE PETRI NET ELEMENTS

The advanced **P**etri **N**et **lib**rary, called **PNlib**, developed in this work enables the modeling of xHPNs. Based on the Petri nets concepts of Section 4 and 5.2, places, transitions, and arcs are modeled as objects by discrete, differential, and algebraic equations in Modelica. Therefore, it has to be distinguished between discrete (`PD`) and continuous places (`PC`), discrete (`TD`), stochastic (`TS`), and continuous transitions (`TC`), and normal, test (`TA`), inhibitor (`IA`), and read arcs (`RA`). Each of them, except the normal arc, is modeled by its own Modelica model and for all of them the specialized class `model` is used (see Section 3.1.4). These Petri net component models are organized and structured in a Modelica `package`, also called **library**. The structure of the PNlib is shown in Figure 6.1. The main package `PNlib` is divided into the following sub-packages

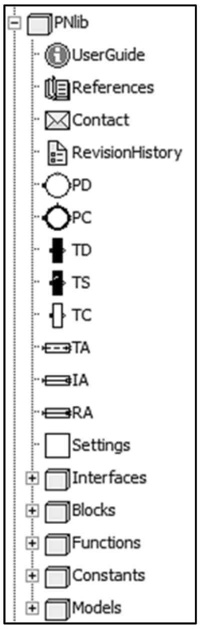
– `Interfaces`: contains the connectors of the Petri net component models.

– `Blocks`: contains blocks with specific procedures that are used in the Petri net component models.

– `Functions`: contains functions with specific algorithmic procedures which are used in the Petri net component models.

– `Constants`: contains constants which are used in the Petri net component models.

– `Models`: contains several examples and offers the possibility to structure further Petri net models.



PNlib
UserGuide
References
Contact
RevisionHistory
PD
PC
TD
TS
TC
TA
IA
RA
Settings
Interfaces
Blocks
Functions
Constants
Models

Figure 6.1: Structure of the PNlib

Additionally, the package contains the component `settings` which enables the setting of global parameters for the display and animation of a Petri net model (see Section 6.2).

Places, transitions, and arcs are represented by the icons depicted in Figure 6.2. Thereby, the discrete place is represented by a circle and the continuous place by a double circle. The transitions are boxes which are black for discrete transitions, black with a white triangle for stochastic transitions, and white for continuous transitions. The test arc is represented by a dashed arc, the inhibitor arc by an arc with a white circle at its end, and the read arc by an arc with a black square at its end.

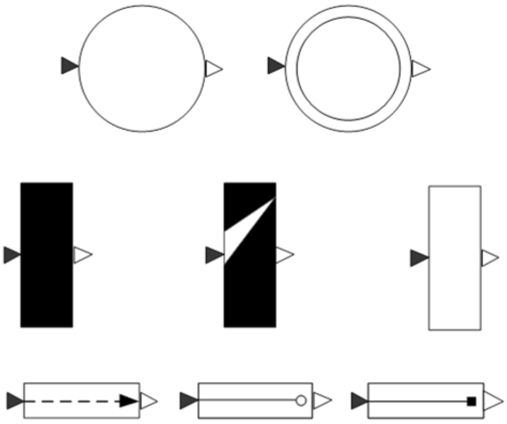


Figure 6.2: Icons of the PNlib: discrete place (circle), continuous place (double circle), discrete transition (black box), stochastic transition (black box with white triangle), continuous transition (white box), test arc (dashed arc), inhibitor arc (arc with circle ending), and read arc (arc with square ending)

Hereafter it is described what is calculated in which Petri net component and at which time. Thereby, the **discrete procedure** - the involved transition is discrete or stochastic - is outlined in Figure 6.3 and the **continuous procedure** - the involved transition is continuous – is displayed in Figure 6.4.

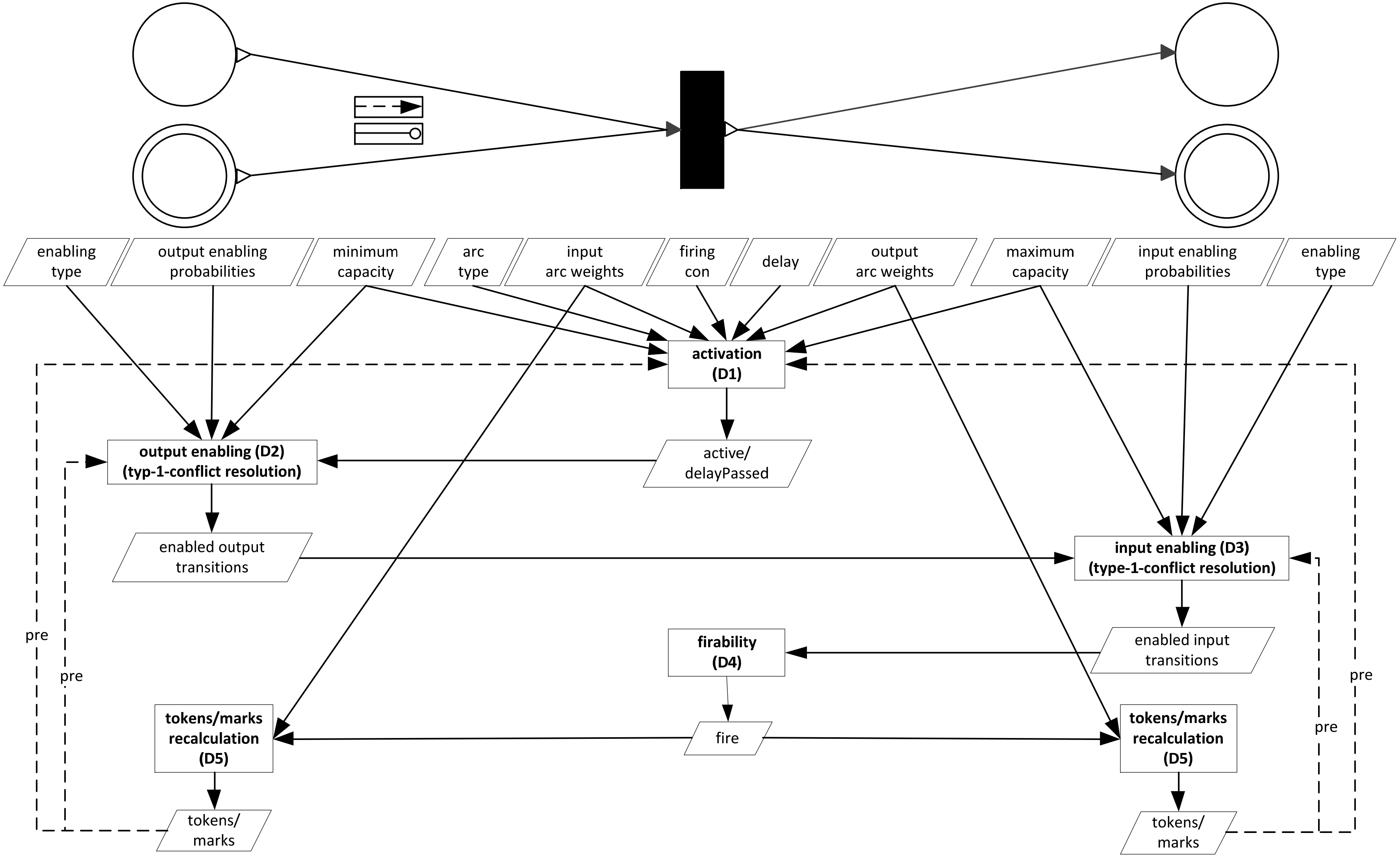

enabling type
output enabling probabilities
minimum capacity
arc type
input arc weights
firing con
delay
output arc weights
maximum capacity
input enabling probabilities
enabling type
activation (D1)
active/ delayPassed
output enabling (D2) (typ-1-conflict resolution)
enabled output transitions
input enabling (D3) (type-1-conflict resolution)
enabled input transitions
firability (D4)
fire
tokens/marks recalculation (D5)
tokens/ marks
tokens/marks recalculation (D5)
tokens/ marks
pre
pre
pre
pre

Figure 6.3: Discrete procedure: what is calculated in which component and at which time if the involved transition is discrete or stochastic. Dashed lines indicate processes that use the predecessor value of the respective variable

The discrete procedure comprises the following steps (the component model in which the step is performed is stated in brackets after the name of the step):

D1 **Activation** (transition): It is checked by means of Definition 4.66 if a discrete transition can become active. When the transition becomes active, the putative firing time $(time + delay)$ is saved.

D2 **Output Enabling** (place): A place enables its output transitions based on which of them are active and if the delays are already passed. Thereby, an occurring type-1-output-conflict (Definition 4.67) is either solved deterministically (Definition 4.19) or probabilistically (Definition 4.20).

D3 **Input Enabling** (place): A place enables its input transitions based on which transitions are already enabled by all input places. Thereby, an occurring type-1-input-conflict (Definition 4.67) is either solved deterministically (Definition 4.19) or probabilistically (Definition 4.20).

D4 **Firability** (transition): If a transition is enabled by all output places, which also implies the enabling by all input places, it is firable and fires immediately.

D5 **Token/marks recalculation** (place): A place recalculates the tokens and marks, respectively, according to Definition 4.61 when one or more connected transitions fire.

This discrete procedure causes an algebraic loop which involves discrete-time variables, hence, it has to be cut by hand to obtain a model that can be translated and simulated (see Section 3.1.7). This is done by putting the `pre`-operator around the discrete-time variable `t` which contains the current token number of a place. In this way, the steps activation, input, and output enabling are based on the predecessor value of `t`, represented in Figure 6.3 by dashed lines, and the outlined ordering of the processes (D1 - D5) is achieved.

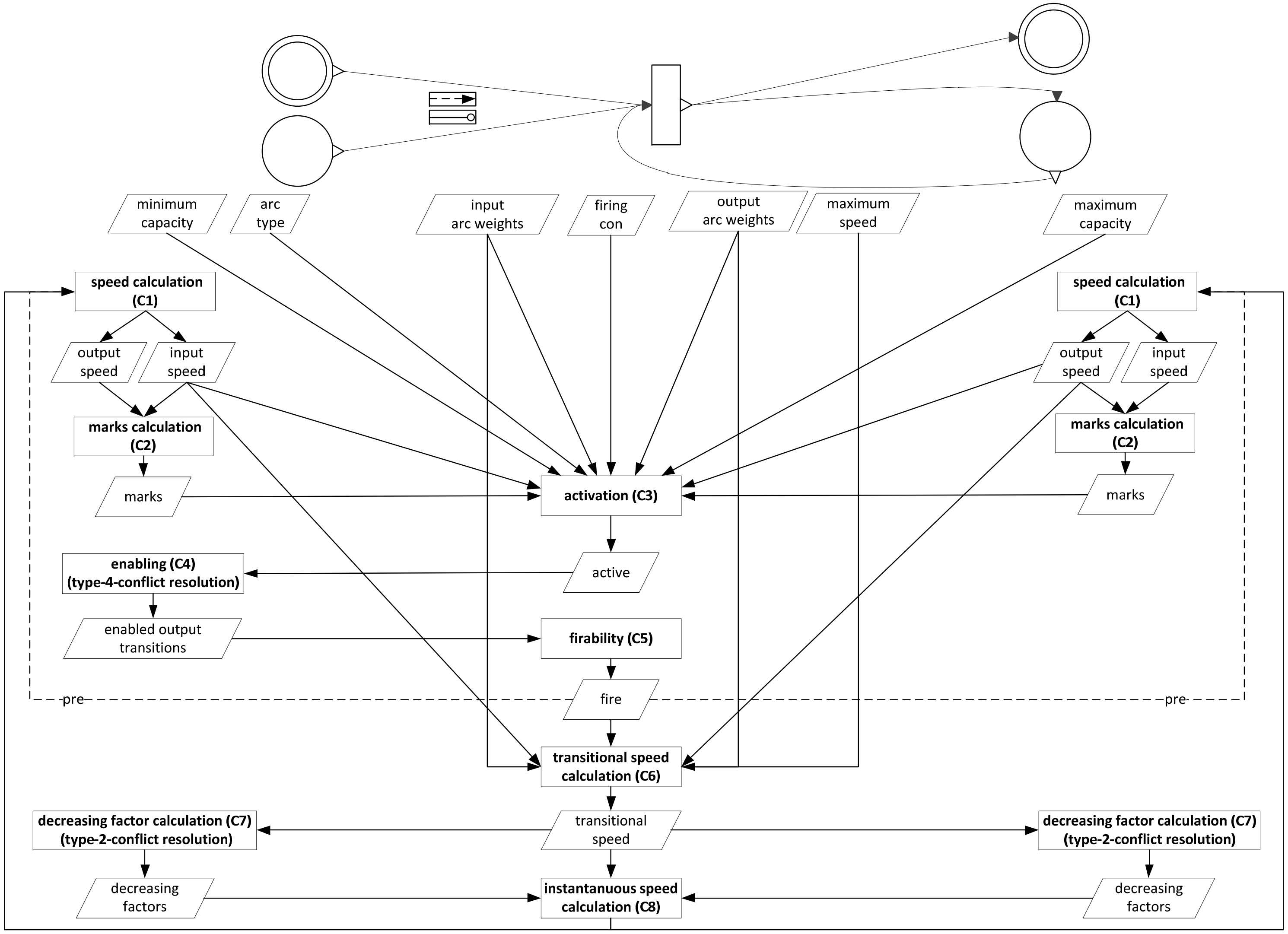


Figure 6.4: Continuous procedure: what is calculated in which component and at which time if the involved transition is continuous. Dashed lines indicate processes that use the predecessor value of the respective variable.

The continuous procedure is performed by the following steps; thereby, it has to be pointed out that tokens of discrete places are never changed by firing continuous transitions. Discrete places can only influence the time when continuous transitions can become active either by connections via test or inhibitor arcs or by loop-connections, i.e. the place is input and output of the transition, simultaneously, with arcs of the same weight (see Section 4.5).

C1 **Speed calculation** (place): The current input and output speed of a continuous place is calculated by means of Definition 4.39.

C2 **Marks calculation** (place): The marking of a continuous place is calculated by means of Definition 4.61.

C3 **Activation** (transition): It is checked with the aid of Definition 4.66 if a continuous transition can become active.

C4 **Enabling** (place): A discrete place enables its continuous output transitions. If a type-4-conflict (Definition 4.67) occurs, it is resolved deterministically by priorities.

C5 **Firability** (transition): A continuous transition is firable if it is active and enabled by all discrete input places.

C6 **Preliminary speed calculation** (transition): The preliminary speed is calculated based on the input speeds of the input places and the output speeds of the output places (Definition 4.48).

C7 **Decreasing factor calculation** (place): The decreasing factor is calculated to resolve type-2-conflicts (Definition 4.67) by sharing the speeds proportional to the maximum speeds of the involved transitions (Definition 4.50).

C8 **Instantaneous speed calculation** (transition): Based on the decreasing factors of the input and output places, the instantaneous speed of a continuous transition can be calculated by means of Definition 4.50.

To realize the steps C1 – C8 in the given order, an algebraic loop which involves discrete-time variables has to be cut by hand. This is done by using the `pre`-operator for the variable `fire` so that the speed calculation (C1) bases on the predecessor value of it. Figure 6.4 indicates this with dashed lines. Additionally, an algebraic loop concerning continuous-time variables may arise by calculating the preliminary and instantaneous speed as well as the decreasing factor. This algebraic loop is solved by the Modelica-tool (see Section 3.1.7).

6.1.1 CONNECTORS

The processes of Figure 6.3 and Figure 6.4 can only be realized if the Petri net components are able to interchange variables. Modelica makes this interaction possible by means of the specialized class `connector` (see Section 3.1.4). A connector comprises variables which are calculated in one component but also needed in the connected components for further calculations. The red and white triangles at the Petri net icons in Figure 6.5 represent the connectors of the Petri net component models. The PNlib contains four different connectors: `PlaceOut`, `PlaceIn`, `TransitionOut`, and `TransitionIn`. The connectors `PlaceOut` and `PlaceIn` are part of the place model and connect them to output and input transitions, respectively. Similarly, `TransitionOut` and `TransitionIn` are connectors of the transition model and connect them to output and input places, respectively. Figure 6.5 shows which connector belongs to which Petri net component model.

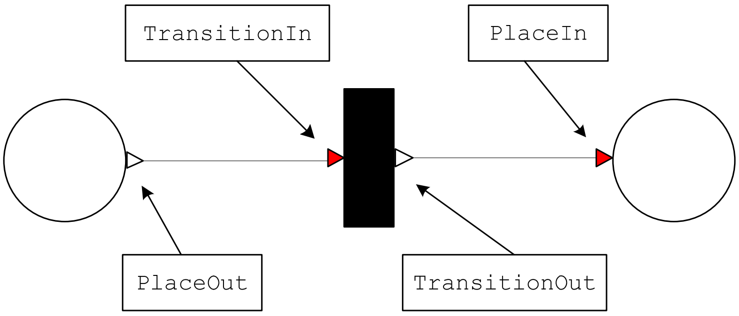


Figure 6.5: Connectors of the PNlib which enable the Petri net components to interact with each other

The connector variables are summarized in Appendix A2. All connector variables are provided either with the prefix `input` or `output` to guarantee balanced models (see Section 3.1.5). If the equation of a connector variable is available in the model where the connector is used, it is prefixed with `output` while it is provided with `input` if the corresponding equation is in a connected component.

Figure 6.6 shows two examples of connector variables. The token number `t` is calculated in the place model but also needed in the connected input and output transitions to determine if they can become active. Hence, it is an output of the place connectors `PlaceOut` and

PlaceIn and an input of the transition connectors TransitionOut and TransitionIn. On the other hand, the variable fire is determined in the transitions but also needed in the connected places for the recalculation of the token number. Hence, it is an output of the transition connectors and an input of the place connectors.

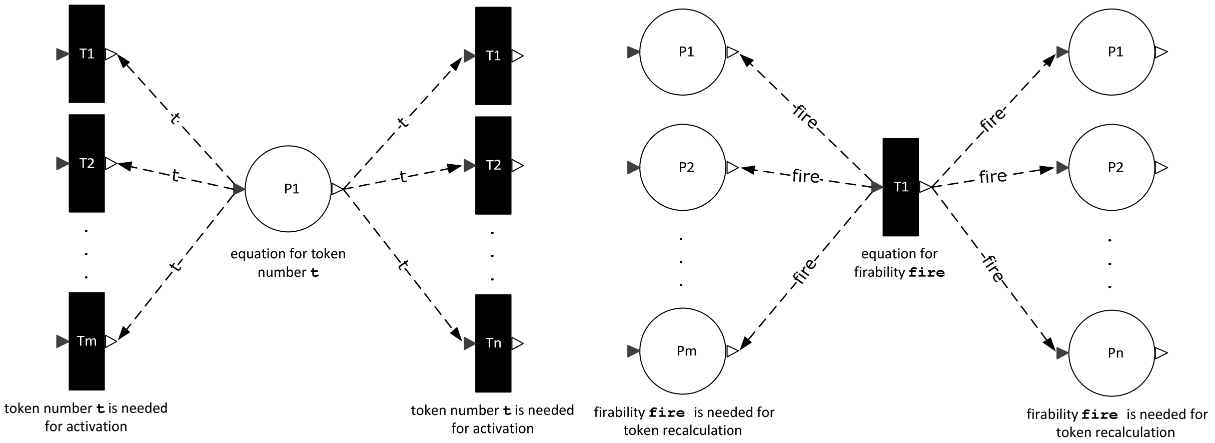

T1
T2
Tm
t
t
t
P1
equation for token number t
t
t
t
T1
T2
Tn
token number t is needed for activation
token number t is needed for activation
P1
P2
Pm
fire
fire
fire
T1
equation for firability fire
fire
fire
fire
P1
P2
Pn
firability fire is needed for token recalculation
firability fire is needed for token recalculation

Figure 6.6: The connector variable t for the token number is output of places and input of transitions (left) and the connector variable fire for the firability is output of transitions and input of places (right)

The connectors of the Petri net component models are vectors to enable the connection to an arbitrary number of input and output components. Therefore, the dimension parameters nIn and nOut are declared in the place and transition models with the connectorSizing annotation.

```
parameter Integer nIn=0 annotation(Dialog(connectorSizing=true))
parameter Integer nOut=0 annotation(Dialog(connectorSizing=true))
Interfaces.PlaceIn inTransition[nIn](<modification equations>)
Interfaces.PlaceOut outTransition[nOut](<modification equations>)
Interfaces.TransitionIn inPlaces[nIn](<modification equations>)
Interfaces.TransitionOut outPlaces[nOut](<modification equations>)
```

If a parameter with the connectorSizing annotation is used as the dimension size of a vector of connectors, it is automatically updated in the following way (Modelica Association 2010):

- If a new connection is added, the dimension parameter is incremented by one and the connection is performed for the new highest index.
- If a connection is deleted, the dimension parameter is decremented by one and all connections with an index above the deleted connection are also decremented by one.

The Petri net components are either connected graphically by drawing a line from a transition to a place or from a place to a transition or textually by a connect equation (see Section 3.1.2) with the following syntax

```
connect "(" componentName1 "." connectorName1 "[" index "]" ","
          componentName2 "." connectorName2 "[" index "]" ")" ";"
```

If a connection is established graphically, the corresponding connect equation is generated on the textual level automatically.

Example 6.1

Figure 6.7 shows a Petri net modeled by the PNlib.

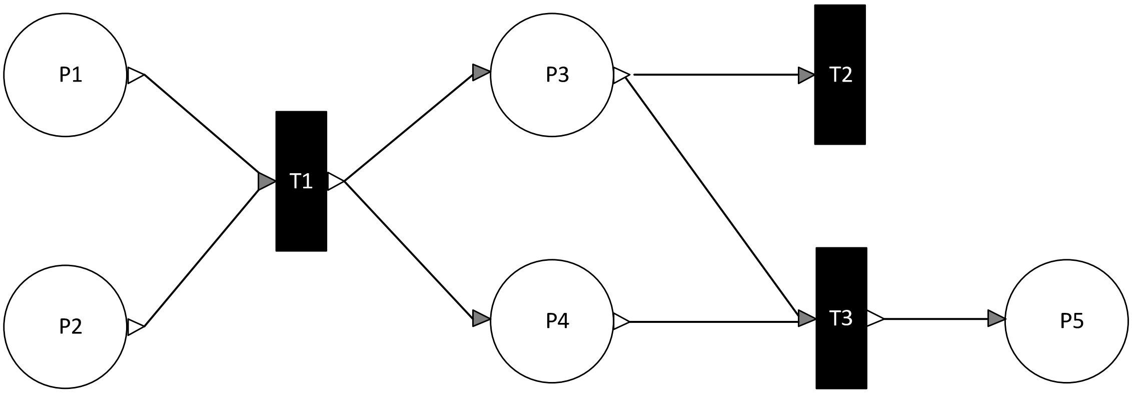


Figure 6.7: Indices of connectors are updated automatically by using the connectorSizing annotation for the dimension parameter (Example 6.1)

The corresponding connect equations are the following

```
connect(P1.outTransition[1], T1.inPlaces[1]);
connect(P2.outTransition[1], T1.inPlaces[2]);
connect(T1.outPlaces[1], P3.inTransition[1]);
connect(T1.outPlaces[2], P4.inTransition[1]);
connect(P3.outTransition[1], T2.inPlaces[1]);
connect(P3.outTransition[2], T3.inPlaces[1]);
connect(P4.outTransition[1], T3.inPlaces[2]);
connect(T3.outPlaces[1], P5.inTransition[1]);
```

If a new connection from $P4$ to $T2$ is drawn, the equation

```
connect(P4.outTransition[2], T2.inPlaces[2]);
```

is generated automatically with the new highest indices of the output connector of $P4$ and the input connector of $T2$. In addition, the dimensions of both connectors are incremented by one.

On the other hand, if the connection from $P3$ to $T3$ is deleted in the graphical editor, the corresponding equation

```
connect(P3.outTransition[2], T3.inPlaces[1])
```

is deleted automatically on the textual level. Additionally, the connector dimensions of $P3$ and $T3$ are decremented by one and the index of the input connector of $T3$ in the connect equation of $P4$ and $T3$ is decremented by one

```
connect(P4.outTransition[1], T3.inPlaces[1]);
```

6.1.2 PLACES

The place models contain several parameters which can be defined by the user. This can either be done by the property dialog on the graphical level or by modification equations (see Section 3.1.2) on the textual level. The property dialog appears by double clicking on the component icon and the contained parameters are detailed in Table 6.1.

Table 6.1: Parameters of discrete (d) and continuous (c) places

Name	Type	Default value	Allowed values	Description
`startTokens/ startMarks`	scalar	`0`	non negative integers (d)/ real values (c)	Marking at the beginning of the simulation Definition 4.65 `minTokens ≤ startTokens ≤ maxTokens`
`minTokens/ minMarks`	scalar	`0`	non negative integers (d)/ real values (c)	Minimum capacity Definition 4.65
`maxTokens/ maxMarks`	scalar	`infinite`	non negative integers (d)/ real values (c)	Maximum capacity Definition 4.65
`enablingType`	choice/ scalar	`Priority`	priority/ probability	Type of enabling if type-1-conflicts occur; the priorities are defined by the connection indices and the probabilities by the variables `enablingProbIn/Out` Definition 4.19, Definition 4.20, Definition 4.67
`enablingProbIn`	vector	`fill(1/nIn,nIn)`	sum must be equal to one	Enabling probabilities of input transitions Definition 4.18
`enablingProbOut`	vector	`fill(1/nOut,nOut)`	sum must be equal to one	Enabling probabilities of output transitions Definition 4.18

`N`	scalar	`settings1.N`	positive integers (d)	Amount of levels for stochastic simulation Section 5.5
`restart`	condition expressions	`false`	Boolean condition expressions	Condition for resetting the marking to `reStartTokens/Marks`; this could may be a global condition
`reStartTokens/` `reStartMarks`	scalar	`0`	non negative integers (d)/ real values (c)	When the `reStart` condition is fulfilled, the marking is set to `reStartTokens/Marks`

If the type-1-conflict is resolved by priorities, the corresponding priorities of the transitions are given by the indices of the connections, i.e. the transition connected to the place with the index 1 has also the priority 1, the transition connected to the place with the index 2 has also the priority 2 etc. Otherwise, if the probabilistic enabling type is chosen, the corresponding probabilities for the transitions have to be entered as a vector (numbers separated by commas within braces). Thereby, the first vector element corresponds to the connection with the index 1, the second to the connection with the index 2 etc. The usage of these parameter vectors is also described in Example 6.2. The input of enabling probabilities as vectors in the place model, and not at the corresponding arcs, is necessary due to the fact that properties cannot be assigned to connections according to the Modelica Specification 3.2 (Modelica Association 2010).

The parameters can also be set by modification equations within brackets after the component name, an example is given below.

```
Discrete.PD PD(startTokens=15, minTokens=8, maxTokens=117,
               enablingType=2,enablingProbIn={0.5,0.25,0.25},
               enablingProbOut={0.7,0.3});
Continuous.PC PC(startMarks=17.8,minMarks=0.9,maxMarks=57.9);
```

If the parameters are set graphically, the modification equations on the textual level are generated automatically and vice versa. Furthermore, for parameters which do not appear in the modification equations the default values are applied.

Example 6.2

Place $P1$ is connected to the transitions $T1$, $T2$, and $T3$ and the connection to $T1$ is indexed by 1, the connection to $T2$ is indexed by 2, and the connection to $T3$ is indexed by 3. Thus, the corresponding connect-equations are

```
connect(P1.outTransition[1], T1.inPlaces[1]);
connect(P1.outTransition[2], T2.inPlaces[1]);
connect(P1.outTransition[3], T3.inPlaces[1]);
```

The enabling probabilities 0.3 for $T1$, 0.25 for $T2$, and 0.45 for $T3$ have to be entered by the parameter vector

```
enablingProbOut={0.3,0.25,0.45}.
```

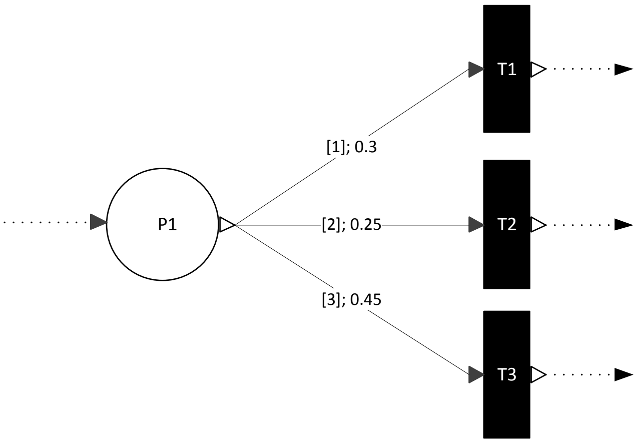


Figure 6.8: The input of parameter vectors (Example 6.2)

The discrete place model has to perform the processes output enabling (D2), input enabling (D3), enabling of continuous transitions (C4), and recalculation of tokens and marks, respectively (D5). The continuous place model also performs the processes D2, D3, and D5 and, in addition, the input and output speeds (C1), marks (C2), and decreasing factors (C7) are calculated by it. These processes are detailed hereafter.

Output and Input Enabling Processes (D2, D3, C4)

If a discrete or continuous place has a type-1-conflict (Definition 4.67) which involves two or more discrete transitions, it has to be resolved either by priorities or probabilities. Thereby, the priorities are reflected by the indices of the connections while the probabilities have to be entered as parameter vectors. Additionally, type-3-conflicts which involve at least one discrete and one continuous transition have to be resolved by the rule that discrete transitions take priority over continuous ones. Furthermore, type-4-conflicts which involve discrete places and at least two continuous transitions have to be resolved by priorities which are also reflected by the corresponding connection indices.

All these conflicts are resolved in the place models by an algorithmic procedure part of the blocks `enablingOutDis`, `enablingInDis`, `enablingOutCon`, and `enablingInCon` which are contained in the sub-package `Blocks`. The blocks for discrete and continuous places are almost the same except the data types of some input arguments and the block enablingOutDis solves also type-4-conflicts. The differentiation is necessary due to the fact that data types of block inputs cannot be variable according to the Modelica Specification 3.2 (Modelica Association 2010).

The enabling process of the places is performed when at least one delay of a connected active transition is passed, i.e. the input connector variable active switches from false to true. This is realized by a when equation (see Section 3.1.2). The algorithmic procedure inside a when equation is only executed at an event instant, thus, exactly at the time when the delay of at least one connected transition is passed. The algorithmic procedure in the block enablingOutDis has to be executed additionally when connected continuous transitions become active or are no longer active, i.e. the connector variable active of at least one continuous output transition switches from false to true or vice versa which is determined with the build-in function change (see Section 3.1.6).

The information which transition is enabled by which places is reported via the connector variable enable to the connected transitions to determine if the transitions are firable.

TOKEN / MARKS RECALCULATION PROCESS AND CALCULATION OF INPUT AND OUTPUT SPEEDS (D5, C1, C2)

Besides the enabling process, the current marking is calculated in the place model. Thereby, it has to be distinguished between the discrete and continuous model. The token numbers of discrete places only change by firing discrete transitions, i.e. only at the corresponding event instants, while the marking of continuous places can change continuously over time as well as discretely.

The discrete marking is recalculated when one or more discrete input or output transitions fire. At that time, the Boolean variable tokeninout switches from false to true and the discrete marking is recalculated by means of Definition 4.13: the arc weight sum of all firing input transitions is added to the predecessor value of the current token number (pre(t)) and the arc weight sum of all firing output transitions is deducted.

```
when tokenInOut or pre(reStart) then
   t = if tokenInOut then pre(t)+firingSumIn-firingSumOut
      else reStartTokens;
end when;
```

Additionally, the tokens are reset to the user-defined value `reStartTokens` when the user-defined condition `reStart` becomes true. This is very useful for resetting the values of several places, simultaneously when a global condition becomes true.

The marking of a continuous place can change continuously as well as discretely (Definition 4.61). This has been implemented by the following construct:

```
der(t) = conMarkChange;
when disMarksInOut then
   reinit(t, t + disMarkChange);
end when;
when reStart then
   reinit(t,reStartMarks);
end when;
```

The continuous mark change is performed by a differential equation while the discrete mark change is performed by the `reinit`-operator within a discrete equation (see Section 3.1.6). This operator causes a re-initialization of the continuous marking every time when at least one connected discrete transition fires. Additionally, the marking is re-initialized by user-defined value `reStartMarks` when the user-defined condition `reStart` becomes true.

Decreasing Factor Calculation (C7)

The continuous place model comprises an additional process, called decreasing factor calculation (C7). The decreasing factors are used to resolve type-2-conflicts (Definition 4.67) by which the input or output speed is not sufficient for firing all transitions at the respective speed. The input and output decreasing factors are calculated in the block `decreasingFactor` by an algorithmic procedure according to Definition 4.50. A block has to be used due to the fact that no events are generated within functions according to the Modelica Specification 3.2 (Modelica Association 2010). The decreasing factors are reported to the connected transitions via the connector variable `decreasingFactor` to calculate the instantaneous speed. In this context, an algebraic loop concerning continuous-time variables may arise which is solved by the Modelica-tool (see Section 3.1.7).

6.1.3 TRANSITIONS

The transition models contain several parameters and variables which can be defined by the user. This can either be done by the property dialog on the graphical level or by modification equations (see Section 3.1.2) on the textual level. The property dialog appears by double clicking on the component icon and the contained parameters and modification possibilities are summarized in Table 6.2. Thereby, it has to be distinguished between the following input types: scalar, vector, scalar function, vector function, and condition expression. The input of the different types is demonstrated by Example 6.3 and Example 6.4. The input of arc weights as vectors in the transition model and not at the respective arcs is necessary due to the fact that connections cannot be provided with properties according to the Modelica Specification 3.2 (Modelica Association 2010).

Table 6.2: Parameters and modification possibilities of discrete (d), stochastic (s), and continuous (c) transitions

Name	Type	Part of	Default value	Allowed values	Description
`delay`	scalar	d	`1`	non-negative real values	Delay of timed transitions Definition 4.32
`h`	scalar or scalar function	s	`1`	non-negative real values	Hazard function to determine the characteristic value of exponential distribution Definition 4.34
`maximumSpeed`	scalar or scalar function	c	`1`	non-negative real values	Maximum speed Definition 4.38
`arcWeightIn`	vector or vector function	d, s, c	`fill(1,nIn)`	non-negative integers (d, s)/real values (c)	Weights of input arcs Definition 4.65
`arcWeightOut`	vector or vector function	d, s, c	`fill(1,nOut)`	non-negative integers (d, s)/real values (c)	Weights of output arcs Definition 4.65
`firingCon`	condition expressions	d, s, c	`true`	Boolean condition expressions	Firing condition Definition 4.62

Example 6.3

Figure 6.9 shows a discrete Petri net. The indices of the connections are written at the arcs within square brackets, e.g. the connection $(P1 \rightarrow T1)$ has the input index [1] and $(T1 \rightarrow P5)$ has the output index [3]. The input of the arc weights displayed after the indices to property dialog or as modification equation is performed by the vector functions

```
arcWeightIn = {2*P1.t,4}
```

and

```
arcWeightOut = {2,1,5*P1.t},
```

whereby the expression `P1.t` accesses the current token number of $P1$. Thus, the weights of the arcs $(P1 \rightarrow T1)$ and $(T1 \rightarrow P5)$ are functions which depend on the token number of $P1$. Transitions can also be provided with additional conditions which have to be satisfied to permit the activation. The condition

```
firingCon = time>9.7
```

means that the transition cannot be activated as long as time is less than 9.7.

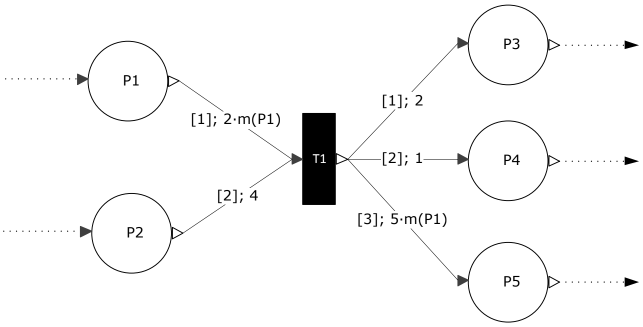


Figure 6.9: The input of arc weights (Example 6.3)

Example 6.4

Figure 6.10 shows two continuous Petri nets. Transition $T1$ has a maximum speed function which depends on the makings of $P1$ and $P2$. The input of this function to the property dialog or as a modification equation is performed by the expression

```
maximumSpeed = 0.75*P1.t*P2.t,
```

whereby `P1.t` and `P2.t` accesses the marks of $P1$ and $P2$, respectively. Transition $T2$ has a maximum speed function that depends on time and can be entered by the expression

```
maximumSpeed = if time<=6.5 then 2.6 else 1.7.
```

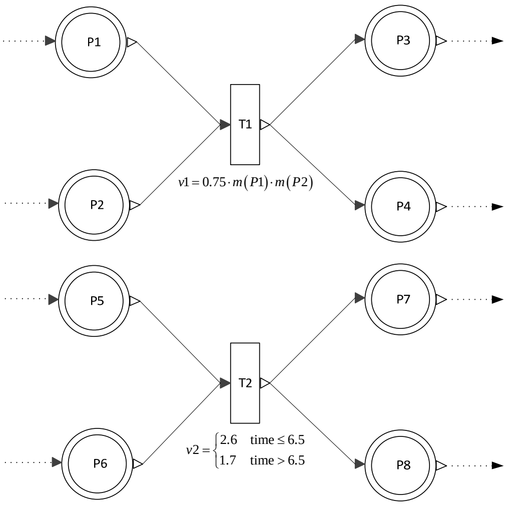

P1
P2
T1
P3
P4
$v1 = 0.75 \cdot m(P1) \cdot m(P2)$
P5
P6
T2
P7
P8
$v2 = \begin{cases} 2.6 & \text{time} \leq 6.5 \\ 1.7 & \text{time} > 6.5 \end{cases}$

Figure 6.10: The input of maximum speeds which depend on markings (top) and time (bottom) (Example 6.4)

The discrete transition model has to perform the processes activation (D1) and firability (D4) and the continuous transition model has also to perform the processes activation (C3) and firability (C5), and, additionally, the preliminary (C6) and instantaneous (C8) speed is calculated. These processes are detailed hereafter.

Activation Process (D1, C3)

The activation process of a discrete transition (D1) and a continuous transition (C3) is performed by an algorithmic procedure in the block `activationDis` and `activationCon`, respectively, based on Definition 4.66. Thereby, an event has to be triggered at every point in time when the transition becomes active to initiate the enabling process of the places. This requires the usage of blocks instead of functions due to the fact that no events are generated in functions according to the Modelica Specification 3.2 (Modelica Association 2010). The continuous activation process differentiates between strongly and weakly input and output active according to Definition 4.46.

When a discrete transition is activated, the next putative firing time is saved as `firingTime`. The variable `delayPassed` indicates if the delay is already elapsed. When the delay is passed, the variable `active` is set to `false` to generate a new event if the transition can become active again. Thereby, the instance of the block `activationDis` is denoted by `activation`.

```
active = activation.active and not pre(delayPassed);
when active then
   firingTime = time + delay;
end when;
delayPassed = active and time>=firingTime;
```

In the case of a stochastic transition, a when-statement contains an equation to determine the next putative firing time. The delay is then an exponentially distributed random number which is generated according to the value of the hazard function `h` at this time instant. Thereby, the exponentially distributed random number is generated by the function `randomexp` which calls an external C-function as described in Section 3.1.4 and Example 3.10. At every event instant a new random number has to be generated; but Modelica functions are only called when the values of the input arguments have changed. They are said to be **pure**. This pure property of a function can be suppressed by the vendor-specific annotation

```
__Dymola_pure=false
```

as it is shown in Example 3.10.

Moreover, a new putative firing time is only generated when a transition has just fired tokens and becomes active again afterwards. The hazard function can depend on the markings of several places to perform a stochastic simulation (see Section 5.5). If the hazard function has changed due to firing other transitions, the putative firing time is only adapted to the new value of the hazard function but no new random number is generated according to the next reaction method of Algorithm 1 in Section 5.5.

```
when active then
   putFireTime := time+Functions.Random.randomexp(h);
   hold := h;
elsewhen (active and h<>hold then
   putFireTime := if h>0 then time+hold/h*(putFireTime-time)
                  else Constants.inf;
   hold := h;
end when;
```

The reusing of the putative times is achieved by a `when`-statement (see Section 3.1.3). The equations of the `elsewhen`-part are active when the hazard function assigned to an active transition has just changed its value. The putative firing time `putFireTime` is then rescaled according to step 5c of Algorithm 1 and the current value of the hazard function is saved as `hold` for the next rescaling.The equations within the `when`-part are active when the transition becomes active. It contains an equation for generating a new putative firing time. The Boolean variable `active` is set to `false` when the transition has just fired tokens. When the transition can become active again, it switches back to `true`. In this manner an event is generated at each firing point in time so that a new putative firing time is always generated.

FIRABILITY (D4, C5)

When the delay of a discrete transition is passed, the connected input places check which transition can be enabled and report back via the connector variable `enable`. If a transition is enabled by all input places, the variable `enabledByInPlaces` is `true`. Based on this, the output places of the transition perform their enabling. If a transition is enabled by all output places, the variable fire becomes true and the transition fires tokens (Definition 4.22).

A continuous transition fires when it is active and, additionally, enabled by all connected discrete places.

PRELIMINARY AND INSTANTANEOUS SPEED CALCULATION (C6, C8)

The preliminary speed is calculated by the block preliminarySpeed according to Definition 4.48. The preliminary speed depends on the activation status of the transition (strongly or weakly input/output active). If a transition is not involved in a type-2-conflict, the instantaneous speed is calculated by means of Definition 4.47; otherwise, the conflict is resolved in the place model and the speed is decreased by the reported decreasing factors (Definition 4.50). The instantaneous speed is reported to the connected places by the connector variable instSpeed to perform the mark change.

```
instantaneousSpeed=min(min(min(decreasingFactorIn),
                min(decreasingFactorOut))*maximumSpeed,prelimSpeed);
```

In this context, an algebraic loop concerning continuous-time variables may arise which is solved by the Modelica-tool (see Section 3.1.7).

6.1.4 ARCS

xHPNs comprise four different kinds of arcs: normal, test, inhibitor, and read arcs. The Modelica language do not support the assignment of properties to arcs that are generated by connect equations (Modelica Association 2010). Due to this fact, the test, inhibitor, and read arcs are realized by component models which are interposed between places and transitions (see Figure 6.11); the normal arc is simply generated by the connect equation.

The parameters of test and inhibitor arcs are summarized in Table 6.3. To avoid connecting the same place to the same transition by a test and normal arc or by an inhibitor and normal arc, the option normalArc can be selected. Then the arc is a double arc.

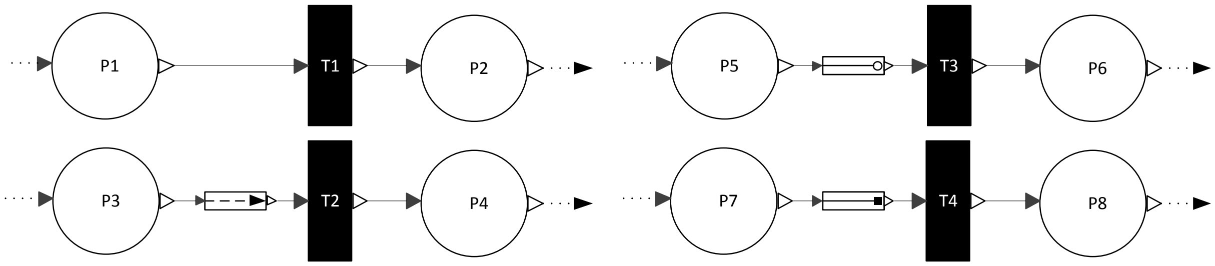


Figure 6.11: Modeling of normal (top left), test (bottom left), inhibitor (top right), and read arcs (bottom right) with the PNlib

Besides the parameters, the models of test, inhibitor, and read arcs solely consist of the two connectors `TransitionIn` and `PlaceOut` and specific modification equations of the output variables within brackets after the connectors.

Table 6.3: Parameters of test and inhibitor arcs

Name	Type	Default value	Allowed values	Description
`testValue`	scalar	`1`	non-negative integers if connected to discrete places, non-negative real values otherwise	The marking of the place must be greater to enable firing of transitions (test arc); the marking of the place must be smaller to enable firing (inhibitor arc).
`normalArc`	choice/scalar	`no`	no or yes	If yes is chosen, then the arc is also a normal arc to change the marking by firing (called **double arc**).

Thereby, the connector variable `arcType` of the `PlaceOut`-connector is set to 2 in case of a test arc, to 3 in the case of an inhibitor arc, and to 4 in the case of a read arc; otherwise, the variable is equal to 1 and indicates a normal arc. The variable `arcType` is needed for the activation process of discrete and continuous transitions (D1, C3, Definition 4.66, Figure 6.3, and Figure 6.4). Additionally, the variables `arcWeight`, `maxSpeed`, `prelimSpeed`, and `instSpeed` of the `TransitionIn`-connector are set to zero to guarantee that the marking is not changed by firing transitions connected by test, inhibitor, or read arcs which are not double arcs. If the arcs are double arcs, the mentioned variables are adopted as they stand. The other connector variables are just conveyed from one connector to another. This process is also illustrated in Figure 6.12. In the case of double arcs, the weights for test and inhibitor arcs are determined by the variable `testValue` of the arc model and the weight for the normal arc are entered by the `arcWeightIn` variable of the transition model. This variable can be set in the property dialog of the transition or by a modification equation (see

Section 6.1.3 and Example 6.5). The component model of read arcs has no parameters or modification possibilities.

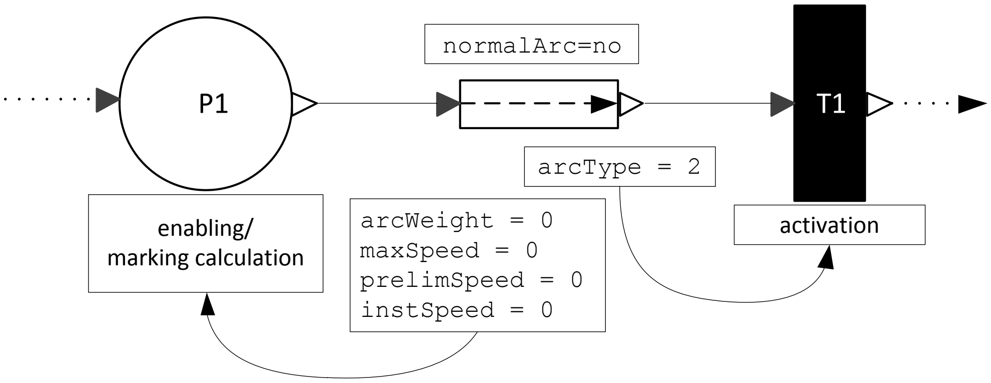


Figure 6.12: The component model of a test arc

Example 6.5

Figure 6.13 shows a Petri net in which the connection $(P2 \rightarrow T1)$ is a double arc (test and normal arc) and $(P5 \rightarrow T2)$ is also a double arc (inhibitor and normal arc). This is modeled by interposing the component models `TA` and `IA` between place and transition. The numbers within square brackets at the arcs are the indices of the connections followed by the weights for the normal arcs. The test values for test and inhibitor arcs are written below the `TA` and `IA` model, respectively. Then, the `arcWeightIn` variable of $T1$ is the vector `{3, 5, 8}` and for $T2$ it is `{2.8, 1.2}`. Additionally, `testValue=7` for TA and `testValue=1.75` for IA. This means that $T1$ can only be activated when $P2$ has more than 7 tokens and $T2$ can only be activated when $P5$ has less than 1.75 marks. In addition, 5 tokens are removed from $P2$ every time $T1$ fires and 1.2 marks are removed from $P5$ continuously in the time of firing $T2$ due to the fact that both arcs are double arcs.

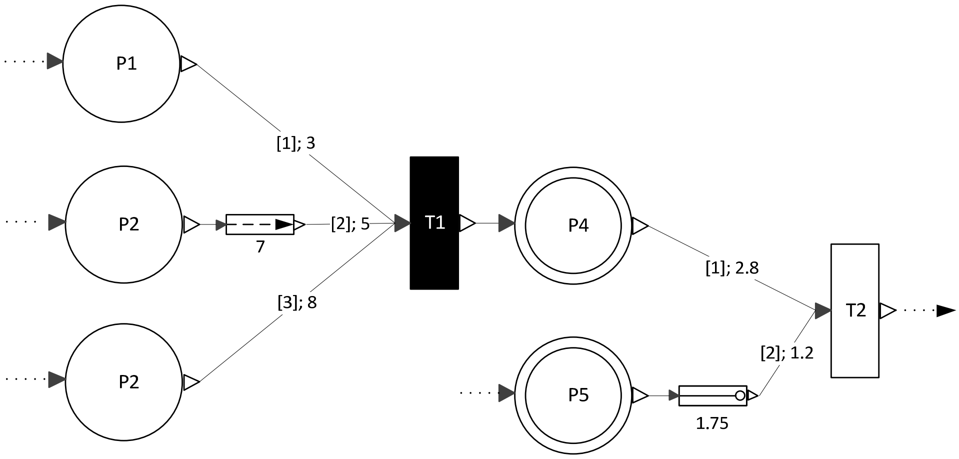
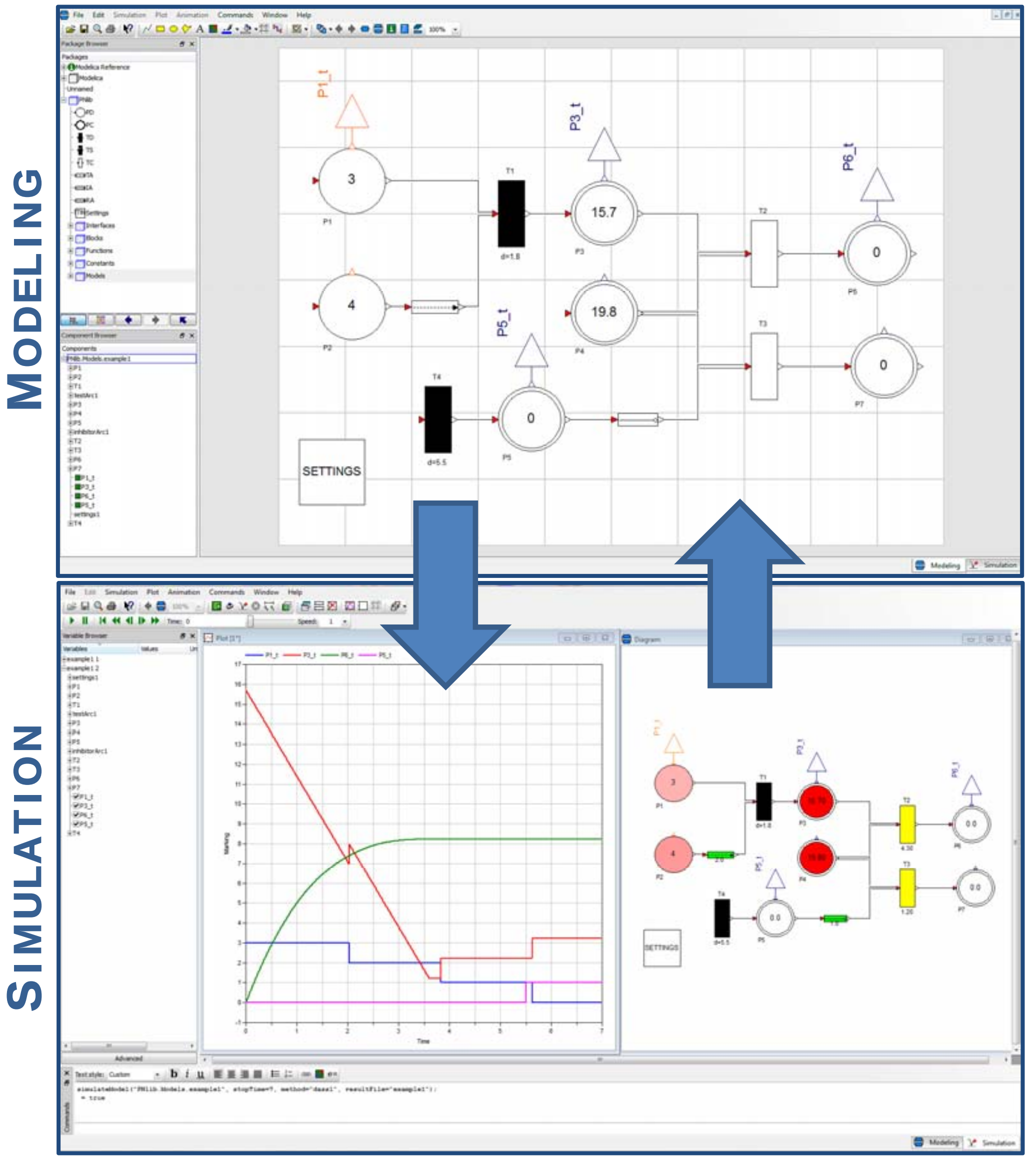


Figure 6.13: A Petri net with test and inhibitor arcs (Example 6.5)

6.2 Modeling, Simulation, and Animation with Dymola

To use the PNlib for graphical modeling, simulation, and animation of xHPNs an appropriate environment is needed. In this study, the tool Dymola version 7.4 is used. This section gives an introduction in the usage of the PNlib in combination with the Dymola tool.

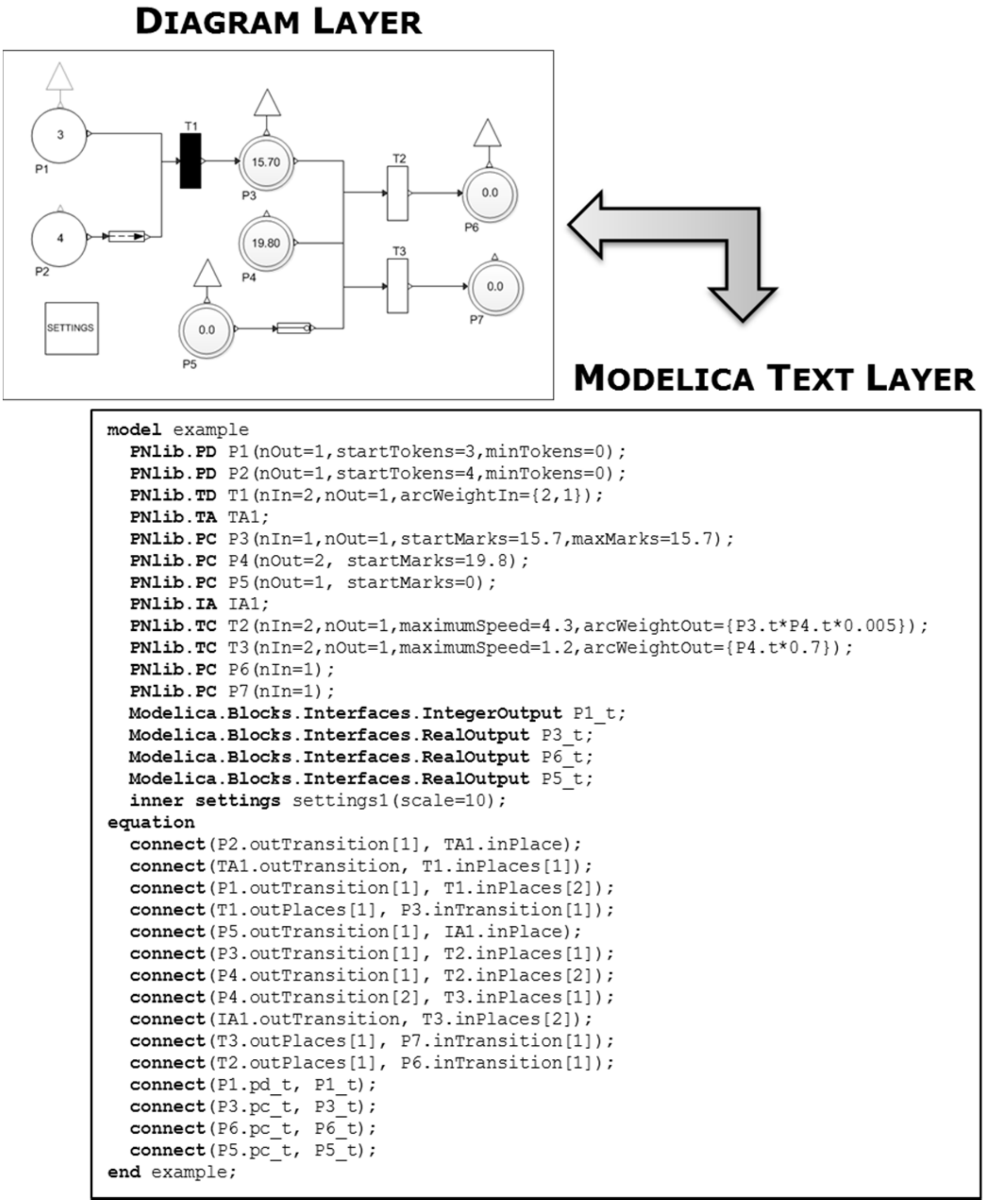
The main window of Dymola has two different operating modes: Modeling and Simulation. The buttons Modeling and Simulation in the lower right corner allow switching from one mode to the other (see Figure 6.14).



```
simulateModel("PNlib.Models.example1", stopTime=7, method="dassl", resultFile="example1");
 = true
```

Figure 6.14: The main window of Dymola with two modes: Modeling (top) and Simulation (bottom)

The main window in the Modeling mode is used to establish models and model components. It consists of a modeling area, a package browser (upper left), and a component browser (lower left). The package browser comprises open packages and displays them in a hierarchical structure. The model components can be drag from it to the diagram layer in order to construct a model. The component browser summarizes all components in a tree structure that are used in the model. With the button ▤, the model can be viewed on the textual level and with the button ⎙, it is displayed in the diagram layer (see Figure 6.15).



```
model example
  PNlib.PD P1(nOut=1,startTokens=3,minTokens=0);
  PNlib.PD P2(nOut=1,startTokens=4,minTokens=0);
  PNlib.TD T1(nIn=2,nOut=1,arcWeightIn={2,1});
  PNlib.TA TA1;
  PNlib.PC P3(nIn=1,nOut=1,startMarks=15.7,maxMarks=15.7);
  PNlib.PC P4(nOut=2, startMarks=19.8);
  PNlib.PC P5(nOut=1, startMarks=0);
  PNlib.IA IA1;
  PNlib.TC T2(nIn=2,nOut=1,maximumSpeed=4.3,arcWeightOut={P3.t*P4.t*0.005});
  PNlib.TC T3(nIn=2,nOut=1,maximumSpeed=1.2,arcWeightOut={P4.t*0.7});
  PNlib.PC P6(nIn=1);
  PNlib.PC P7(nIn=1);
  Modelica.Blocks.Interfaces.IntegerOutput P1_t;
  Modelica.Blocks.Interfaces.RealOutput P3_t;
  Modelica.Blocks.Interfaces.RealOutput P6_t;
  Modelica.Blocks.Interfaces.RealOutput P5_t;
  inner settings settings1(scale=10);
equation
  connect(P2.outTransition[1], TA1.inPlace);
  connect(TA1.outTransition, T1.inPlaces[1]);
  connect(P1.outTransition[1], T1.inPlaces[2]);
  connect(T1.outPlaces[1], P3.inTransition[1]);
  connect(P5.outTransition[1], IA1.inPlace);
  connect(P3.outTransition[1], T2.inPlaces[1]);
  connect(P4.outTransition[1], T2.inPlaces[2]);
  connect(P4.outTransition[2], T3.inPlaces[1]);
  connect(IA1.outTransition, T3.inPlaces[2]);
  connect(T3.outPlaces[1], P7.inTransition[1]);
  connect(T2.outPlaces[1], P6.inTransition[1]);
  connect(P1.pd_t, P1_t);
  connect(P3.pc_t, P3_t);
  connect(P6.pc_t, P6_t);
  connect(P5.pc_t, P5_t);
end example;
```

Figure 6.15: A Modelica model displayed in the diagram layer (top) and in the Modelica text layer (bottom)

Before an established model is simulated, it should be checked. This check is performed by clicking the button and comprises several symbolic and syntactic tests to trap a lot of errors. If the check was successful, a message dialog appears to indicate that the requirements for a subsequent simulation are met.

A successfully checked model can be translated and simulated by switching to the Simulation mode. A model is translated to C-code (see also Section 3.1.7) by clicking the button and the button performs the simulation and also the translation if it was not performed in advance. Dymola offers a wide range of setup possibilities to manipulate the simulation. The setup dialog appears by clicking the button . There, the start and stop time can be set and, additionally, an integration method out of 16 possibilities can be chosen, e.g. Dassl, Euler, Runge-Kutta.

When the simulation has been performed successfully, the results can be displayed as plots. The variable browser on the left hand side shows all variables of the model to plot in a hierarchical structure (see Figure 6.14). Thereby, the variable `t` of the place model accesses the marking evolution over time.

Another possibility to represent the simulation results of an xHPN model is an animation. Thereby, several settings can be made in the property dialog of the `settings`-box. These settings are global and, thus, affect all components of the Petri net model. Using the prefixes `inner` and `outer` (see Section 3.1.1), means that the settings are common to all Petri net components of a model. The `settings`-box provides the following display and animation options:

- `showPlaceName`: displays the names of places,
- `showTransitionName`: displays the names of transitions,
- `showDelay`: displays the delays of discrete transitions,
- `showCapacity`: displays the minimum and maximum capacities of places,
- `showWeightTIarc`: displays the arc weights of test and inhibitor arcs,
- `animateMarking`: animates the current marking in the places; the change of tokens/marks is displayed in the places during animation,
- `animatePlace`: animates the color of places. Places change their degree of redness according to the amount of tokens/marks; thereby, the redness degree is scaled by the parameter `scale` from 0 to 100,

- `antimateTransition`: animates the color of transitions. Transitions change their color to yellow when they fire; thereby, discrete transitions blink yellow for specified time units `(timeFire)` while continuous transitions are yellow the entire time while firing,
- `animatePutFireTime`: animates the putative firing time of stochastic transitions; the putative firing time is displayed under the transition during animation,
- `animateHazardFunc`: animates the hazard function of stochastic transitions; the hazard function is displayed under the transition during animation,
- `animateSpeed`: animates the instantaneous speed of continuous transitions; the instantaneous speed is displayed under the transition during animation,
- `animateWeightTIarc`: animates the weights of test and inhibitor arcs; the weights are displayed under the arc during animation,
- `animateTIarc`: animates the color of test and inhibitor arcs; the arc is green when the weight is satisfied and red otherwise,

The animation toolbar allows control of the animation. An animation offers a way to analyze the marking evolutions of large and complex xHPNs. Figure 6.16 shows four selected points in time of the animation of an xHPN example.

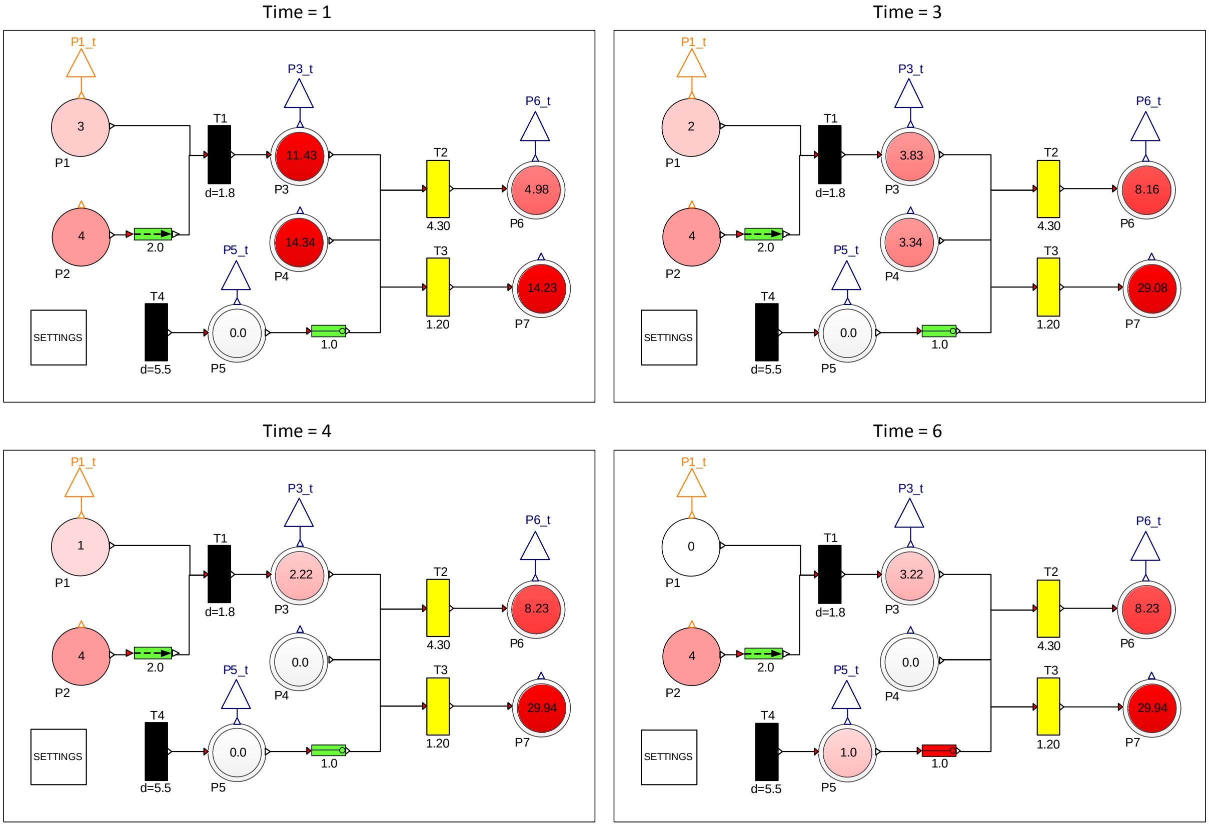


Figure 6.16: Animation of an xHPN model

All display and animation options are realized with the `DynamicSelect` annotation. The first argument specifies the value of the editing state and the second argument the value of the non-editing state (Modelica Association 2010).

```
annotation(Icon(...Text(...textString=DynamicSelect("%name",
     if showPlaceName==1 then "%name" else " ")),...));
```

6.3 Connection between Dymola and Matlab/Simulink

To simulate the established Modelica model several times with different parameter settings and use the arising simulation results for parameter estimation, sensitivity analysis, deterministic and stochastic hybrid simulation, or process optimization (see Chapter 4), the Modelica models in Dymola are connected to Matlab/Simulink. This is realized with the aid of a Dymola interface in Simulink and a set of Matlab m-files utilities (Dassault Systèmes AB 2011). After some specific paths are set in Matlab (see Dassault Systèmes AB 2011 for a detailed description), the Dymola interface can be found in the Simulink library browser and dragged to Simulink models. The connection process is performed using the following steps:

1. Construct an arbitrary Modelica model, for example, an xHPN with the PNlib. All variables whose values should be available in Matlab have to be declared with the prefix `output` on the highest level. If the marking of a place is needed for subsequent analyses, this is achieved by creating a connector of the output connector at the top of the place icon. In the case of discrete places it is an orange `IntegerOutput` connector and in the case of continuous places it is a blue `RealOutput` connector as displayed in Figure 6.17; both connectors are part of the Modelica standard library. In the example of Figure 6.18 the markings of places $P1$, $P3$, $P5$, and $P6$ are available in Matlab. Similarly, all variables which are declared as input on the highest level in Modelica can be determined in Matlab/Simulink.
2. Create a new Simulink model file and drag a DymolaBlock into it.
3. Open the GUI by clicking on the DymolaBlock. Enter the name of the model and its path or click the button *Select from Dymola* if the model has just been opened in Dymola.

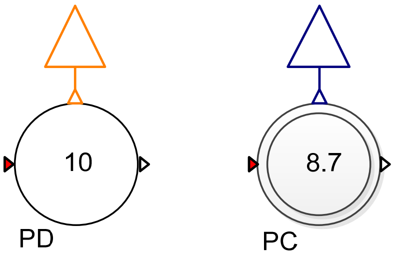


Figure 6.17: `IntegerOutput` (left) and `RealOutput` (right) connectors to access the marking of the places in Matlab/Simulink

4. Compile the model by clicking the button *Compile model*.
5. After successful compilation, all parameters and start values of the corresponding Modelica model are displayed in the GUI in a hierarchical structure.
6. The parameters and start values can either be changed in the GUI or by specific functions called within an algorithmic procedure of an m-file. The following table contains the functions needed for loading and changing parameters and start values (Dassault Systèmes AB 2011).

Table 6.4: Functions for loading and changing parameters in Matlab procedures (Dassault Systèmes AB 2011)

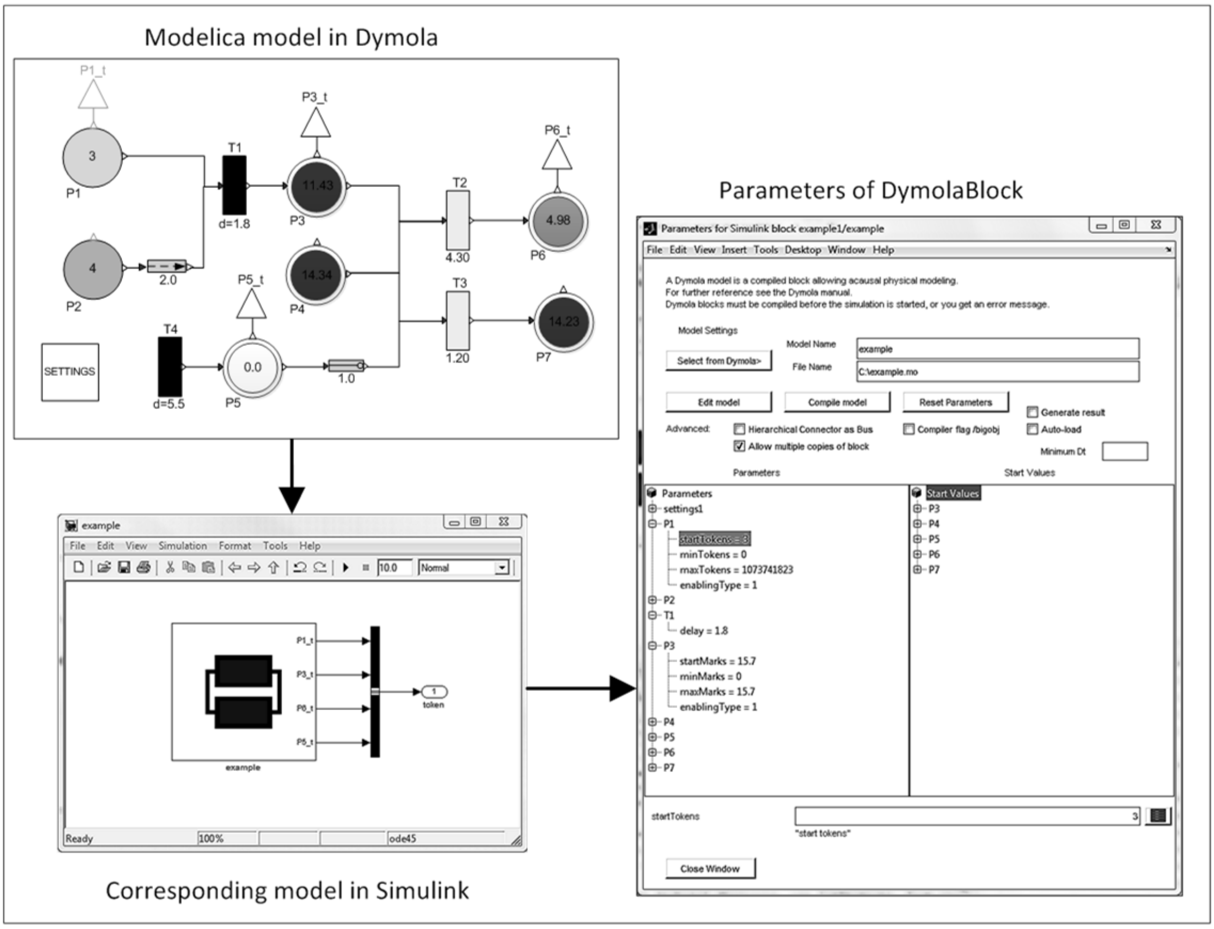
Name	Explanation
loadsin	Loads values from dsin.txt (or <modelname>.txt file).
setParameterByName	Sets parameters and start values using the name of the corresponding Modelica variable.
setParametersFDsin	Modifies the parameters and start values of a DymolaBlock.
setfromdsin	Sets parameters and initial conditions from values in dsin.txt (the same as Reset Parameters in the DymolaBlock GUI). Calls loaddsin followed by setParametersFDsin.

7. For each variable declared as output in Modelica, a corresponding output port is added to the DymolaBlock. The names of these ports are those of the Modelica variables. The values of the output variables over time can be summarized in a matrix by connecting all ports to a bus creator (black bar in Figure 6.18) which is in turn connected to an Outport block. Similarly, for all input variables of the Modelica model, a corresponding input port is added to the DymolaBlock. These input ports can be, for example, connected to external signal sources.
8. Simulate the Simulink model either with the Simulink GUI or by the prompt

```
[t,x,y] = sim(model,timespan,options,ut)
```

within Matlab procedures (see Matlab help for a detailed description).

9. The arising simulation results can be used for further calculations (parameter estimation, sensitivity analysis, stochastic simulation, and process optimization).


Modelica model in Dymola
P1_t
P3_t
P6_t
3
P1
T1
d=1.8
11.43
P3
T2
4.30
4.98
P6
4
P2
2.0
P5_t
14.34
P4
T3
1.20
14.23
P7
T4
SETTINGS
d=5.5
0.0
P5
1.0
Corresponding model in Simulink
example
File Edit View Simulation Format Tools Help
10.0
Normal
P1_t
P3_t
P6_t
P5_t
example
1
token
Ready
100%
ode45
Parameters of DymolaBlock
Parameters for Simulink block example1/example
File Edit View Insert Tools Desktop Window Help

A Dymola model is a compiled block allowing acausal physical modeling.
For further reference see the Dymola manual.
Dymola blocks must be compiled before the simulation is started, or you get an error message.

Model Settings

Select from Dymola> Model Name: example
File Name: C:\example.mo

Edit model | Compile model | Reset Parameters

□ Generate result
Advanced: □ Hierarchical Connector as Bus □ Compiler flag /bigobj □ Auto-load
☑ Allow multiple copies of block
Minimum Dt

Parameters
- Parameters
  - settings1
  - P1
    - startTokens = 3
    - minTokens = 0
    - maxTokens = 1073741823
    - enablingType = 1
  - P2
  - T1
    - delay = 1.8
  - P3
    - startMarks = 15.7
    - minMarks = 0
    - maxMarks = 15.7
    - enablingType = 1
  - P4
  - P5
  - P6
  - P7

Start Values
- Start Values
  - P3
  - P4
  - P5
  - P6
  - P7

startTokens: 3
"start tokens"

Close Window

Figure 6.18: By the Simulink interface DymolaBlock, a connection between Dymola and Matlab/Simulink can be established

The general procedure of a Matlab algorithm which performs several simulations with different parameter settings and uses the arising results for further calculation is outlined in Figure 6.19.

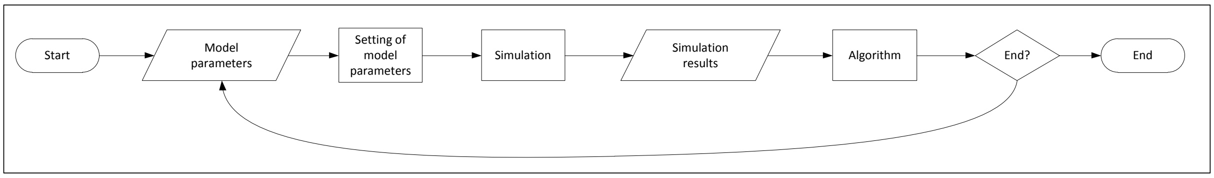


Figure 6.19: General procedure of a Matlab algorithm which performs several simulations with different parameter settings and uses the results for further calculations

6.4 Petri Net Library for Process Modeling of Biological Systems

Based on the PNlib, a library specifically adapted for modeling biological systems has been developed, called **PNproBio** (**P**etri **N**ets for **pro**cess modeling of **Bio**logical systems). This library comprises so-called **wrappers** to simplify the modeling process. These wrappers are sub-models consisting of connected component models of the PNlib which represent specific biological processes and reactions. With this procedure, a wrapped xHPN model can be reused several times in the same or in different models which offers on the one hand an easy-to-use model at the top level with an intuitive and familiar adapted biological view and on the other the flexibility and generality of the xHPN concept at a lower level.

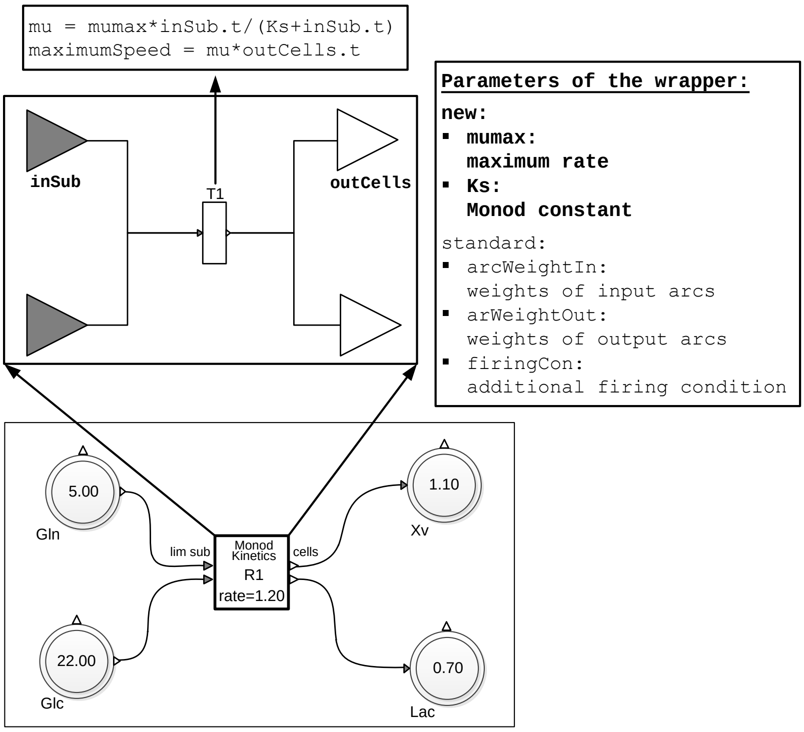
The PNproBio library is further divided into the following sub-libraries:

- **Kinetics**: comprises continuous transitions with mass action kinetics, Michaelis-Menten kinetics, or enzyme inhibition kinetics as maximum speed functions,
- **Stochastic**: comprises stochastic transitions with stochastic mass action kinetics or stochastic level kinetics as hazard function in order to perform stochastic simulations (see Section 5.5),
- **CellGrowth**: comprises sub-models for modeling growth with and without inhibition mechanisms,
- **CellDeath**: comprises sub-models for modeling death of organisms,
- **SubstrateUptake**: comprises sub-models for modeling the uptake of a substrate of an organism from the environment,
- **ProductFormation**: comprises sub-models for modeling the product formation of an organism,
- **Process**: comprises sub-models for modeling activation or inhibition mechanisms of processes or reactions,
- **Fermenter**: comprises sub-models for modeling fermentation processes,
- **OddAndEnds**: Several sub-models that are needed within different kinds of models.

An example of a wrapping process is depicted in Figure 6.20. The wrapper $R1$ represents the Monod kinetics which models the growth of cells limited by a substrate. It consists of one continuous transition with the Monod kinetics

$$\mu \cdot X_v = \frac{\mu_{max} S}{S + K_s} \cdot X_v \qquad \text{Eq. 6-1}$$

as maximum speed function, whereby μ_{max} is the maximum specific growth rate, K_s is the substrate concentration at half maximum rate, S is the concentration of the limited substrate, and X_v is the concentration of living cells. The limited substrate glutamine is modeled by the place Gln. This place is connected to the upper input port of the Monod kinetics wrapper and the place Xv which represents the living cell concentration is connected to the upper output port. Other non-limited substrates which are needed for growth are connected to the lower input port and similar by-products which arise during growth are connected to the lower output port. In this example, these are glucose and lactate which are represented by the places Glc and Lac, respectively. The arc weights are coefficients which represent the yield of cells and by-products from the substrates. The wrapping of the Monod kinetics means that only the two parameters μ_{max} and K_s have to be entered instead of the entire maximum speed function each time.



```
mu = mumax*inSub.t/(Ks+inSub.t)
maximumSpeed = mu*outCells.t
```

Parameters of the wrapper:

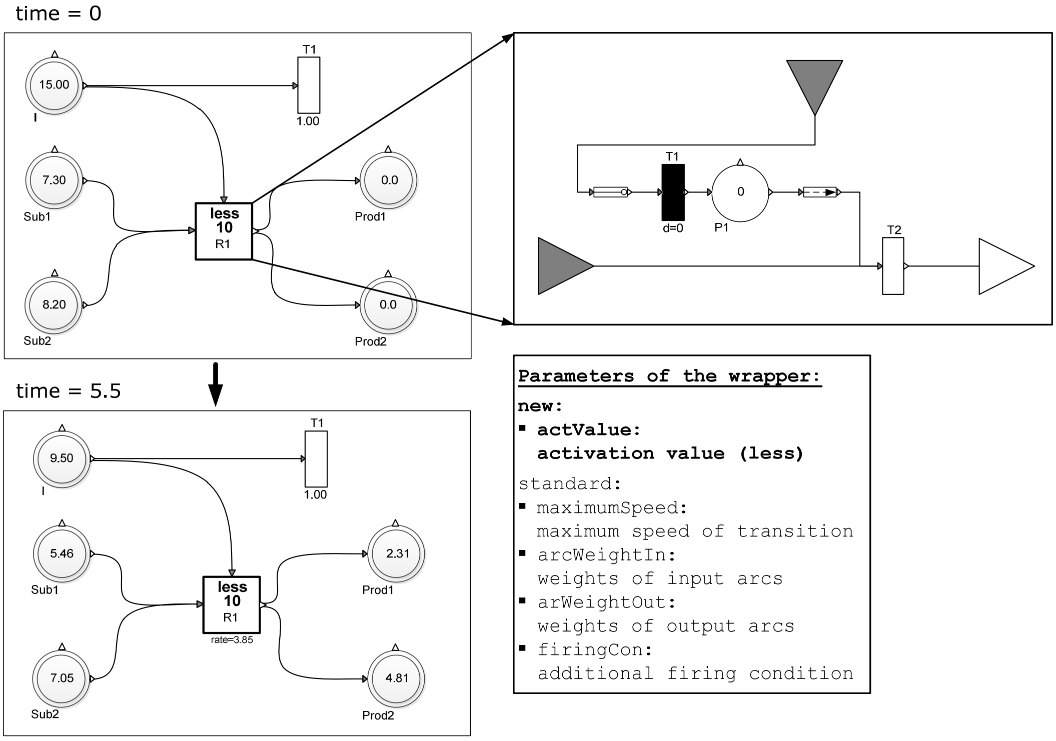
new:
- **mumax:**
  maximum rate
- **Ks:**
  Monod constant

standard:
- arcWeightIn:
  weights of input arcs
- arWeightOut:
  weights of output arcs
- firingCon:
  additional firing condition

Figure 6.20: Wrapping process of the Monod kinetics

Figure 6.21 shows the wrapper for activating a process. The reaction $R1$ proceeds first when the marking of I becomes less than the activation value of 10. Behind the wrapper is a Petri net model that connects I to a discrete transition via an inhibitor arc. When the marking of I falls below 10, one token is added to the discrete Place $P1$. $P1$ is connected to the continuous transition via a test arc with the test value one. This causes that $T2$ becomes firable at the time when $P1$ gets the token and remains firable regardless of whether the activation value is exceeded or not.

Furthermore, a fermentation process can be modeled with the aid of the wrappers summarized in the **Fermenter** sub-library. Figure 6.22 (a) shows an example of a fermentation model. It is comprised of a red fermenter wrapper to determine the operation mode, and specific continuous places - displayed with a red margin - to consider the volume change during fermentation if repeated batch, fed batch, or continuous feeding is applied as the feeding strategy.



```
Parameters of the wrapper:
new:
▪ actValue:
  activation value (less)
standard:
▪ maximumSpeed:
  maximum speed of transition
▪ arcWeightIn:
  weights of input arcs
▪ arWeightOut:
  weights of output arcs
▪ firingCon:
  additional firing condition
```

Figure 6.21: Wrapper for process activation; the reaction *R*1 proceeds first when the marking of *I* becomes less than 10

The fermenter wrapper offers the possibility to model the following operation modes, whereby the parameter `Vstart` is the start volume of the fermenter (see **Figure** 6.23):

- **Batch**: No additional substrate is added during the fermentation process,
- **Repeated Batch fixed interval**: A volume (`Vin`) with specific substrate concentrations (`Sin`) is added to the fermenter at fixed time steps (`deltaT`) starting at a determined point in time (`feedStartTime`) and stopping when the fermenter reaches the maximum capacity (`Vmax`),
- **Repeated Batch fixed feeding number**: A volume (`Vin`) with specific substrate concentrations (`Sin`) is added to the fermenter at `n` fixed points in time (`feedingTimes`); thereby, the feeding concentrations can be different at each feeding time,

- **Fed Batch**: A volume is added to the fermenter continuously with the substrate concentration `Sin` and the flow rate `Fin` starting from a determined point in time (`feedStartTime`) until the fermenter reaches the maximum capacity (`Vmax`); hence, the volume (V) changes by the differential equation $\frac{dV}{dt} = F_{in}$,
- **Continuous**: A volume is added to the fermenter continuously with the substrate concentration `Sin` and the flow rate `Fin` and it is also taken away continuously with the flow rate `Fout` starting from a determined point in time (`feedStartTime`); hence, the volume changes by the differential equation $\frac{dV}{dt} = F_{in} - F_{out}$. Usually, equal input and output flows are chosen ($F_{in} = F_{out}$) so that $\frac{dV}{dt} = 0$.

The sub-model of the fermenter is displayed in Figure 6.22 (b). If a (discrete) repeated batch mode is chosen, the discrete transition $T1$ fires by adding `Vin` marks to the volume place `V` and

```
Sin*Vin/(V.t+Vin)
```

to the connected substrate places at specified points in time while the continuous transition fires with the maximum speed `Fin` in the case of fed batch or continuous mode. Thereby, the weight of the arc to the volume place is one and the arc weights of the substrate places are determined by

```
Sin/V.t.
```

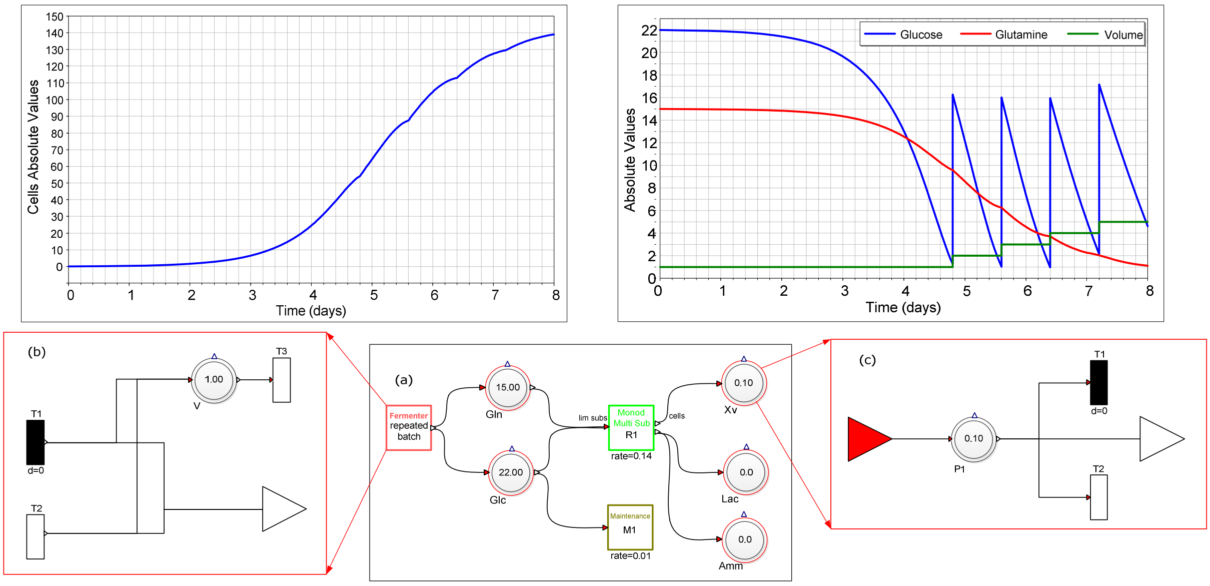


Figure 6.22: Modeling a fermentation process with the wrappers of the Fermenter sub-library: (a) an example of a fermentation model, (b) the fermenter wrapper, and (c) the fermenter place wrapper

Furthermore, a modified continuous place is needed to model fermentation processes due to the fact that the concentrations of the regarded substances change by changing the volume. This wrapper is depicted in Figure 6.22 (c). In the case of repeated batch, the discrete transition $T1$ removes

`P1.t*Vin/(V.t+Vin)`

marks from the place at each feeding time and in the case of fed batch and continuous feeding, the continuous transition $T2$ fires at the maximum speed

`Fin/V.t*P1.t`

to adapt the concentration to the current fermenter volume. The prefixes `inner` and `outer` are used to make the marking of the volume place (`V.t`), the volume addition (`Vin`), the flow rate (`Fin`), and the applied operation mode (`mode`) available in the modified place model and in this manner perform the respective adaptations. Figure 6.22 shows the simulation results of a repeated batch experiment. Thereby, $1\ l$ volume with a glucose concentration of $15\ g/l$ is added to the fermenter every 0.80 days starting at day 4.8 and stopping when the maximum volume of $5\ l$ is reached.

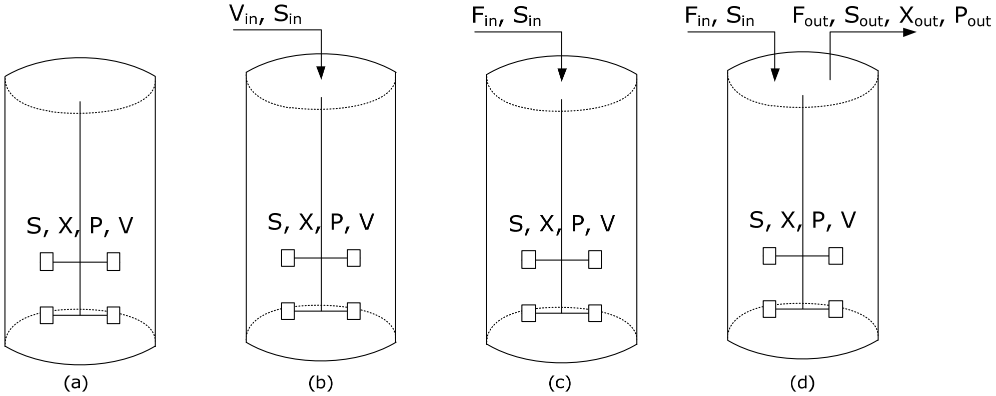


Figure 6.23: Possible operation modes of a fermenter: (a) batch, (b) repeated batch, (c) fed-batch, and (d) continuous mode

Moreover, to simplify the stochastic modeling process of biochemical reactions, two wrappers have been implemented which can be found in the sub-library **Stochastic** of the PNproBio library. Both wrappers consist of a stochastic transition with a specific hazard function so that only the rate constants have to be entered instead of the whole function each time. The putative firing time of the wrapper **StochasticMassAction** is generated by using the stochastic mass action hazard function in Eq. 5-19 while the wrapper **StochasticLevel** uses the stochastic level hazard function in Eq. 5-20. The values for the maximum concentration M and the amount of levels $N + 1$ can be determined globally in the Settings-component and the parameter N can also be defined locally in the place components.

6.5 Tool for the Analysis of Modelica Models

In addition to the Modelica libraries PNlib and PNproBio, a Matlab-tool, called **AMMod** (**A**nalysis of **M**odelica **Mod**els), has been implemented to realize the following steps of the modeling process:

- Preprocessing and Relationship Analysis (PRA, Section 5.1),
- Parameter Estimation (PE, Section 5.3),
- Sensitivity Analysis (SA, Section 5.4),
- Deterministic and Stochastic Hybrid Simulation (DHS, SHS, Section 5.5),
- Model Predictions (MP, Section 5.6), and
- Process Optimization (PO, Section 5.7).

Figure 6.24 again shows the developed modeling process of Section 5 but now with a concrete assignment of the modeling steps to the tools. Orange-marked steps are performed with the AMMod-tool in Matlab/Simulink and green-marked steps are accomplished by the Modelica-tool Dymola and the Modelica libraries PNlib and PNproBio. The deterministic hybrid simulation is marked by both colors to indicate that it can be performed with both tools.

At first, the Modelica model has to be constructed in Dymola either graphically with the aid of PNlib and PNproBio or textually with a system of discrete, differential, and algebraic equations. Afterwards, the model has to be connected to Matlab/Simulink with the Dymola interface as described in Section 6.3 to perform one of the analysis methods above. The analysis methods can be chosen from the main menu of the AMMod-tool:

- PRA: Preprocessing and Relationship Analysis,
- PESA: Parameter Estimation and Sensitiviy Analysis of structure parameters,
- DHS: Deterministic Hybrid Simulation,
- SHS: Stochastic Hybrid Simulation, and
- POSA: Process Optimization and Sensitivity Analysis of process parameters.

If **PRA** is selected, the GUI of the curve fitting toolbox of Matlab appears and experimental data can be preprocessed by smoothing splines and relationships between biological compounds can be analyzed as described in Section 5.1 (see Appendix A3-1 for more details).

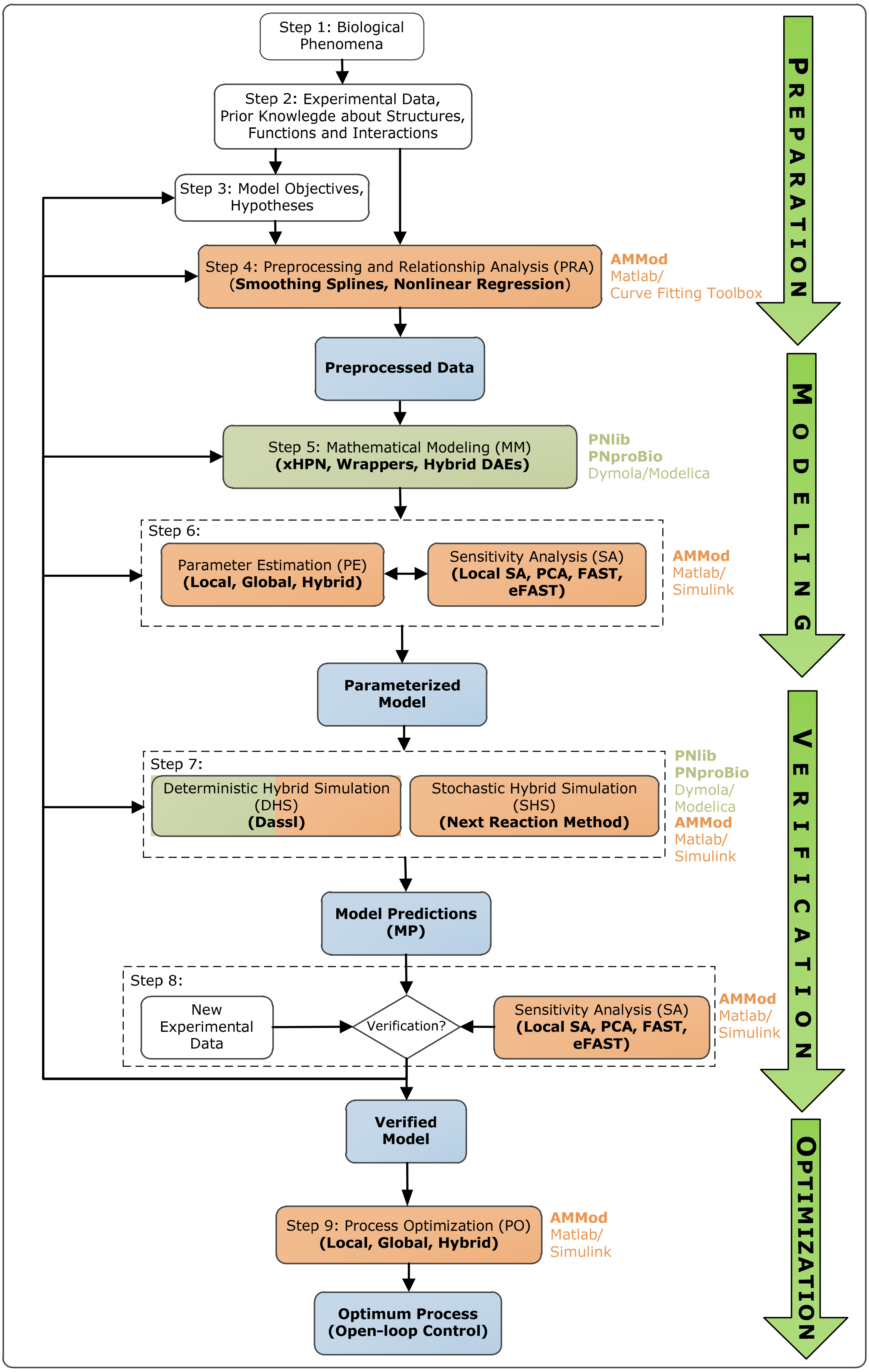

Step 1: Biological Phenomena
Step 2: Experimental Data, Prior Knowlegde about Structures, Functions and Interactions
Step 3: Model Objectives, Hypotheses
Step 4: Preprocessing and Relationship Analysis (PRA) (Smoothing Splines, Nonlinear Regression)
AMMod Matlab/ Curve Fitting Toolbox
Preprocessed Data
Step 5: Mathematical Modeling (MM) (xHPN, Wrappers, Hybrid DAEs)
PNlib PNproBio Dymola/Modelica
Step 6:
Parameter Estimation (PE) (Local, Global, Hybrid)
Sensitivity Analysis (SA) (Local SA, PCA, FAST, eFAST)
AMMod Matlab/ Simulink
Parameterized Model
Step 7:
Deterministic Hybrid Simulation (DHS) (Dassl)
Stochastic Hybrid Simulation (SHS) (Next Reaction Method)
PNlib PNproBio Dymola/ Modelica AMMod Matlab/ Simulink
Model Predictions (MP)
Step 8:
New Experimental Data
Verification?
Sensitivity Analysis (SA) (Local SA, PCA, FAST, eFAST)
AMMod Matlab/ Simulink
Verified Model
Step 9: Process Optimization (PO) (Local, Global, Hybrid)
AMMod Matlab/ Simulink
Optimum Process (Open-loop Control)
PREPARATION
MODELING
VERIFICATION
OPTIMIZATION

Figure 6.24: The modeling process of Figure 5.1 with the assignment of the modeling steps to the developed tools. Orange-marked steps are performed with AMMod in Matlab/Simulink and green-marked steps are performed with the Modelica-tool Dymola and PNlib/PNproBio

The **PESA** option offers the possibility to perform the estimation and sensitivity analysis of structure parameters to adapt the model as good as possible to the experimental data. Therefore, the experimental data has to be prepared in an Excel map and several settings have to be made on the PESA GUI as described in Appendix A3-2. The results of PE and SA are saved in a .mat-file.

The **DHS** option offers the possibility to perform a deterministic hybrid simulation of the Modelica model in Matlab/Simulink while the **SHS** option executes a stochastic hybrid simulation as described in Section 5.5. For both methods the path of the Simulink model, the name of the Dymola block, and the path of the input file have to be entered into the GUI. Additionally, the start time, the stop time, and the step size can be chosen. Afterwards, the model parameters can be loaded into the GUI to adapt them for the simulation. The SHS requires also a determination of the number of simulation runs and the number of means that have to be calculated from the simulation data. The model outputs to plot can then be selected, and plotted by clicking *plot*. The simulation results are saved in a .mat-file.

The last option on the AMMod main menu is to perform a process optimization and sensitivity analysis of process parameters (**POSA**) as described in Section 5.7. Therefore, the same optimization and sensitivity analysis methods can be applied as mentioned for the PESA option (see Appendix A3-2). The aim of PO can either be to minimize or to maximize one output of the model. This model output is determined by entering the number of its output port at the Dymola block. The port $P1\_t$ in Figure 6.18, for example, has the number 1 and the port $P6\_t$ the number 3. Furthermore, time can be integrated to the objective function due to the fact that it usually correlates with the cost of the process. Thus, the objective functions in Eq. 5-24 and Eq. 5-25 are possible. After the model inputs have been set, the model parameters can be loaded into the GUI and the process parameters can be selected.

The application of the AMMod tool is described exemplarily in Chapter 7 by modeling the xanthan production of *Xanthomonas campestris* bacteria.

6.6 Comparison to other Petri net Tools

The following table summarizes the advantages of the PNlib towards the existing tools mentioned in Chapter 2 and clarifies the demand of developing a new Petri net simulation environment for biological applications.

Table 6.5: Comparison of the other Petri nets tools mention in Chapter 2 and the new Petri net simulation environment (PNlib, PNproBio, AMMod)

Drawbacks of other Petri net tools presented in Chapter 2	**New environment: PNlib, PNproBio, AMMod**
The underlying formalism is partly – not defined precisely, – not common, or – not completely.	The xHPN and the xHPNbio formalism (see Definition 4.68, Definition 5.1) have been developed which can represent nearly all kinds of biological processes and serve as specification for the implementation of the PNlib.
The definitions of processes essential for simulation are partly – not common, – not precise enough, – not complete enough to cover all possible conflict situations that could occur during simulation.	All processes have been defined precisely in Chapter 4 and all possible conflict situations which could occur during simulation of an xHPN are trapped. Hence, the simulation yields reliable results.
The possibilities for post-processing simulation results are limited.	The connection to Matlab/Simulink, described in Section 6.3 , enables using the whole power of Matlab for post-processing simulation results. Several mathematical methods have been already implemented and summarized in the tool AMMod.
Partly, there is no graphical user interface and no animation possibilities.	The Modelica tool Dymola provides a user-friendly graphical user interface for modeling with the libraries PNlib and PNproBio (see Section 6.2). The models can either be established graphically or textually. If the model is constructed graphically, the textual description is generated automatically. Additionally, the models can be animated which can be controlled using several settings (see Section 6.2). Furthermore, the PNlib can be easily integrated into any other network modeling tool; the biological network modeling tool VANESA already uses the PNlib for simulation biological networks (see Section 8).

Partly, no hierarchically modeling is supported and the number of predefined biological components is limited.	The library PNproBio is based on the PNlib and provides several wrappers which represent specific biological processes and reactions. These sub-models can be reused several times in the same or in different models which offers on the one hand an easy-to-use model at the top level with an intuitive and familiar adapted biological view and on the other the flexibility and generality of the xHPN concept at a lower level (see Section 6.4).
Partly, no object-oriented modeling concept is used.	The object-oriented modeling concept by means of discrete, differential, and algebraic equations allows an easy way to maintain, extend, and modify the components of the PNlib (see Section 6.1).
Partly, the solvers and the corresponding settings are not changeable or not common.	The hybrid simulation is performed by using the Dymola-tool which comprises several possibilities to adapt the solver settings in order to achieve reliable simulation results (see Section 6.2).

7 APPLICATION

The modeling process described in Section 5 and depicted in Figure 6.24 as well as the application of the tools developed in this work, PNlib, PNproBio, and AMMod, is demonstrated exemplarily by the xanthan production of *Xanthomonas campestris* bacteria. Hereby, this chapter focuses on describing the general procedure of modeling and not on the selected example. Hence, no new insights about the bacteria and the xanthan production are presented due to using pseudo experimental data; but rather the usage of the developed environment is shown as well as the power and permit is proven.

The next sections are structured according to the steps of the modeling process in Figure 6.24. At first, the considered biological phenomena - the xanthan production of *Xanthomonas campestris* bacteria – is presented which founds the first step of the modeling process. Afterwards in step 2, known experiments and results concerning influence factors on growth and xanthan production are introduced. Additionally, known model approaches for growth and xanthan production are presented. To show that the methods work and no other side effects cause success or failure, pseudo experimental data generated by simulation have been used instead of real data from a wet lab. Based on the facts from literature, model hypotheses are formulated which should be represented by the models (step 3). Step 4, the procedure of preprocessing experimental data and relationship analysis, is performed by pseudo experimental data as previously mentioned in order to construct a simple unstructured model (step 5). Parameter estimation and sensitivity analysis – step 6 - are shown by a previously created unstructured model. The process optimization procedure – step 9 - is described by an already parameterized and verified metabolically structured model. Furthermore, all optimization methods introduced in Section 3.2 are applied for parameter estimation and process optimization to compare the performance and goodness of the achieved solutions. Finally, a chemically structured model has been established to perform deterministic and stochastic hybrid simulations which is step 7 of the developed modeling process.

7.1 Step 1: Biological Phenomena

The *Xanthomonas campestris* species are gram-negative, aerobe proteobacteria which are rod shaped with a size of $0.2 - 0.6 \times 0.8 - 2.9\ \mu m$. The genome contains typically 5 million base pairs (Vorhölter et al. 2008). *Xanthomonas campestris* bacteria are plant-pathogen. They harm more than 400 different kinds of plants and a further classification can be made into different pathovars based on the type of host plant which they attack (Sieber et al. 2006). *Xanthomonas campestris* pv. campestris, for example, infects crucifer plants like cabbage and cauliflower and causes the black rot disease.

Besides the plant-pathogenicity, *Xanthomonas campestris* cells produce the exopolysaccharide **xanthan**. Xanthan is widely used in industry for a number of reasons which include its pseudo plastic properties, i.e. xanthan solutions are very viscos but the viscosity decreases with increasing shear rate. It is used as a thickening agent, emulsifier, and stabilizer. In the 1980s xanthan was allowed as a food additive and assigned with the E number E-415. It is now an ingredient in many food products such as salad dressings, syrups, toppings, relishes, sauces, baked goods, and frozen foods. Besides, xanthan is used for pharmaceuticals such as creams and suspensions as well as for cosmetic products such as dentures, cleaners, shampoos, and lotions. Additionally, an important application of xanthan is in the petroleum industry; there, it is used in drilling fluids and in enhanced oil recovery processes (Garcia-Ochoa et al. 2000).

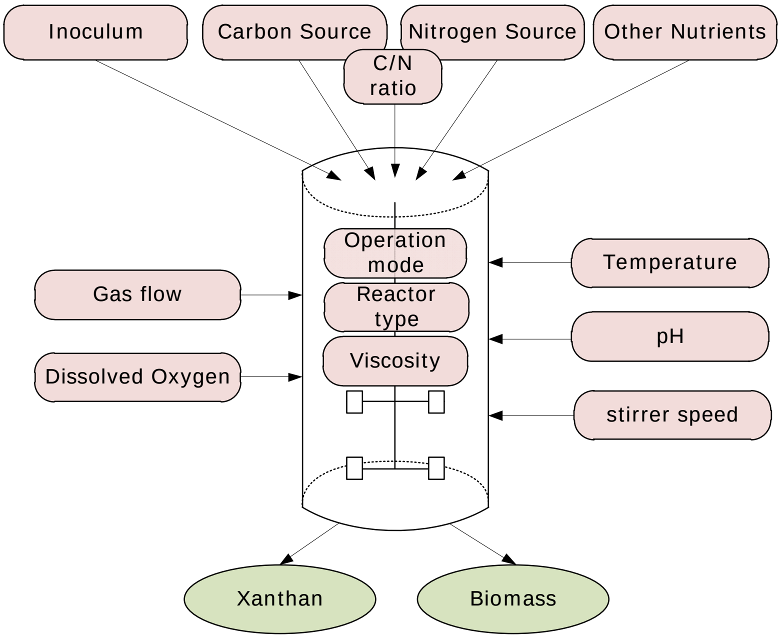


Figure 7.1: Factors that influence *Xanthomonas campestris* cell growth and xanthan production during the fermentation process

Several factors influence the growth of *Xanthomonas campestris* bacteria and their xanthan production during the fermentation process. Some of them are depicted in Figure 7.1. Within this study, a model has been established which can predict and optimize growth and xanthan production subject to the input factors: carbon source, nitrogen source, operation mode of the bioreactor, and temperature. The model is based on known phenomena from literature which are detailed hereafter. The predictions and suggested optimized input factors have to be validated by wet-lab experiments in future projects (see Chapter 8).

7.2 STEP 2: EXPERIMENTAL DATA AND PRIOR KNOWLEDGE

The beginning of every modeling process is founded by extensive investigation into the studied organism and the corresponding processes. The question is asked: Which factors influences growth and xanthan production of *Xanthomonas campestris* bacteria? And in addition: Are there already any models available? The results of this literature study are presented hereafter.

To show that the methods work and no other side effects cause success or failure, pseudo experimental data generated by simulation have been used instead of real data from a wet lab.

7.2.1 INFLUENCE FACTORS ON GROWTH AND XANTHAN PRODUCTION

The model should include the influence of the factors: carbon and nitrogen source (C/N ratio), viscosity, operation mode of the fermenter, and temperature. The proposed results of experiments in this juncture are summarized in this section.

CARBON SOURCE

Several authors have studied the influence of different carbon sources and different initial concentrations on growth and xanthan production. Souw and Demain suggested that glucose and sucrose are best for xanthan production with an initial concentration of 40 g/l (Souw and Demain 1979).

Leela and Sharma tested various types of sugars and obtained a maximum xanthan yield with glucose, sucrose, maltose, and starch (Leela and Sharma 2000). All these sugars were good enough for xanthan production but using glucose a slightly higher amount of xanthan was produced.

Funahashi et al. showed inhibition of growth and cessation of xanthan production in initial glucose concentrations greater than 50 g/l (Funahashi et al. 1987). This inhibition phenomenon was also verified by experiments done by (Amanullah et al. 1998). Additionally, Lo et al. pointed out that the specific growth rate and the cell yield from glucose are decreased as glucose concentration is increased (Lo et al. 1997). On the hand, higher glucose concentrations yielded a higher final xanthan concentration and, in addition, the xanthan yield from glucose and the specific xanthan production rate were also increased. Moraine and Rogovin detected no effect on the cell growth rate by varying the initial glucose concentration in the range of 5 g/l to 50 g/l (Moraine and Rogovin 1973).

Nitrogen Source

Furthermore, the influence of the nitrogen source on growth and xanthan production was investigated by several authors. Souw and Demain showed that high nitrogen concentrations inhibit xanthan production and stimulate growth and that the xanthan production is better with nitrogen limitation (Souw and Demain 1979). These results are verified by the experiments of Lo et al. which indicate that higher nitrogen concentrations give higher cell yields and specific growth rates but lower xanthan yields and specific xanthan production rates (Lo et al. 1997). They also found that the specific xanthan production rate depends on the initial glucose concentration.

C/N Ratio

Moreover, the effects of the carbon to nitrogen ratio (C/N ratio) are often studied. Lo et al. showed that the xanthan yield and the specific production rate increased with increasing C/N ratio while the cell yield and the specific growth rate decreased (Lo et al. 1997). Tait et al. also achieved a greater xanthan production in media which contained higher C/N ratios (Tait et al. 1986). Roseiro et al. reached a maximal xanthan production in nitrogen limitation at a C/N ratio of 23 (Roseiro et al. 1992).

Viscosity

Moraine and Rogovin showed that growth stops when the viscosity reached 2 $Pa\ s$ (Moraine and Rogovin 1973). They assumed that possibly a stagnant slime layer builds up around the cells as viscosity increases which causes resistance to the transport of nitrogen into cells to decrease and restrict the growth. Higher concentrations of nitrogen permits growth at higher viscosities due to the increased driving force for transporting nitrogen into the cells.

Additionally, they observed a decline of the specific product formation rate at high viscosities which can also be most likely explained by a slime layer around the cells. The layer could also restrict the rate of diffusion of nutrients into the cell; perhaps, it also controls a xanthan synthesis feedback mechanism. Increased shear may enhance its removal and improve the production rate.

Operation Mode

To avoid inhibition effects and enhance xanthan concentrations repeated batch, fed-batch, and continuous operation modes were proposed. Lo et al. studied batch and fed-batch cultures with best results for a two-stage batch fermentation with a low C/N ratio in the first stage and a high C/N ratio in the second stage. Thereby, the second stage begins at the end of the exponential growth phase (Lo et al. 1997). In such a manner, a fast cell growth is achieved during the exponential phase due to the low C/N ratio with a moderate nitrogen concentration and the high C/N ratio in the second stage causes an enhanced xanthan production. They also performed fed-batch fermentations in which additional glucose was added in five equal parts during the stationary phase so that the C/N ratio was kept low throughout the fermentation. Due to the low C/N ratio the xanthan production was poor even through cells grew well in the growth phase.

Funahashi et al. suggested controlling the glucose concentration at $30 - 40\ g/kg$ broth by intermittent additions of glucose to prevent inhibition of cell growth and cessation of xanthan production (Funahashi et al. 1987).

Vuyst et al. replaced the conventional fermentation process - a growth phase is followed by a xanthan production phase - by a two-step process (Vuyst et al. 1987). The two-step process comprises an initial biomass production step followed by a xanthan formation step which requires a high C/N ratio and higher aeration and agitation levels. The optimal time for changing from one step to the other is critical and occurred at $25 - 30\ h$ after inoculation. At this time, the content of one biomass production fermenter could be apportioned up to 10 xanthan production fermenter.

Amanullah et al. compared the biological performance of three fed-batch cultures, two-step glucose addition, multiple-step glucose addition, and continuous glucose addition, with two batch cultures with different initial glucose concentrations (Amanullah et al. 1998). They showed that the performance cannot be improved by increasing the initial glucose concentration above $50\ g/l$ nor by an intial glucose concentration of $40\ g/l$ followed by a single glucose addition of $10\ g/l$ while nitrogen is still present. However, when the nitrogen is exhausted, the two-step glucose addition and the continuous feeding strategy can enhance the performance compared to batch fermentations.

Temperature

Shu and Yang studied the influence of temperature on cell growth and xanthan formation. They found that higher temperatures lead to xanthan biosynthesis while lower temperatures favor cell growth (Shu and Yang 1991). Due to the fact that the optimum temperatures for cell growth and xanthan production are not the same, they suggest a two-stage fermentation with a temperature shift from 27°C to 32°C. Both xanthan yield and production rate were improved by this type of fermentation.

7.2.2 Modeling of Growth and Xanthan Production

Several kinetic models have been proposed to model *Xanthomonas campestris* growth and xanthan production. These models can be divided into structured and unstructured models as mentioned in Section 5.2.

Unstructured Models

At first, unstructured models should be considered which describe the bacteria as a black box and are used most frequently according to their simplicity and technical robustness. These models represent at least the evolution of biomass, carbon, and xanthan over time and often the development of the nitrogen source is included, too.

Moraine and Rogovin proposed an unstructured model for the xanthan production which takes the dependencies of growth and production on nutrients into account (Moraine and Rogovin 1966). In addition, Moraine and Rogovin proposed using the logistic equation to describe growth when the limited nutrient is not known

$$\frac{dX}{dt} = \mu_{max} X \left(1 - \frac{X}{X_{max}}\right), \qquad \text{Eq. 7-1}$$

whereby X is the biomass concentration, μ_{max} is the maximum growth rate, and X_{max} is the maximum biomass concentration.

Subsequently, Weiss and Ollis proposed a model which only depends on biomass concentration and its evolution over time (Weiss and Ollis 1980). Thereby, the growth is described by the logistic in Eq. 7-1 and the product formation is expressed by the **Luedeking-Piret equation**. The Luedeking-Piret equation models the production formation dependent on the current biomass concentration (X) and growth $\left(\frac{dX}{dt}\right)$ in a linear fashion

$$\frac{dP}{dt} = \alpha \frac{dX}{dt} + \beta\, X, \qquad \text{Eq. 7-2}$$

where P is the xanthan concentration and α and β are empirical constants which vary with fermentation conditions (pH, temperature, etc.). Hence, the term $\alpha \frac{dX}{dt}$ represents the growth associated part of the product formation and $\beta\, X$ the non-growth associated part. The convenience of this model is that β can be estimated from stationary phase data $\left(\frac{dX}{dt} = 0\right)$ and the parameter α can be fit throughout the early exponential phase. Weiss and Ollis assumed that xanthan production is both growth and non-growth associated $(\alpha \neq 0, \beta \neq 0)$.

Pinches and Pallent also modeled the growth with the logistic equation in Eq. 7-1 and showed independence of the parameter μ_{max} on the initial nitrogen concentration (Pinches and Pallent 1986). Product formation and substrate consumption are expressed by the Luedeking-Piret equation. Additionally, they added an equation for oxygen consumption which depends also on biomass and growth.

Quinlan pointed out that the use of the Luedeking-Piret equation for modeling xanthan production has the serious weakness that it neither explicitly nor implicitly depends on the substrate concentration (Quinlan 1986). This can be avoided by expressing the quasi-stoichiometric relationship between xanthan production and glucose consumption by the following differential equation

$$\frac{dP}{dt} = k_P \cdot S \cdot X, \qquad \text{Eq. 7-3}$$

where S is the glucose concentration and the rate constant k_p varies with physicochemical variables such as temperature, pH, dissolved oxygen, and stirring speed. In addition, they found a quasi-stoichiometric relationship between produced biomass and consumed nitrogen.

Shu and Yang integrated the effect of temperature on xanthan production and growth into their model (Shu and Yang 1991). They modeled the growth with the logistic equation in Eq. 7-1 and found that the parameter μ_{max} is highly dependent on temperature. They expressed this temperature effect by the **root-square model**

$$\mu_{max}(T) = \{C_1(T - T_{min})[1 - exp(C_2(T - T_{max}))]\}^2. \qquad \text{Eq. 7-4}$$

Xanthan production and glucose consumption are represented by the Luedeking-Piret model. The growth associated parameters were also modeled by the root-square model. The non-growth associated parameters were found to follow temperature dependency according to the **Arrhenius law**

$$\beta(T) = a_A\, exp\left(-\frac{E_A}{R_A \cdot T}\right) \qquad \text{Eq. 7-5}$$

Garía-Ochoa et al. mentioned that an unstructured kinetic model for describing xanthan production has to take into account biomass evolution as a function of the nitrogen source as it is a limiting nutrient (Garía-Ochoa et al. 1995). They showed in experiments that growth stops when nitrogen is exhausted and, therefore, they proposed the following modified logistic equation

$$\frac{dX}{dt} = \mu_{max}\left(\frac{X^0}{Y_{XN}} + N^0\right)X\left(1 - \frac{X}{X^0 + Y_{XN}N^0}\right), \qquad \text{Eq. 7-6}$$

where N^0 is the initial nitrogen concentration, $X^0$is the initial biomass concentration, and Y_{XN} is the yield of biomass from nitrogen. Additionally, they added the following differential equation to describe the evolution of dissolved oxygen (O_2)

$$\frac{dO_2}{dt} = \underbrace{k_L a \cdot (O_2^* - O_2)}_{oxygen\ mass\ transfer} - \underbrace{\left(m_O X + \frac{1}{Y_{XO}}\frac{dX}{dt}\right)}_{oxygen\ consumption}, \qquad \text{Eq. 7-7}$$

where O_2^* is the saturated value of the dissolved oxygen and m_O is the dissolved oxygen consumption coefficient. The equation includes a term for oxygen mass transfer and another for oxygen consumption for maintenance and growth.

All these models do not include the observed inhibition effect of high substrate concentrations and viscosity on xanthan production and growth previously mentioned. To integrate these effects, Mulchandani et al. introduced the following modified form of the logistic equation in Eq. 7-1 (Mulchandani et al. 1988)

$$\frac{dX}{dt} = \mu_{max} X \left(1 - \left(\frac{X}{X_{max}}\right)^{\theta}\right). \qquad \text{Eq. 7-8}$$

The constant $\theta > 0$ could be defined as an index of inhibitory effect, i.e. for very large values of θ the growth will follow an exponential pattern and a value close to zero describes complete inhibition. This modified logistic equation provides a means to describe growth inhibition caused by heat and mass transfer problems due to increased viscosity. In order to also consider the availability of substrate, the authors proposed a combination of the Monod equation and the modified logistic equation in Eq. 7-8

$$\frac{dX}{dt} = \mu_{max} \left(\frac{N}{K_N + N}\right) X \left(1 - \left(\frac{X}{X_{max}}\right)^{\theta}\right), \qquad \text{Eq. 7-9}$$

where N is the nitrogen concentration and K_N is substrate concentration at half maximum speed. Furthermore, Luong and Mulchandani proposed a generalized form of the Monod kinetics to consider the inhibition effect of glucose on growth (Luong and Mulchandani 1988)

$$\frac{dX}{dt} = \mu_{max} X \left(\frac{N}{K_N + N}\right) \left(1 - \frac{S}{S_{max}}\right)^{\delta}, \qquad \text{Eq. 7-10}$$

where S_{max} is the maximum glucose concentration above which growth is completely inhibited. Furthermore, several kinetics for substrate inhibition of other organisms have been proposed.

Structured Models

All mentioned unstructured kinetic models do not recognize the complex set of metabolic reactions which occur within the cell. Hence, they are not able to predict the dynamic behavior of cells when operational and external conditions are changed. Additionally, unstructured models can predict intracellular concentrations only if it is assumed that there is a constant fraction of the respective metabolite in the cell. Thus, their usage for understanding the cellular dynamics and the involved regulation mechanism is limited (Blanch and Clark 1997).

Structured kinetic models are needed to reveal the influence of process variables on kinetics and to offer a possibility to predict the behavior of the cell under different external conditions. Structured models can be divided according to (Garcia-Ochoa et al. 1998) into:

1. **Compartmental models**: Biomass is modeled by dividing it into a set of compartments such as DNA, RNA, proteins, hydrocarbons, etc.
2. **Metabolically structured models**: Growth is modeled unstructured but the carbon source metabolism in the cell is considered to describe the xanthan formation.
3. **Chemically structured models**: Nitrogen and carbon source metabolisms in the cell are taken into account to describe growth and xanthan production.

Compartmental Model

Garcia-Ochoa et al. proposed a compartmental structured model which describes the growth of *Xanthomonas campestris* bacteria by involving the influence of the initial nitrogen concentration (Garcia-Ochoa et al. 2004a). At first, they simplified the biomass pathway by lumping it into three different groups of macromolecules: DNA, RNA, and proteins. It is assumed that nitrogen is the limited substrate and other compounds that are necessary for growth, such carbon source and phosphate, are unlimited in availability. The ammonium metabolism is divided into non-forming bases amino acids synthesis and forming bases amino acids synthesis (see Figure 7.2). Based on the stoichiometric equations for this metabolic pathway, they proposed a model of differential equations. Thereby, the pseudo steady state is applied on forming bases amino acids and $\mathrm{RNA_I}$ and the reactions r_1, r_3, r_4, and $r_5$are expressed by the second order power law and r_2 by the first order power law.

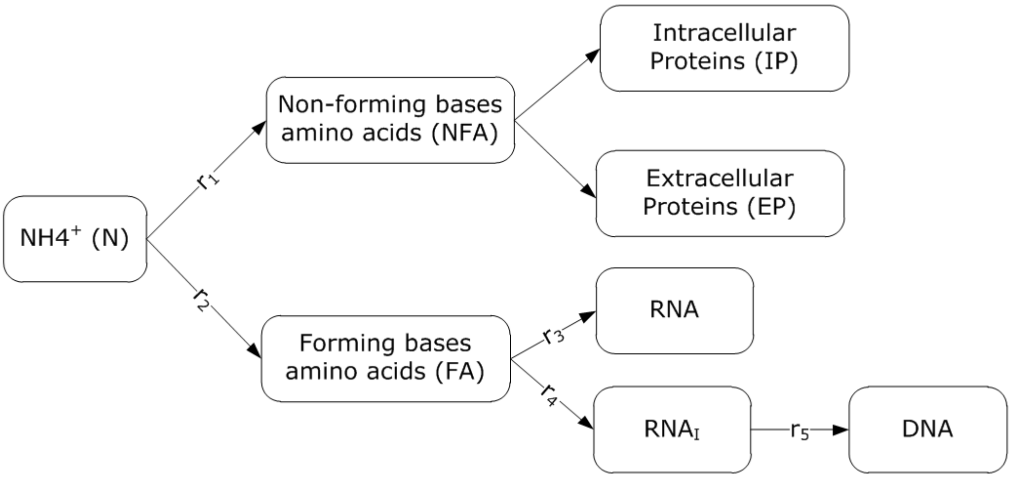

Intracellular Proteins (IP)
Non-forming bases amino acids (NFA)
Extracellular Proteins (EP)
NH4+ (N)
r1
r2
RNA
Forming bases amino acids (FA)
r3
r4
RNAI
r5
DNA

Figure 7.2: The simplified metabolic pathway for the components of biomass proposed by (Garcia-Ochoa et al. 2004a)

METABOLICALLY STRUCTURED MODEL

Pons et al. introduced a metabolically structured model which comprised an unstructured part in order to represent growth and a structured part to describe xanthan production, glucose consumption, and oxygen consumption according to a reaction network as a simplified scheme of the intracellular carbon metabolism (Pons et al. 1989).

Thereby, the growth is modeled by the logistic equation in Eq. 7-1. The structured part is based on the metabolic network in Figure 7.3 which represents the xanthan synthesis and the total catabolism of glucose, i.e. Entner-Doudoroff pathway and tricarboxylic acid cycle. The stoichiometric equations are derived from this metabolic network and their analysis leads to several assumptions.

Based on the model of Pons et al. and their corresponding assumptions, Garcia-Ochoa et al. developed a metabolically structured model with an unstructured part for growth described by Eq. 7-6 (Garcia-Ochoa et al. 1998). The stoichiometric relationships for total glucose catabolism, maintenance energy, and oxidative phosphorylation have been taken from (Pons et al. 1989) but the stoichiometry of xanthan production from glucose has been rearranged. Additionally, the authors pointed out some simplifications due to their observations. Furthermore, they studied the influence of temperature on the model parameters. Thereby, they expressed some parameters by the root-square model already mentioned in Eq. 7-4 and others were fitted to linear expressions.

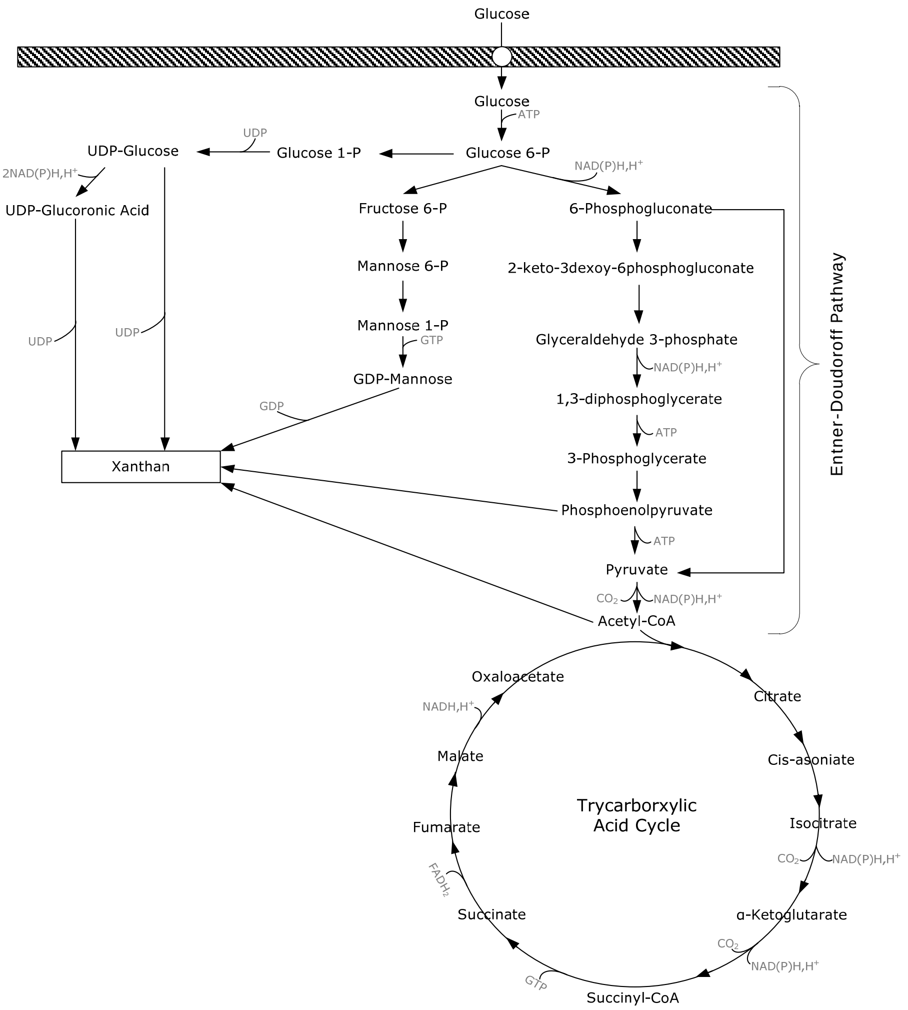
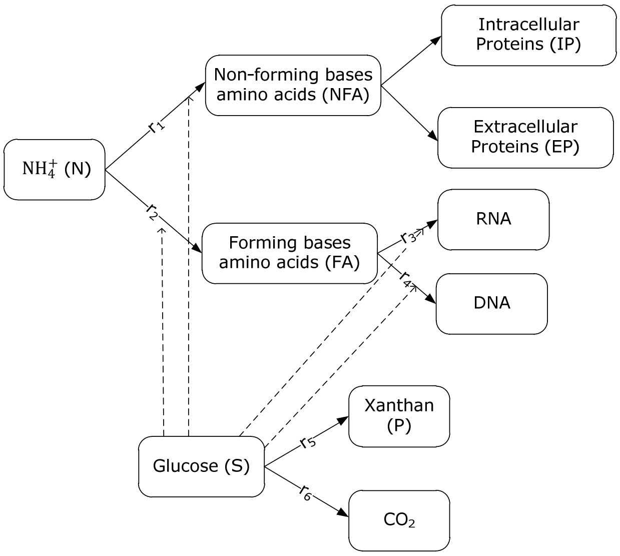


Figure 7.3: Coupling between xanthan biosynthetic pathway and glucose metabolism of *Xanthomonas campestris* bacteria

CHEMICALLY STRUCTURED MODEL

Garcia-Ochoa et al. also combined the compartmental model with the metabolically structured model to a chemically structured model (Garcia-Ochoa et al. 2004b, Garcia-Ochoa et al. 1996). This model considers the nitrogen metabolism for growth and the carbon metabolism for xanthan production. Figure 7.4 shows the assumed simplified reaction scheme. The coupling of the xanthan biosynthesis pathway and the glucose metabolism is displayed in Figure 7.3. The stoichiometric equations for the nine reactions are the same as the ones for the compartmental model and metabolically structured model as well as the kinetic equations and the model assumptions.

The authors studied, in addition, the influence of initial nitrogen concentration and temperature on the model parameters. They found no relationships between the initial nitrogen concentration and the model parameters and different relationships between the temperature and the model parameters: some parameters have a maximum, some parameters have a linear relationship, and the remaining do not change with temperature. The parameters with a maximum are described by the root-square model.


Intracellular Proteins (IP)
Non-forming bases amino acids (NFA)
Extracellular Proteins (EP)
NH_4^+ (N)
r_1
r_2
RNA
Forming bases amino acids (FA)
r_3
r_4
DNA
Xanthan (P)
Glucose (S)
r_5
r_6
CO_2

$$NADH, H^+ + 0.5O_2 \xrightarrow{r_7} NAD^+ + H_2O$$

$$FADH_2 + 0.5O_2 \xrightarrow{r_7} FAD^+ + H_2O$$

$$ADP + Pi \xrightarrow{r_8} ATP + H_2O$$

$$ATP + H_2O \xrightarrow{r_9} ADP + Pi$$

Figure 7.4: Simplified metabolic pathways for xanthan and biomass assumed by (Garcia-Ochoa et al. 2004b)

Modeling of Oxygen Mass Transfer Coefficient and Viscosity

The introduced metabolically and chemically structured models as well as the unstructured model from (Garcia-Ochoa et al. 1995) involve the evolution of oxygen dependent on the oxygen mass transfer coefficient k_La. Garcia-Ochoa and Gomez. studied the influence of stirrer speed (R), superficial gas velocity (V_S), and liquid effective viscosity $\left(\mu_{eff}\right)$ on the oxygen transfer rate and found a correlation between these variables and the oxygen mass transfer coefficient (Garcia-Ochoa and Gomez 1998)

$$k_La = C \cdot V_S^{\alpha} \cdot R^{\beta} \cdot \mu_{eff}^{\lambda}, \qquad \text{Eq. 7-11}$$

where the constant C depends on the geometry of the vessel and stirrer employed. Several approaches to model the viscosity μ_{eff} have been proposed (see García-Ochoa et al. 2000, Marcotte et al. 2001, Speers and Tung 1986, Xuewu et al. 1996). Marcotte and others, for example, suggested expressing the viscosity dependent on xanthan concentration P by the following three model equations (Marcotte et al. 2001)

$$\mu_{eff} = aP^b \tag{Eq. 7-12}$$

$$\mu_{eff} = a\,exp(bP) \tag{Eq. 7-13}$$

$$\mu_{eff} = 1 + aP + bP^2 \tag{Eq. 7-14}$$

and the temperature dependency was described by the Arrhenius model

$$\mu_{eff} = a_A\,exp\left(\frac{E_A}{R_A \cdot T}\right) \tag{Eq. 7-15}$$

where a_A is the frequency factor, E_A is the activation energy, and R_A is the gas constant.

7.3 Step 3: Model Hypotheses

The following table summarizes the hypotheses which should be represented partly by the established models.

Table 7.1: Model hypotheses

H1	Bacteria growth is limited by nitrogen.
H2	High glucose concentrations inhibit growth. Growth is totally inhibited if the glucose concentration exceeds 50 g/l.
H3	Xanthan production is partly growth associated.
H4	High nitrogen concentrations inhibit xanthan production. The xanthan production is totally inhibited if the nitrogen concentration exceeds 0.3 g/l.
H5	Glucose and oxygen are used for maintenance processes.
H6	The viscosity correlates with the xanthan concentration.
H7	High viscosities inhibit growth. The growth is totally inhibited if the viscosity exceeds 2 Pa s.
H8	Growth depends on temperature.
H9	Xanthan production depends on temperature.

H10	Xanthan production is limited by oxygen.
H11	Glucose is used for growth and xanthan production.
H12	Nitrogen is used for growth.

7.4 Step 4 and 5: Preprocessing and Relationship Analysis

Preprocessing of experimental data and the subsequent relationship analysis are the first steps of the modeling process after hypotheses have been formulated and the corresponding experiments have been completed (see Figure 6.24). The procedure is performed in this section for pseudo experimental data of a fermentation process of *Xanthomonas campestris* bacteria. Pseudo experimental data are data gained from simulations and not from the wet lab. In this manner, it can be shown that the methods work and no side effects influence success or failure. Thereby, the concentrations of glucose (S), nitrogen (N), xanthan (P), and biomass (X) have been measured at several points in time. At first, the experimental data are preprocessed in Matlab by smoothing splines to get the development of the concentrations at each point in time and the corresponding derivatives with respect to time. Based on these results, the relationships can be studied with the purpose of establishing a simple unstructured model of the xanthan production.

The RA is performed by the following four steps.

Step 1: Relationship between growth rate μ and substrates

It is assumed that the increase of biomass can be described by the following differential equation

$$\frac{dX}{dt} = \mu \cdot X. \qquad \text{Eq. 7-16}$$

Since all derivatives are approximated by smoothing splines in the preprocessing phase, the growth rate can be calculated by rearranging Eq. 7-16

$$\mu = \frac{1}{X} \cdot \frac{dX}{dt} = \frac{d\,ln(X)}{dt}. \qquad \text{Eq. 7-17}$$

Thereby, it has to be preferred to take the derivative of the logarithmic biomass concentration evaluated by smoothing splines based on logarithmic measurements. In this manner, numerical difficulties due to small values of X can be avoided.

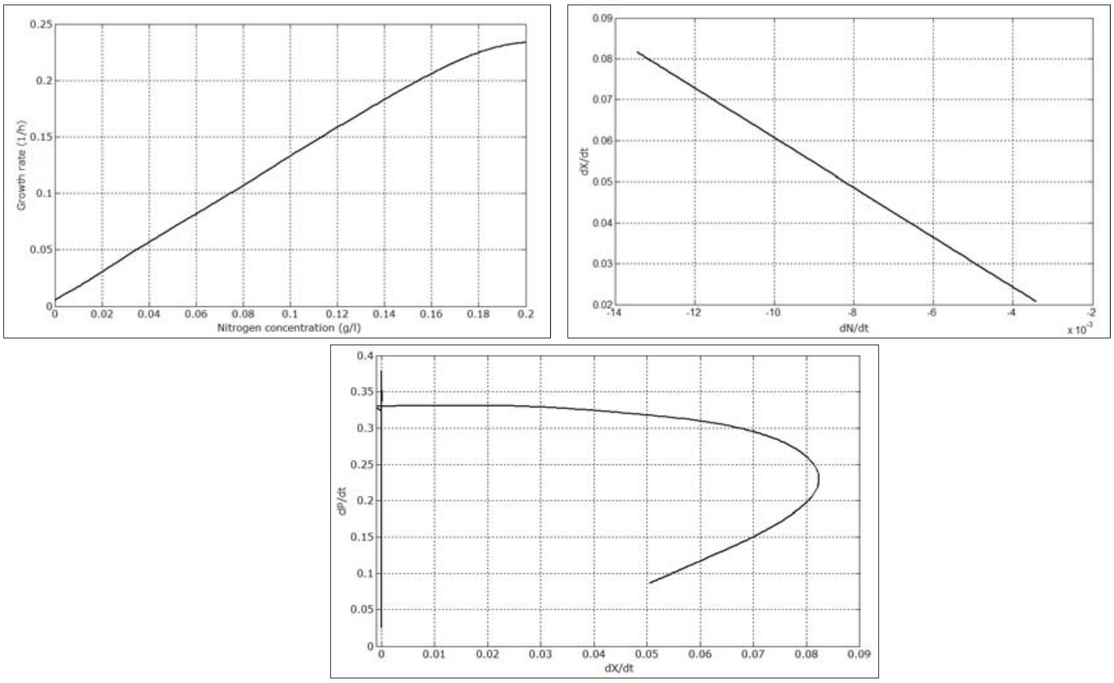


Figure 7.5: upper left: growth rate (μ) versus nitrogen concentration (N), upper right: the relationship between growth and nitrogen consumption, bottom: the relationship between growth and product formation over the whole fermentation time

Furthermore, it assumed that nitrogen is the limited substrate because glucose is still available when the bacteria stop growing and nitrogen is exhausted at this time (**H1**, Table 7.1). The growth rate is plotted against the nitrogen concentration to identify a functional relationship (see Figure 7.5 upper left). The plot shows a nearly linear relationship which can be described either by

$$\mu = k \cdot N \qquad \text{Eq. 7-18}$$

or by Monod kinetics with a much higher Monod constant than the initial nitrogen concentration

$$\mu = \frac{\mu_{max} \cdot N}{K_N + N}. \qquad \text{Eq. 7-19}$$

The smoothed data are fitted to both functions. Both functions reach good fittings with slightly better results for Monod kinetics so that the Monod kinetics is taken to describe the growth dependent on the nitrogen concentration.

Step 2: Relationship between growth rate μ and nitrogen consumption rate q_N

It is assumed that there is a relationship between the growth rate and the consumption rate of nitrogen (**H12**, Table 7.1). To detect this relationship, the derivative of X is plotted against the derivative of N in the time of growth. This plot shows a linear correlation between the rates (see Figure 7.5 upper right). Hence, the nitrogen consumption rate and growth rate differ from each other only in a constant denoted by Y_{XN} which is the yield of biomass from nitrogen. The value of this coefficient is estimated to 6.0745 g biomass/g nitrogen.

Step 3: Production rate of xanthan

It is assumed that the production of xanthan can be described with the Luedeking-Piret equation (**H3**, Table 7.1)

$$\frac{dP}{dt} = \alpha \frac{dX}{dt} + \beta\, X \qquad \text{Eq. 7-20}$$

with a growth associated term $\alpha \frac{dX}{dt}$ and a non-growth associated term $\beta\, X$. If the product formation is totally growth associated, then $\beta = 0$ and if it is totally independent from bacterial growth, then $\alpha = 0$. The data shows that the production also continues when growth stops so that production is not totally growth associated, i.e. $\beta \neq 0$. On the other hand, if the production is totally non-growth associated, there will be a linear correlation between the change of xanthan and biomass over the whole fermentation time; but Figure 7.5 (bottom) shows clearly that no linear correlation exists, i.e. $\alpha \neq 0$.

At first, the coefficient β is estimated from the xanthan data of the stationary phase when growth stops $(dX/dt = 0)$ because then Eq. 7-20 simplifies to

$$\frac{dP}{dt} = \beta\, X \;\Leftrightarrow \beta = \frac{1}{X_{max}} \cdot \frac{dP}{dt}. \qquad \text{Eq. 7-21}$$

When the assumption is true, xanthan has to increase linearly in the stationary phase, i.e. $dP/dt = const.$ To consider this fact, the measurements are interpolated additionally with linear functions to avoid misadaption due to the curvature of the cubic smoothing splines. Then Eq. 7-21 leads to the constant value $\beta = 0.2471$.

Afterwards, the value of the parameter α can be obtained from data of the exponential growth phase by

$$\alpha = \frac{\frac{dP}{dt} - \beta X}{\frac{dX}{dt}} = 0.8023. \qquad \text{Eq. 7-22}$$

Step 4: Glucose consumption rate

It is assumed that glucose is used for growth and xanthan production and described by the following differential equation (**H11**, Table 7.1)

$$\frac{dS}{dt} = -\frac{1}{Y_{XS}}\frac{dX}{dt} - \frac{1}{Y_{PS}}\frac{dP}{dt} = -\frac{1}{Y_{XS}}\mu X - \frac{1}{Y_{PS}}\left(\alpha\frac{dX}{dt} + \beta X\right). \qquad \text{Eq. 7-23}$$

To estimate the coefficients Y_{XS}, yield of biomass from glucose, and Y_{PS}, yield of xanthan from glucose, the differential equation is rearranged to

$$\frac{dS}{dt} = -\left(\frac{1}{Y_{XS}} + \frac{1}{Y_{PS}}\alpha\right)\frac{dX}{dt} - \frac{1}{Y_{PS}}\beta X = -A\frac{dX}{dt} - BX. \qquad \text{Eq. 7-24}$$

Hence, the new parameters A and B can be determined in the same manner as described in step 3 for xanthan production due to the fact that Eq. 7-23 has been transformed to the Luedeking-Piret equation. Afterwards, the found values for A and B can be transformed back to the yield coefficients

$$Y_{PS} = \frac{\beta}{B} = 0.7734$$
$$Y_{XS} = \frac{Y_{PS}}{AY_{PS} - \alpha} = 0.1692. \qquad \text{Eq. 7-25}$$

Based these results, a simple unstructured model for the xanthan production of *Xanthomonas campestris* bacteria can be established by means of the PNlib and the PNproBio in Dymola (see Figure 7.6). Thereby, glucose, nitrogen, xanthan, and biomass are modeled by continuous places, the growth is represented by the Monod wrapper of the PNproBio library (see Section 6.4), and the xanthan production is modeled by the Luedeking-Piret wrapper. The fermentation is performed in a batch-mode due to the fact that no additional feedings have been recognized in the experimental data.

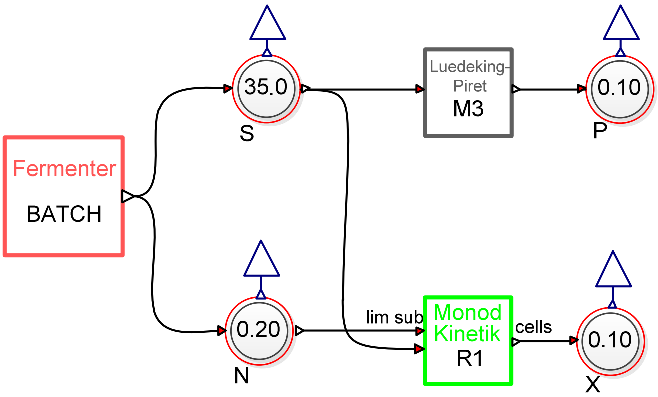


Figure 7.6: Simple unstructured model of the xanthan production of *Xanthomonas campestris* bacteria

The parameters are set to the found values and the model is simulated. The results show a good agreement between the pseudo experimental data and simulation results. The error sum could be possibly reduced by parameter estimation using the Nelder-Mead simplex method with the found values as the starting point. A local method is applied due to the assumption that the found values are in the neighborhood of the global minimum. The error sum could be further reduced and nearly the same parameter values are found that are used for generating the pseudo experimental data. The relative errors between original and optimized parameters are negligible.

7.5 Step 6 und 8: Parameter Estimation and Sensitivity Analysis

An unstructured model of the xanthan production of *Xanthomonas campestris* bacteria has been constructed by means of the PNlib and PNproBio. The model is depicted in Figure 7.7. It describes the evolution of glucose (S), nitrogen (N), oxygen ($O2$), xanthan (P), and biomass (X) over time with the initial nitrogen concentration, the initial glucose concentration, the fermentation operation mode, the gas flow rate, the stirrer speed, and the temperature as input factors. The model has been established based on facts from literature summarized in Section 7.2. Several equations have been found in different papers in this literature study which can be now combined within a xHPNbio model.

The growth is modeled by Monod kinetics with nitrogen as a limited substrate (**H1**, Table 7.1) combined with the term $(1 - S/S_{max})$ to express the inhibition phenomenon at high glucose concentrations (**H2**, Table 7.1); thereby, $S_{max} = 50$. Besides nitrogen, glucose and oxygen are also consumed during growth. The growth is modeled by the wrapper Rx (green box in Figure 7.7).

The xanthan production is described by the Luedeking-Piret equation in Eq. 7-2 combined with the term $(1 - N/N_{max})$ to represent the observed inhibition effect on xanthan production at high nitrogen concentrations (**H4**, Table 7.1); thereby, $N_{max} = 0.3$. This is modeled by the wrapper Rp (grey box in Figure 7.7). Furthermore, glucose and oxygen are used for maintenance processes, (**H5**, Table 7.1) which are described by the wrappers Rms and Rmo (olive-green boxes in Figure 7.7).

The oxygen mass transfer is modeled by the continuous transition TO with the maximum speed function $k_La \cdot (O_2^* - O_2)$ as previously mentioned in Eq. 7-7. Thereby, the oxygen mass transfer coefficient k_La is expressed by the equation Eq. 7-11 with $C = 3.08 \cdot 10^{-3}$, $\alpha = 0.43$, $\beta = 1.75$, and $\lambda = -0.39$ as suggested in (García-Ochoa et al. 1995), and $O_2^* = 0.0002$. It is assumed that the viscosity correlates with xanthan concentration modeled by Eq. 7-12 with $a = 0.08$ and $b = 0.3$ (**H6**, Table 7.1). The fermentation is performed as a batch mode, i.e. glucose and nitrogen are added to the fermenter only at the beginning of the experiment.

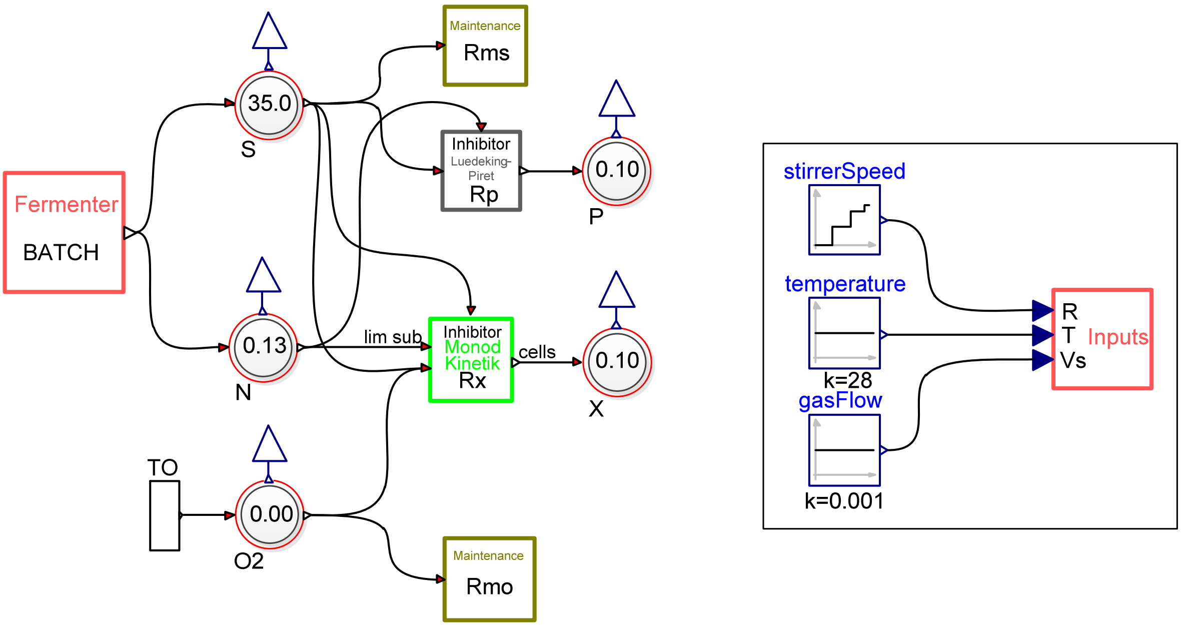


Figure 7.7: Unstructured model of xanthan production of *Xanthomonas campestris* bacteria

The input factors which influence the parameters, such as stirrer speed, temperature, and gas flow, can be determined by selecting one of the input components (Constant, Step, Ramp, Steps, or conditional Steps) and connecting them with the `Input` wrapper. In this manner the stirrer speed, the temperature, and the gas flow rate can be accessed by `Input.R`, `Input.T`, and `Input.Vs`, respectively. The stirrer speed, for example, is necessary to calculate the oxygen mass transfer coefficient k_La and influences the oxygen mass transfer in the fermenter.

To check the performance of PE and SA methods, five pseudo experimental data sets are generated by the unstructured model with different initial glucose (S^0) and nitrogen concentrations (N^0) (see Table 7.2) while stirrer speed, temperature, and gas flow rate are always the same. The stirrer speed begins with 3.3 rps and is increased in two steps to 9.2 rps, the temperature is a constant 28°C, and the gas flow rate is a constant 0.001 m/s. The initial values of biomass, xanthan, and oxygen are 0.1 g/l, 0.1 g/l, and 0.00002 mol/l, respectively. The data sets are listed in Table A6 - Table A10 (Appendix A4-1).

Table 7.2: Initial nitrogen and glucose concentrations for generating five pseudo experimental data sets

Run	N^0	S^0
1	0.13 g/l	35 g/l
2	0.25 g/l	30 g/l
3	0.28 g/l	45 g/l
4	0.28 g/l	20 g/l
5	0.12 g/l	30 g/l

Table 7.3 summarizes the model parameters to be estimated based on the five data sets. The objective function Q_{ss} in Eq. 5-11 is used for the optimization problem with the weightings

$$w_{i,j}^{k} = \left(\frac{1}{max\left\{\hat{y}_{i,j}^{k}, i = 1, \dots, n_t\right\}}\right)^2, k = 1, \dots, n_d, i = 1, \dots, n_t, j = 1, \dots, n_y \qquad \text{Eq. 7-26}$$

to normalize the contribution of each residual. The weightings are also listed in the appendix (see Table A11, Appendix A4-1).

Table 7.3: Model parameters of the unstructured model in Figure 7.7

Name	Description	Minimum Value	Maximum Value
YXN	Yield of biomass from nitrogen	3	12
YXS	Yield of biomass from glucose	0.01	1
YPS	Yield of xanthan from glucose	0.3	1
YXO	Yield of biomass from oxygen	1	300
mumax	Maximum specific growth rate	0.01	5
Kn	Monod constant for growth	0.1	10
alpha	Growth associated constant of Luedeking-Piret equation for xanthan production	0.1	10
beta	Non-growth associated constant of Luedeking-Piret equation for xanthan production	0.1	10
ms	Maintenance coefficient of glucose	0.01	1
mo	Maintenance coefficient of oxygen	0.00001	0.1

Local Sensitivities

At first, the sensitivities of the parameters are calculated locally at the original parameters by evaluating the normalized sensitivity matrix in Eq. 3-43. This analysis yields a 275×10 matrix with 2750 elements that have to be compared. To facilitate this, the absolute matrix elements are plotted in Figure 7.8 by representing the high sensitivities as black and low sensitivities as white. The plot indicates that the glucose concentration (S) is highly sensitive to changes in the parameters Y_{XN}, Y_{XS}, Y_{PS}, μ_{max}, K_N, β, and m_s at almost all points in time and in almost all data sets. The oxygen and nitrogen concentrations are less sensitive to changes in all parameters. Additionally, the parameters Y_{XO}, α, and m_O have less influence on all observed concentrations and the parameter m_S have a high influence only on the glucose concentration and less on the others.

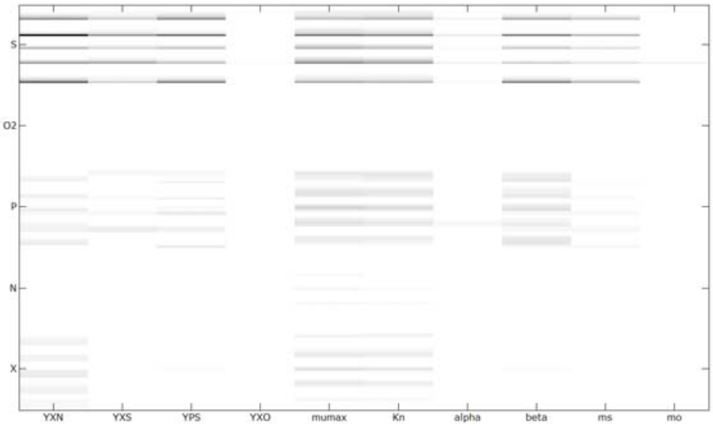


Figure 7.8: Plot of the absolute values of the normalized sensitivity matrix. Black fields correspond to high sensitivities and white fields to low sensitivities.

Principal Component Analysis

Based on the normalized sensitivity matrix, a principal component analysis has been performed which yields the following eigenvalues λ for the principal components ψ

Principal Component	ψ_1	ψ_2	ψ_3	ψ_4	ψ_5	ψ_6	ψ_7	ψ_8	ψ_9	ψ_{10}
λ	3.75E-04	1.26E-03	9.29E-02	1.87E-01	1.36E+00	2.37E+00	1.16E+01	1.95E+01	5.83E+01	1.02E+03

The eigenvectors reveal the contribution of each original parameter to the principal components (see Figure 7.9). This analysis reveals that the objective function value is negligibly sensitive to changes in parameter Y_{XO}, α, and m_O because these parameters belong only to the principal components with small λ values. On the other hand, the objective function is extremely sensitive to changes in parameter Y_{XN}, Y_{XS}, μ_{max}, K_N, and β because these parameters contribute large parts to principal components with high λ values.

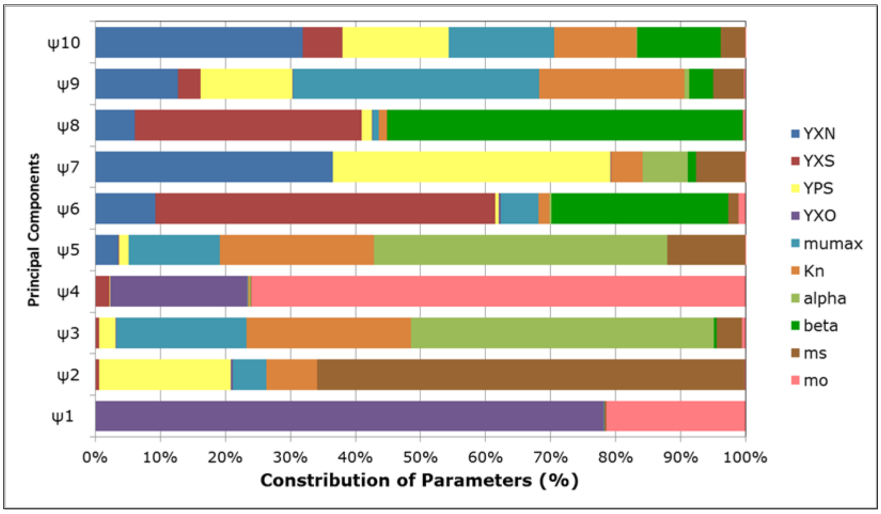


Figure 7.9: Contribution of the original model parameters to the principal components

Global Sensitivities

However, these local sensitivities are only valid in a small neighborhood of the actual parameter values. To reveal the effect of changes over the whole parameter range, global sensitivity analysis methods have to be applied. Therefore, first-order sensitivity coefficients and total-order sensitivity coefficients are calculated by the means of the eFAST method (see Section 3.3.3) with 593 samples and 2 resamples for each parameter ($N = 593, N_r = 2$).

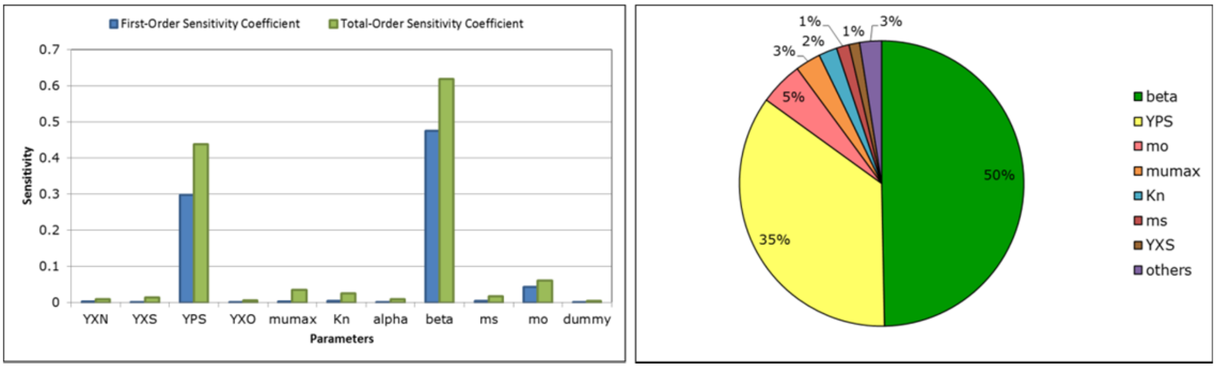


Figure 7.10: eFAST sensitivity coefficients of the parameters of the unstructured model in Figure 7.7. Right: first and total-order sensitivity coefficients, left: contribution of each parameter to the variance of the objective function value according to total-order sensitivity coefficients (others=$\{Y_{XN}, Y_{XO}, \alpha\}$).

The eFAST method generates small but non-zero sensitivity coefficients for parameters to which the model is completely independent (Marino et al. 2008). To reveal this effect a dummy parameter is added to the model which does not appear in the model equations nor does it affect the model in any way so that the sensitivity coefficient has to be zero. The dummy parameter of the unstructured model has a first-order sensitivity of 0.00006 and a total-order sensitivity 0.0055 derived from aliasing and interference effects. It is assumed that all parameters with a total-order sensitivity less than that of the dummy parameter are not significantly different from zero. However, no parameter has a total-order sensitivity less than that of the dummy parameter so it is assumed that all sensitivities are significantly different from zero. The results of the SA are depicted in Figure 7.10 and reveal that the parameters Y_{PS} and β have a much higher sensitivity than the other parameters. These two parameters contribute about 85% of the variance of the objective function value. The parameters Y_{XN}, Y_{XO}, and α are negligibly sensitive to the objective function because they each contribute less than 1% to the variance of the objective function value.

PARAMETER ESTIMATION

The PE is performed by all optimization algorithms introduced in Section 3.2 to compare the performance and the achieved solutions. Thereby, the following four hybrid methods are applied:

– **Hybrid1**: ES and Nelder-Mead simplex method (NMS),
– **Hybrid2**: ES and Hooke-Jeeves method (HJ),
– **Hybrid3**: CMAES and NMS, and
– **Hybrid4**: CMAES and HJ.

The method specific parameters are listed in Table A12 (Appendix A4-2). To improve the optimization procedure, all parameters are scaled to the interval [0,1] by the defined minimum and maximum values in Table 7.3. For the simulation, the parameters are re-scaled by the equation

$$\wp_{sim} = \wp^{l} + (\wp^{u} - \wp^{l}) \cdot \wp_{opt}. \qquad \text{Eq. 7-27}$$

Furthermore, the objective function Q_{ss} in Eq. 5-11 is used with the weightings in Eq. 7-26 to normalize the contribution of each residual.

At first, all parameters are optimized regardless of their sensitivities. The results of the applied methods are summarized in Figure 7.11 (left) and reveal the best results for the methods Hybrid2 and Hybrid4. Both methods reach estimators with objective function values less than 10^{-6} while the other methods only achieve objective function values higher than 10^{-3}.

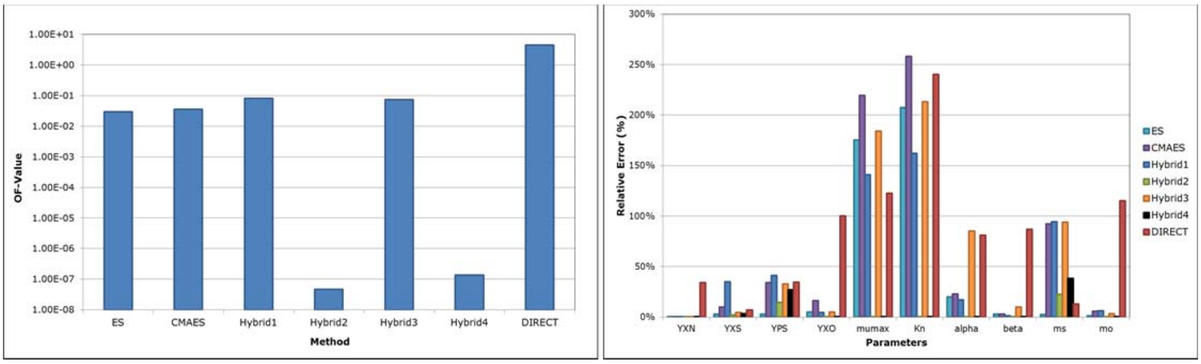


Figure 7.11: left: PE results of the unstructured model in Figure 7.7; right: Relative error (%) of the estimated parameters

The estimator delivered by ES and CMAES could always be improved by local methods. Thereby, HJ seems to work much better than NMS. But the estimators found in the first phase of the hybrid algorithms underlie random mechanisms so that the assumption was checked by running HJ with the start values generated in the first phase of Hybrid1 and Hybrid3 and running NMS with the start values generated in the first phase of Hybrid2 and Hybrid4. The results verify the assumption and show that HJ always achieves good estimators with objective function values less than 10^{-6} while NMS improve the estimators produced in the first phase slightly. MNS always stopped at a non-optimum solution which is not even a stationary point of the objective function. But HJ requires more than double computing time to find the estimators as NMS. Hence, the HJ method should be used preferably even if it needs many more function evaluations. Additionally, it is advisable to take CMAES for the first phase because this method achieves similar results to ES but in less than the half computing time. An advantage of both - ES and CMAES - is that they can be easily parallelized so that the problem can be handled by parallel sessions. In this way, models with more parameters as shown within this study can be parameterized. Further studies aim at the parallelization of the hybrid optimization methods (see Chapter 8). Furthermore, the DIRECT method completely fails in achieving acceptable parameter estimators in an appropriate time, which was also observed by Moles et al. who performed a PE for a pathway model with 36 parameters (Moles et al. 2003).

The relative errors of the parameter estimates are displayed in Figure 7.11 (right) and show highest variations (more than 100%) for the parameters μ_{max} and K_N. Only the methods

Hybrid2 and Hybrid4 reach estimators with a relative error less than 0.1%. Further studies indicated that the minimum of this optimization problem lies in a narrow valley so that it could be assumed that some of the applied algorithms are stuck due to the flat shape of the objective function in that region.

Fixed Parameter Estimation

In agreement with the results of the global sensitivity analysis, the parameters Y_{XN}, Y_{XO}, and α seem to have less influence on the objective function value and should be fixed to reduce the dimension of the search space and improve the optimization procedure. The fixed PE has been performed with same settings as the previous PE which are listed in Table A12 (Appendix A4-2). The results show that all hybrid algorithms converge to the right solution and the objective function values are much better compared to those before fixing less sensitive parameters. Additionally, fewer function evaluations and, thus, less computing time was needed to achieve these solutions. However, the results of ES and CMAES could not be improved by fixing less sensitive parameters. They still converge to the wrong solution, perhaps due to the location of the minimum in a narrow valley. Furthermore, the DIRECT method achieves a much better solution as before but the solution still has the worst objective function value and the computing time that is needed to reach this solution is also the highest of the applied methods.

7.6 Step 9: Process Optimization

Besides the unstructured model, a metabolically structured model has been constructed by means of the PNlib and PNproBio to optimize the xanthan yield of the fermentation process (see Figure 7.12). Thereby, the growth of the *Xanthomonas campestris* bacteria is modeled unstructured by the continuous transition T_x with $\mu_{max} \cdot N \cdot X$ as maximum speed function combined with two additional terms to express the mentioned inhibition effects of high glucose concentrations $(1 - S/S_{max})$ and high viscosities $\left(1 - \mu_{eff}/\mu_{eff,max}\right)$

$$r_X = \mu_{max} \cdot N \cdot X \cdot \left(1 - \frac{S}{S_{max}}\right) \cdot \left(1 - \frac{\mu_{eff}}{\mu_{eff,max}}\right) \qquad \text{Eq. 7-28}$$

(**H1, H2, H7**, Table 7.1). Nitrogen, glucose, and oxygen are consumed to produce biomass and, hence, the places N, S, and $O2$ are the inputs of the transition T_x with the reciprocal of the yield coefficients as arc weights $(1/Y_{XN}, 1/Y_{XS}, 1/Y_{XO})$. The xanthan production is modeled structured by the transitions $T1, T2, T3,$ and $T4$. The reaction rate of the xanthan production is expanded by the term $(1 - N/N_{max})$ to describe the inhibition effect of high nitrogen concentrations on xanthan production (**H4, H10**,. Table 7.1).

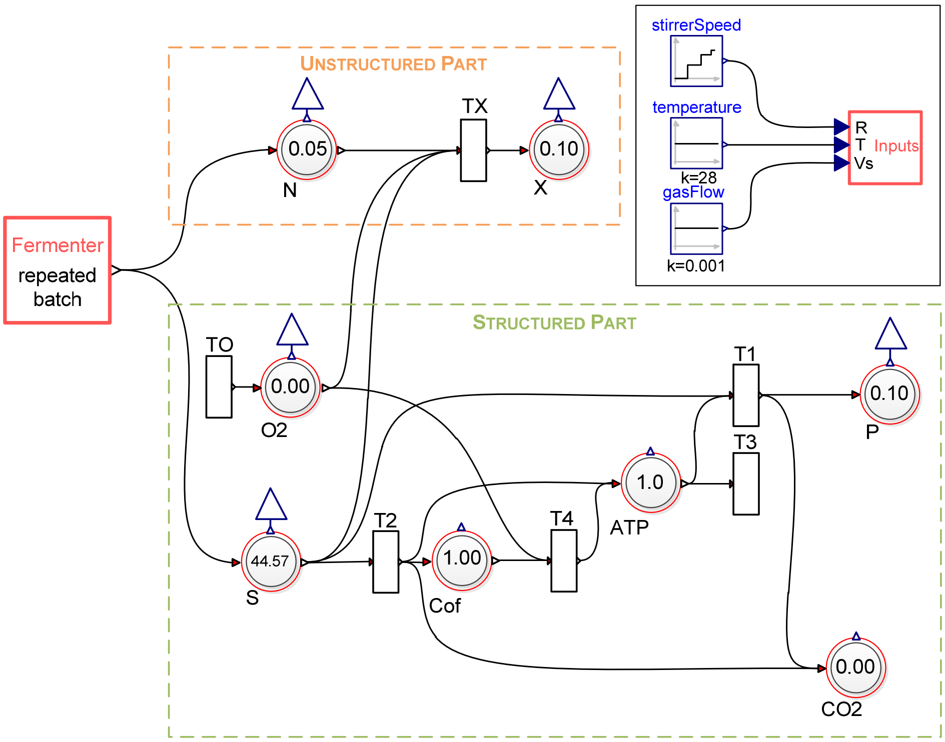


Figure 7.12: Metabolically structured model of xanthan production of *Xanthomonas campestris* bacteria

Some of the model parameters are modeled temperature dependent by the equations mentioned in Section 7.2. Additionally, the parameter μ_{max} in Eq. 7-28 depends on the temperature which is modeled by the root-square model as proposed in (Garcia-Ochoa et al. 2000) (**H8, H10**, Table 7.1). The transition T_O represents the oxygen mass transfer using the oxygen mass transfer coefficient k_La of equation Eq. 7-11 with $C = 3.08 \cdot 10^{-3}$, $\alpha = 0.43$, $\beta = 1.75$, and $\lambda = -0.39$ as suggested in (García-Ochoa et al. 1995) and $O_2^* = 0.0002$. Thereby, it is assumed that viscosity correlates with xanthan concentration modeled by Eq. 7-12 with $a = 0.08$ and $b = 0.3$ (**H7**, Table 7.1). The red fermenter wrapper models the operation mode of the fermentation, i.e. when and with which concentration glucose and nitrogen are added to the fermenter. The input factors which influence the parameters (stirrer speed, temperature, and gas flow) can be determined by selecting one of the input components (Constant, Step, Ramp, Steps, and conditional Steps) and connecting them with the `Input` wrapper. In this manner, the stirrer speed, temperature, and gas flow can be accessed by `Input.R`, `Input.T,` and `Input.Vs`, respectively. All components of the metabolically structured model are explained in detail in Table A15 and the parameters are listed in Table A16 (Appendix A4-3). Thereby, the values are taken from (Garcia-Ochoa et al. 1998) and (García-Ochoa et al. 1995).

The Optimization Problem

Based on this parameterized model, the xanthan yield should be maximized by varying the process parameters summarized in the following table between their minimum and maximum values.

Table 7.4: Process parameters to optimize of the metabolically structured model in Figure 7.12

Process Parameter	Description	Minimum Value	Maximum Value
$N0$	Initial nitrogen concentration	0.01	0.3
$S0$	Initial glucose concentration	10	50
$Nadd$	Nitrogen addition	0.01	1
$Sadd$	Glucose addition	5	70
ft	Feeding point in time	0	100
$temp0$	Initial temperature	25	34
tt	Time of temperature increase	0	100
$tempHeight$	Increase of temperature	0	10

Hence, the optimization problem formalized in Eq. 5-21 - Eq. 5-23 is solved by regarding two different optimization problems. The first optimization problem, denoted by **OP1**, only considers the yield of xanthan $P_f(z) = P\left(z, t_f^{fix}\right)$ at the end of the fermentation ($t_f^{fix} = 100\ h$) and has the objective function

$$\begin{gathered} Q_1 = -P_f(z, 100) \rightarrow min \\ z = (N^0, S^0, N_{add}, S_{add}, ft, temp_0, tt, tempHeight) \end{gathered} \qquad \text{Eq. 7-29}$$

while the objective function of the second optimization problem, denoted by **OP2**, considers also the time t_f when the maximum xanthan concentration $P_f\left(z, t_f\right)$ is reached due to the fact that fermentation time correlates with costs

$$Q_2 = -\frac{P_f(z, t_f)}{t_f} \rightarrow min \qquad \text{Eq. 7-30}$$

$$z = (N^0, S^0, N_{add}, S_{add}, ft, temp_0, tt, tempHeight).$$

These two optimization problems are solved by the optimization algorithms introduced in Section 3.2 with the settings of Table A13 (Appendix A4-2).

RESULTS OF OP1 AND OP2

The results for OP1 are displayed in Figure 7.13 and show that the achieved xanthan yields range between 225 and 230 g after 100 h fermentation time. Thereby, the best solution was found by ES but the required time is high compared to the hybrid methods.

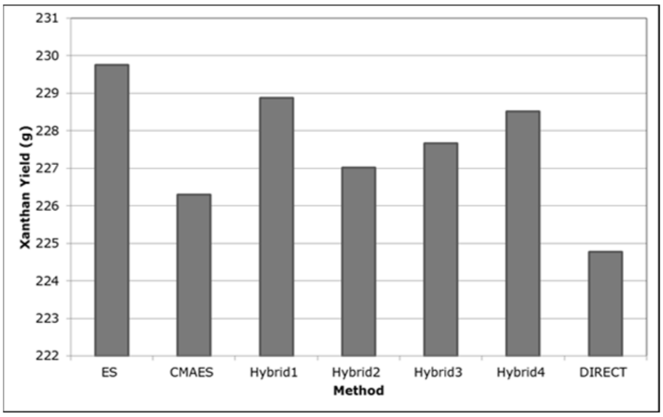


Figure 7.13: OP1: Results of PO for the metabolically structured model in Figure 7.12 with the settings of Table A13 (Appendix A4-2) and $t_f = 100\ h$

The DIRECT method reached the worst solution in the highest computing time. All hybrid methods achieve good solutions in a moderate time. Both local optimization methods work well in the second phase of the hybrid algorithms in contrast to the unfixed PE of Section 7.5.

The second optimization problem OP2 additionally includes the point in time when the maximum xanthan yield is reached. The fermentation is stopped at this time. The results are displayed in Figure 7.14. Similar to OP1, OP2 reaches relatively equal values for the initial and feeding concentrations but here also the feeding point in time and the initial temperature is equal for all applied methods. Only the amount of the temperature increase and the time for the increase differ from each other. Additionally, the lower bound for the nitrogen addition and the upper bound for the glucose addition are reached. Furthermore, the maximum xanthan yield is plotted against the required fermentation time (see Figure 7.14 right). This plot indicates that no found solution dominates another solution, i.e. if the maximum xanthan yield is smaller compared to another solution, also the required fermentation time to achieve it is smaller and if the maximum xanthan yield is higher compared to another solution, the required time to achieve it is also higher.

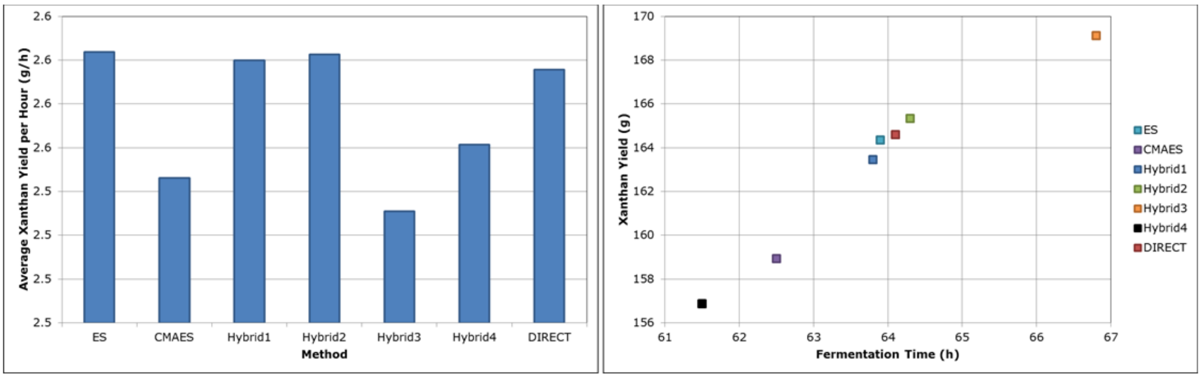


Figure 7.14: OP2: Results of PO for the metabolically structured model in Figure 7.12 with the settings of Table A13 (Appendix A4-2); left: average xanthan yields per hour; right: xanthan yield vs. fermentation time

Modifications of the Bounds for Glucose and Nitrogen Addition

To detect if even higher (average) xanthan yields can be reached, the lower bound of the nitrogen addition is set to 0 g/l, denoted by **Bound B** and the original bounds of Table 7.4 are denoted by **Bound A**. The corresponding optimization problems are denoted hereafter by **OP1A**, **OP1B**, **OP2A**, and **OP2B**. Afterwards, OP1 and OP2 are solved by setting the upper bound of the glucose addition to 1000 g/l and the lower bound of the nitrogen addition to 0 g/l, denoted by **Bound C**. The corresponding optimization problems are denoted following by **OP1C** and **OP2C**. The results are displayed in Figure 7.15 and show a negligible effect of setting the nitrogen addition to zero and a relative high effect of the increase of the glucose addition. The xanthan yield can be enhanced about 50 % and the average xanthan yield per hour is approximately 30 % improved.

The increase of the upper bound for the glucose addition causes a totally different growth curve (see Figure 7.16). In the case of OP1A and OP1B the biomass increases slightly over the whole fermentation and the same is true for xanthan production. The glucose is added at a point in time when it is still available and it is not exhausted at the end of the fermentation. However, biomass and xanthan production of OP2A and OP2B stop at the time when the glucose is exhausted. In the case of OP1C and OP2C, the bacteria only grow in the first 22-28 hours. In this phase nearly no xanthan is produced due to suppression by the high initial nitrogen concentration. After the addition of glucose, the xanthan production is enhanced and growth is suppressed by the high glucose concentration in the fermenter. Hence, the process optimization achieves the same two-phase fermentation approach as suggested in (Vuyst et al. 1987) and (Lo et al. 1997) mentioned in Section 7.2. In the first phase the bacterial growth is improved by using a low C/N ratio and in the second phase the xanthan production is enhanced by the high glucose concentration.

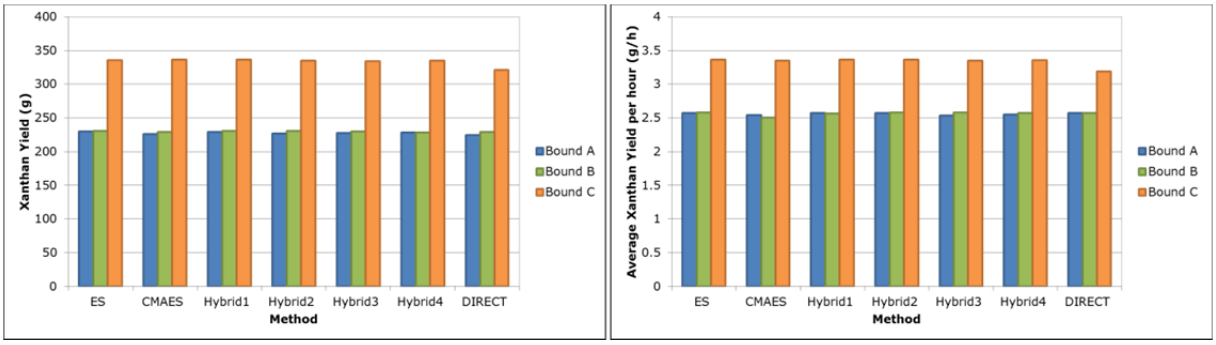


Figure 7.15: Left: xanthan yield after 100 h fermentation (OP1), right: average xanthan yield per hour (OP2) for the original bounds in Table 7.4 (Bound A), the lower bound of the nitrogen addition set to 0 g/l (Bound B), and the lower bound of the nitrogen addition set to 0 g/l and the upper bound of the glucose addition set to 1000 g/l (Bound C).

Additionally, the suggested point in time for changing over from growth to xanthan production can be verified by this PO. De Vuyst found that the changeover time is critical and suggested a time period of 25-30 hours after inoculation. OP1C yields a changeover time of 21-28 hours which depends on the applied optimization method and OP2C reaches changeover times of about 21 hours. Furthermore, Amanullah et al. found that the xanthan production can only be enhanced by glucose addition if the nitrogen is exhausted when the glucose is fed (Amanullah et al. 1998). This is also verified by this PO. Nitrogen is exhausted in all cases when glucose is added and nearly no nitrogen is added.

Moreover, all methods achieve times for the temperature shift which are nearly exactly the same as the feeding time. This supports the approach of Shu and Yang mentioned in Section 7.2.1 (Shu and Yang 1991). They suggested a two-step fermentation with a temperature increase from 27°C to 32°C at the end of the exponential growth phase due to the fact that growth favors a lower temperature as the xanthan production. The methods reached starting temperatures in the range of 28.0°C to 30.3°C which are increased by 3.3°C to 5.8°C after 21 to 28 hours. The Hybrid1 method, for example, yields an initial temperature of 28.8°C which is shifted to 33.9°C at a time period of 21.4 hours. This time is also the end of the exponential growth due to the high glucose feeding at this time. This feeding totally suppresses growth and initiates the stationary growth phase and also the xanthan production phase.

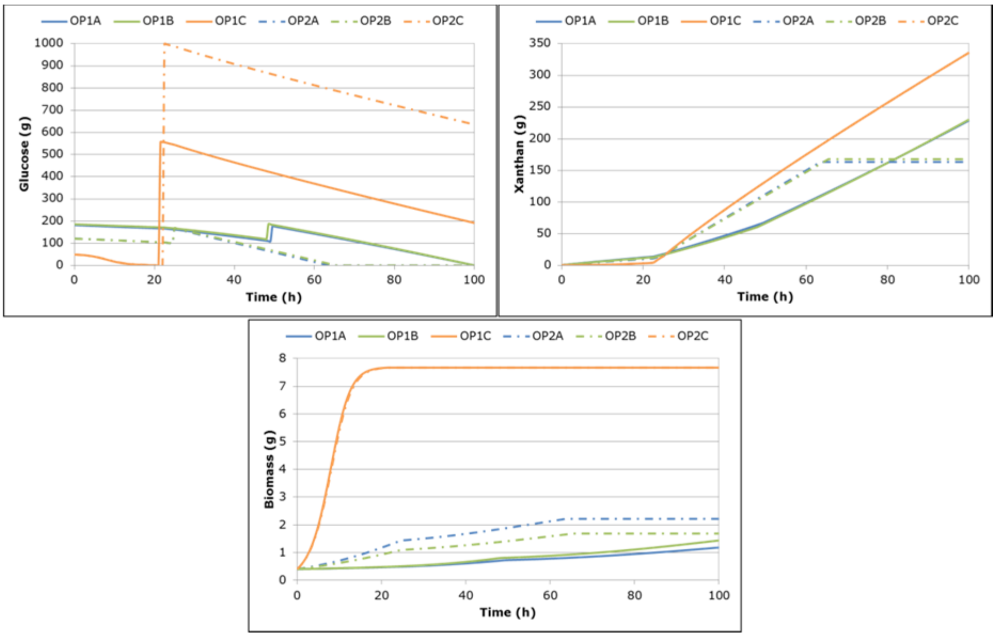


Figure 7.16: Simulation results of the optimized process parameters found by the hybrid 1 method for OP1 and OP2 with the bounds A, B, and C

Modification of the Feeding Number

The metabolically structured model in Figure 7.12 has been further modified to reach even higher (average) xanthan yields. Therefore, it is now possible that five times an amount of nitrogen and glucose is added to the fermenter. The modified model has 20 process parameters that have to be optimized to achieve maximum (average) xanthan yields. The process parameters are summarized in Table 7.5 together with their minimum and maximum values.

Table 7.5: Process parameters of the modified metabolically structured model in Figure 7.12

Factor	Description	Minimum Value	Maximum Value
$N0$	Initial nitrogen concentration	0.01	0.3
$S0$	Initial glucose concentration	10	50
$Sadd(i)1$	Nitrogen concentration of feeding at time $t_i = t_{i-1} + dft(i)$	0	1
$Sadd(i)2$	Glucose concentration of feeding at time $t_i = t_{i-1} + dft(i)$	0	70
$dft(i)$	Time from fermentation start to first feeding at $t_i = t_{i-1} + dft(i)$ $(t_0 = 0)$	0.1	100

$temp0$	Initial temperature	25	34
tt	Time of temperature increase	0	100
$tempHeight$	Increase of temperature	0	10

Thereby, the time between two subsequent feedings is optimized and not the feeding times to guarantee that $t_1 \leq t_2 \leq t_3 \leq t_4 \leq t_5$. The maximum fermentation time is set to $t_f = 100\ h$ so that if from a t_i on the feeding times are greater than 100 hours, these feedings are not considered. In this manner, it is realized that the optimization procedure can also find the best number of feedings. The same objective functions as for the original model defined in Eq. 7-29 and Eq. 7-30 are applied with

$$z = (N^0,\quad S^0,\quad Sadd_{11},\quad Sadd_{12},\quad Sadd_{21},\quad Sadd_{22},\quad Sadd_{31},\quad Sadd_{32},\quad Sadd_{41},\quad Sadd_{42},\quad Sadd_{51},\quad Sadd_{52},\quad dft_1,\quad dft_2,\quad dft_3,\quad dft_4,\quad dft_5,\quad temp_0,\quad tt,\quad tempHeight)$$

and the optimization problem corresponding to Eq. 7-29 is denoted by **MOP1** and the optimization problem corresponding to Eq. 7-30 is denoted by **MOP2**. The same parameter settings for the optimization methods are used as listed in Table A13 (Appendix A4-2). However, some has been adapted to take into consideration that the modified optimization problem has more than twice as many process parameters (see Table A14).

Results of MOP1 and MOP2

The best solutions for MOP1 and MOP2 are achieved by ES, Hybrid1, and Hybrid2; thereby, both hybrid algorithms have ES as a global optimizer. The solution found by DIRECT is better than those of CMAES, Hybrid3, and Hybrid4 but the (average) xanthan yield less compared to the best solution and, in addition, the required computing time is much more higher. In comparison to the best results of OP1C and OP2C, the PO yielded a 35 % higher xanthan yield and a 60 % higher average xanthan yield per hour.

A more precise view of the found solutions reveals that CMAES, Hybrid3, and Hybrid4 reach solutions by which not all five feedings are used. The feedings of MOP1 which are carried out after the maximum fermentation time of 100 hours have no effect on the fermentation. Similarly, the feedings of MOP2 which are carried out after the achieved fermentation time have no effect on the fermentation process.

It is also shown that the best working method, ES, achieves feeding times that are close together. Additionally, the nitrogen additions are close to zero and all glucose additions reach the upper limit of 70 g/l as in the original model. This leads to the suggestion that it is better to feed a higher glucose concentration once and not add nitrogen.

The best solution of MOP1 reaches similar curve shapes as the OP1A and OP1B and the best solution of MOP2 causes the two-step fermentation with a growth and a xanthan production phase similar to OP1C and OP2C. The worst solution of MOP2 found by CMAES achieves a solution by which the glucose concentration is exhausted at about 70 h and new glucose is not fed till time 91 and all other feeding times are higher than the maximum fermentation time. This solution is not acceptable because the xanthan production stops for that time period due to glucose exhaustion. Similarly, the Hybrid3 solution of MOP1 is not feasible because of the low glucose feeding and the corresponding low xanthan production rate.

Modification of the Bound for the Glucose Addition

Because the best solutions of MOP1 and MOP2 achieve glucose additions which reach the upper bound, this upper bound is increased from 70 g/l to 1000 g/l, denoted by **Bound C** and the optimization problems are denoted by **MOP1C** and **MOP2C**, to yield possibly even more xanthan. The results show that best solutions are achieved by ES, Hybrid1, and Hybrid2 for MOP1 higher xanthan yields than that of the other methods and by ES and Hybrid2 for MOP2C higher average xanthan yields per hour. Compared to the results before, the xanthan yield is increased by 32 % and the average xanthan yield per hour by 12 %. The simulation results of the solution found by ES again show the two-step fermentation due to the fact that the feeding times are so close together that the sum of the glucose additions could be added at the same point in time. The nitrogen additions are close to zero and can be neglected. Additionally, MOP1C and MOP2C terminate with nearly the same results so that the maximum xanthan yield is always observed after the maximum fermentation time. The glucose addition of MOP2C is chosen such that the glucose is totally exhausted after 100 hours and, hence, the xanthan production stops at this point in time. CMAES again achieves the worst solution for both problems due to too small and too late additions of glucose, respectively.

Modification of the Bounds for the Feeding Times and the Initial Nitrogen Concentration

To show that only one feeding is necessary, the lower bounds of the time between two feedings are set to zero. Additionally, the upper bound of the initial nitrogen concentration is increased to 10 g/l because the best solutions have always reached it. These expanded bounds are denoted by **Bound D** and the corresponding optimization problems by **MOP1D** and **MOP2D**. Due to the fact that ES always reached good process parameters in comparison to the other methods, the following PO studies are only performed using this method.

The results verify the assumption. The time between two feedings reached by both problems, MOP1D and MOP2D, is less than 22 seconds. The first feeding is supposed to take place at time 15.97 and 15.93 for MOP1D and MOP2D, respectively. Thereby, the nitrogen feeding concentration is always close to zero so that it can be assumed that no additional nitrogen has to be added to the fermenter.

The simulation results show clearly the division of the fermentation process in two phases (see Figure 7.17). The first phase is the growth phase with a high nitrogen concentration of 0.47 and 0.59 g/l and a low glucose concentration of 18.32 and 22.46 g/l for MOP1D and MOP2D, respectively. Higher nitrogen concentrations than 0.3 g/l totally suppress the xanthan production according to the model assumptions. This is reflected by nearly no xanthan production in the first fermentation phase also due to the low glucose concentration. The biomass increases in this phase up to an amount of 11.84 and 14.68 g, respectively.

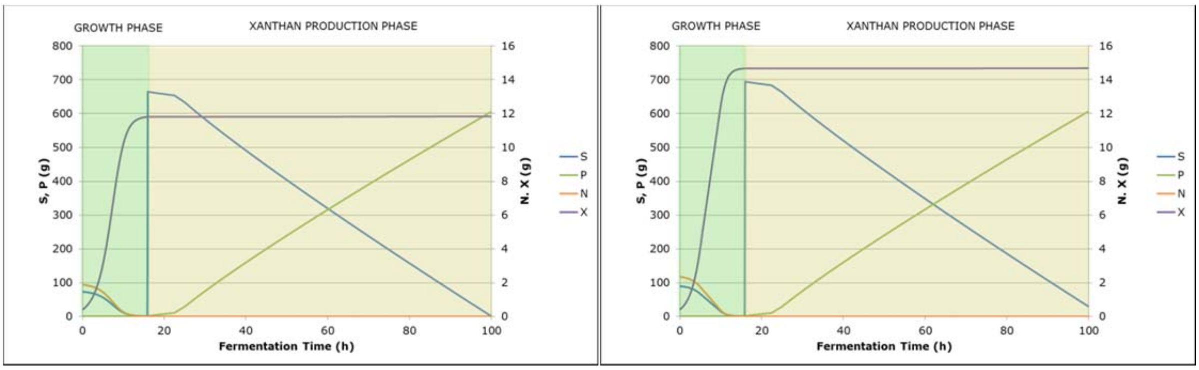


Figure 7.17: Simulation results of the modified metabolically structured model with process parameters found by optimization MOP1D (left) and MOP2D (right) with ES

The second phase is initiated by the first glucose addition. The others follow only seconds after so that all feedings can be combined to one which is added to the fermenter at time 15.97 and 15.93, respectively. At this time, the growth is totally suppressed by the high glucose concentration but no inhibition of xanthan by nitrogen can be observed because the concentration of nitrogen is nearly zero. This phase is the xanthan production phase with nearly no growth but high xanthan production up to an amount of 606 and 607 g, respectively.

The maximum xanthan yield in the case of MOP2D was also achieved at the maximum fermentation time of 100 hours; so there is no difference between optimizing MOP1D or MOP2D. In addition, the time of the temperature shift is nearly exact the same as that for the feeding. The temperature is then increased from 29.4°C and 29.1°C to 33.8°C and 33.9°C for MOP1D and MOP2D, respectively.

OXYGEN REGULATION

The slight slope of the xanthan curve at the beginning of the second phase is caused by the low oxygen content in the fermenter at this time (see Figure 7.18). At time 22.5 the oxygen content is increased due to an increase of the stirrer speed. This causes an increase in the slope of the xanthan curve. Hence, the stirrer speed has to be increased earlier to avoid an oxygen shortage in the fermenter.

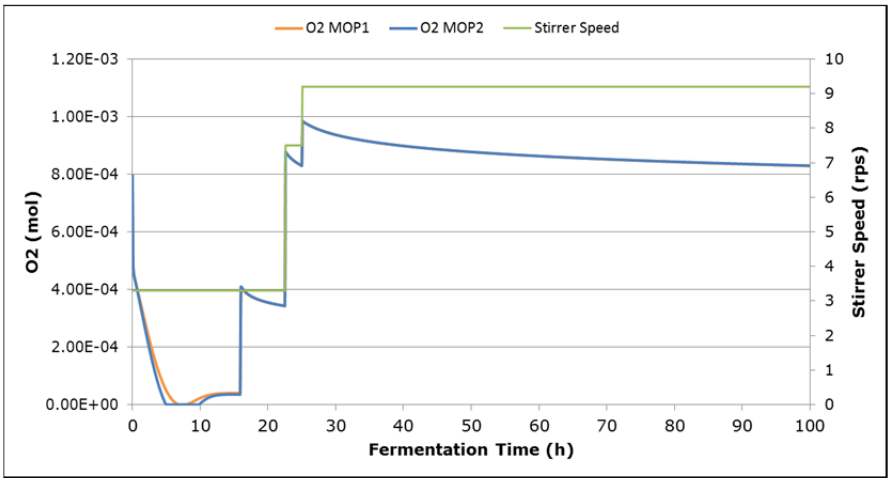


Figure 7.18: Simulation results of oxygen compared to stirrer speed (Bound D)

If the speed is increased at the beginning of the second phase, the xanthan yield can be further improved. It would be even better if the oxygen content was maintained at a constant level by a closed loop control of the stirrer speed, i.e. if the oxygen concentration in the fermenter falls below a determined value, the stirrer speed is increased until the oxygen remains above this limit.

Therefore, a new model component has been constructed with the aid of a discrete when-equation (see Section 3.1.2). If the oxygen concentration falls below 0.00015 mol/l, the stirrer speed is increased until the oxygen concentration remains above this limit up to the maximum speed of 20.3 rps.

The model is simulated with this new oxygen regulation by stirrer speed and the parameters found by MOP1D. It reaches a xanthan yield of 629.66 g in 52 hours. At this time the glucose is exhausted and the xanthan production stops. This shows that oxygen has a deep impact on xanthan production and should be controlled by a closed loop.

It can be assumed that this regulation with the proper process parameters can achieve even higher xanthan yields so that this regulated model is again optimized by ES. The results are displayed in Figure 7.19 and once more show the division in the two phases. Thereby, the simulation results achieved without oxygen regulation are plotted by dashed lines to highlight the high influence of oxygen regulation. A maximum xanthan yield of 2257.86 g is reached which is 3.5 times higher than the yield without regulation. The regulation has mainly influenced growth while the effect on xanthan production is minor. If the oxygen is not regulated, it is very low in the growth phase and causes a reduced biomass accumulation. The regulation insures that the oxygen concentration is maintained about 0.00015 g/l until the maximum stirrer speed is reached. In this manner the growth is improved, i.e. in less time more biomass is produced. The oxygen in the xanthan production phase differs slightly from that without regulation. However, more bacteria produce more xanthan which explains the high xanthan yield.

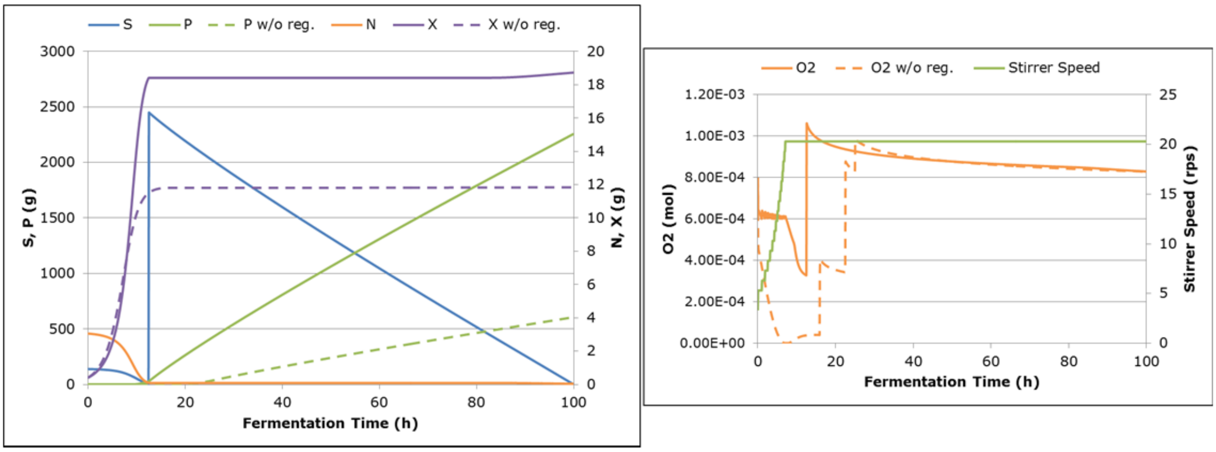


Figure 7.19: Simulation results with oxygen regulation by stirrer speed. Dashed lines indicate the results of MOP1D (see Figure 7.17)

7.7 Step 7: Deterministic and Stochastic Hybrid Simulation

In addition to the unstructured and the metabolically structured model, a chemically structured model has been constructed on the one hand with the aid of a stochastic Petri net (Figure 7.20) and on the other with the aid of a continuous Petri net (the stochastic transitions are replaced by continuous ones). Both models consist of a structured part for xanthan production from glucose which is modeled in the same manner as in the metabolically structured model in Figure 7.12 as well as a structured part for growth. In the case of the continuous model, the tokens directly reflect the concentrations of the biological compounds and in the case of the stochastic model, the tokens correspond to specific levels of concentrations, which has been already mentioned in Section 5.5.

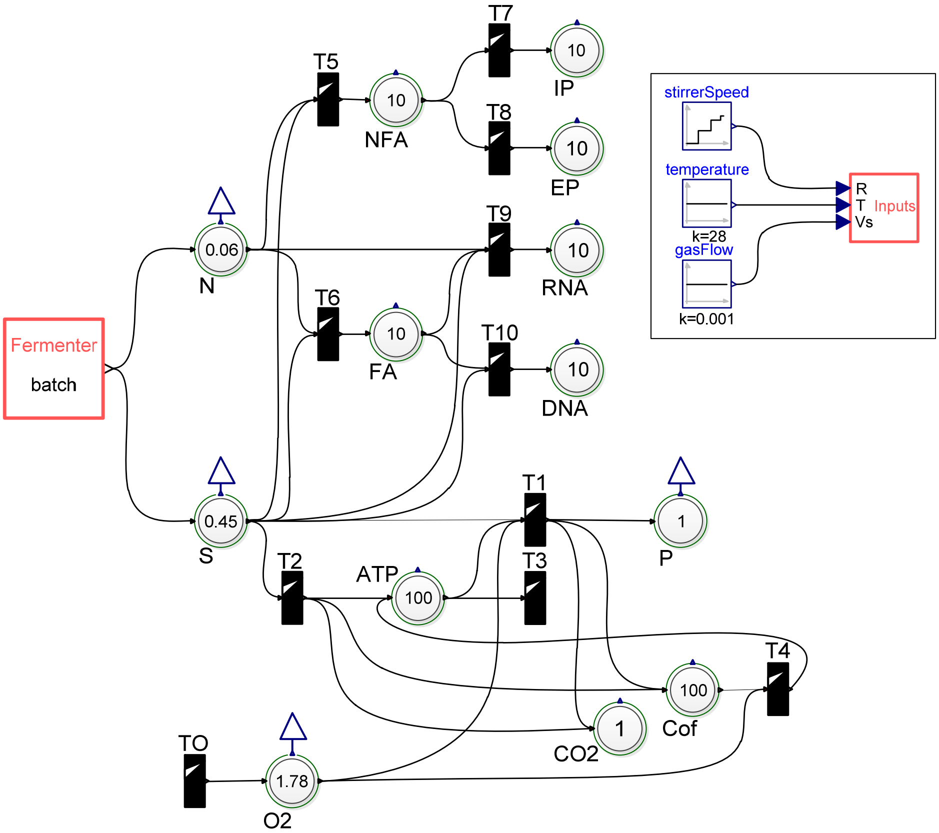


Figure 7.20: Chemically structured model of xanthan production and growth of *Xanthomonas campestris* bacteria

These concentration levels are achieved by defining a global maximum concentration M and a local highest level N_i different for each place so that $N_i + 1$ is the amount of levels. Then, the tokens $\mathcal{T}_i$ can be converted to level concentrations $\mathcal{L}_i$ by

$$\mathcal{L}_i = \mathcal{T}_i \cdot \frac{M}{N_i}. \qquad \text{Eq. 7-31}$$

The stochastic Petri net is constructed in Dymola by means of the PNlib and afterwards, the stochastic hybrid simulation (SHS) and further calculations are performed in Matlab/Simulink by the AMMod-tool. A place wrapper (green margin) can be found in the **Stochastic** sub-library of the **PNproBio** library to get the level concentrations instead of the token numbers as outputs in Matlab/Simulink. The deterministic hybrid simulation (DHS) of the continuous Petri net model can be performed directly in Dymola or with the AMMod-tool.

As demonstrated exemplarily in Section 5.5, there is a relationship between the rate of a reaction which corresponds to the maximum speed function of the continuous transition and the hazard function of the stochastic transition so that both can be easily transformed into each other. The SHS is performed by simulating the stochastic model 50 times and calculating the mean from the results for each compound at each point in time. The results are displayed in Figure 7.21 and show a good agreement between SHS of the stochastic Petri net model and DHS of the continuous Petri net model. Hence, the stochastic model can be approximated by the continuous model but the continuous model can only represent the averaged behavior.

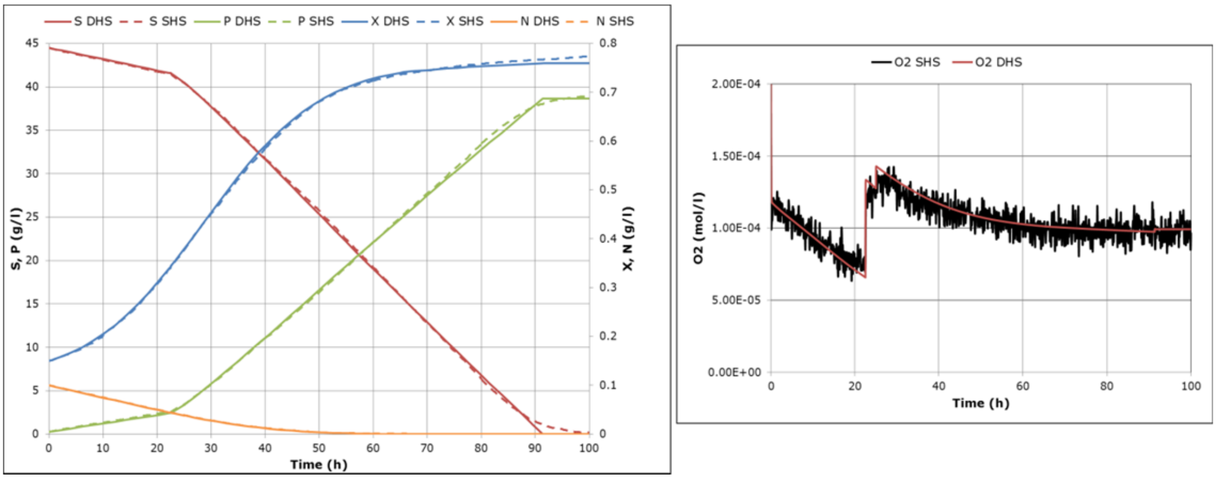


Figure 7.21: Comparison of SHS of the stochastic model and DHS of the corresponding continuous model (see Figure 7.20)

8 DISCUSSION AND OUTLOOK

An **environment** has been developed which comprises mathematical methods, concepts, and tools to enable the processing of experimental data to usable new insights about biological systems. Therefore, a **general, universally usable modeling process for biological systems** has been developed as well as a mathematical concept – **xHPN** (e**x**tended **H**ybrid **P**etri **N**ets) – which is properly adapted to the demands of biological processes. The definition of this new modeling formalism was necessary due to serious problems according to lacking unity, accuracy, and completeness of the current Petri net formalisms in literature.

The new xHPN concept has been transferred to the object-oriented modeling language **Modelica**. The developed Modelica library, called **PNlib** (**P**etri **N**et **lib**rary), in combination with an appropriate Modelica-tool enables graphical hierarchical modeling, hybrid simulation, and animation of xHPNs. Thereby, an additional Modelica library, called **PNproBio** (**P**etri **N**ets for **pro**cess modeling of **Bio**logical systems), provides wrapped xHPNs which offers on the one hand an easy-to-use-model at the top level with an intuitive and familiar adapted biological view and on the other the flexibility and generality of the xHPN concept. In addition to the Modelica libraries, the **AMMod** (**A**nalysis of **M**odelica **Mod**els) tool provides several mathematical methods for data preprocessing, relationship analysis, parameter estimation, sensitivity analysis, deterministic and stochastic hybrid simulation, and process optimization. The development of this new environment was necessary because no current tool gets along without drawbacks and interprets the hybrid Petri net formalism as needed for modeling biological systems in this work.

The application of the developed mathematical methods and concepts as well as the tools PNlib, PNproBio, and AMMod has been shown exemplarily by the xanthan production of *Xanthomonas campestris* bacteria. Additionally, it has given information about which method works best for which modeling step. Thereby, the focus was on showing the usage, the power, and the permit of the new environment rather than on yielding new insights about the studied organism. All investigations are based on pseudo experimental data and only one selected case, the xanthan production of *Xanthomonas campestris* bacteria, has been considered. The aim of further research is, hence, to verify the results by experimental data gained from wet lab fermentations of *Xanthomonas campestris* bacteria. Besides, the tools could be applied for other biological processes; the modeling process of the protein production of animal cells has just begun and will be enhanced in the future.

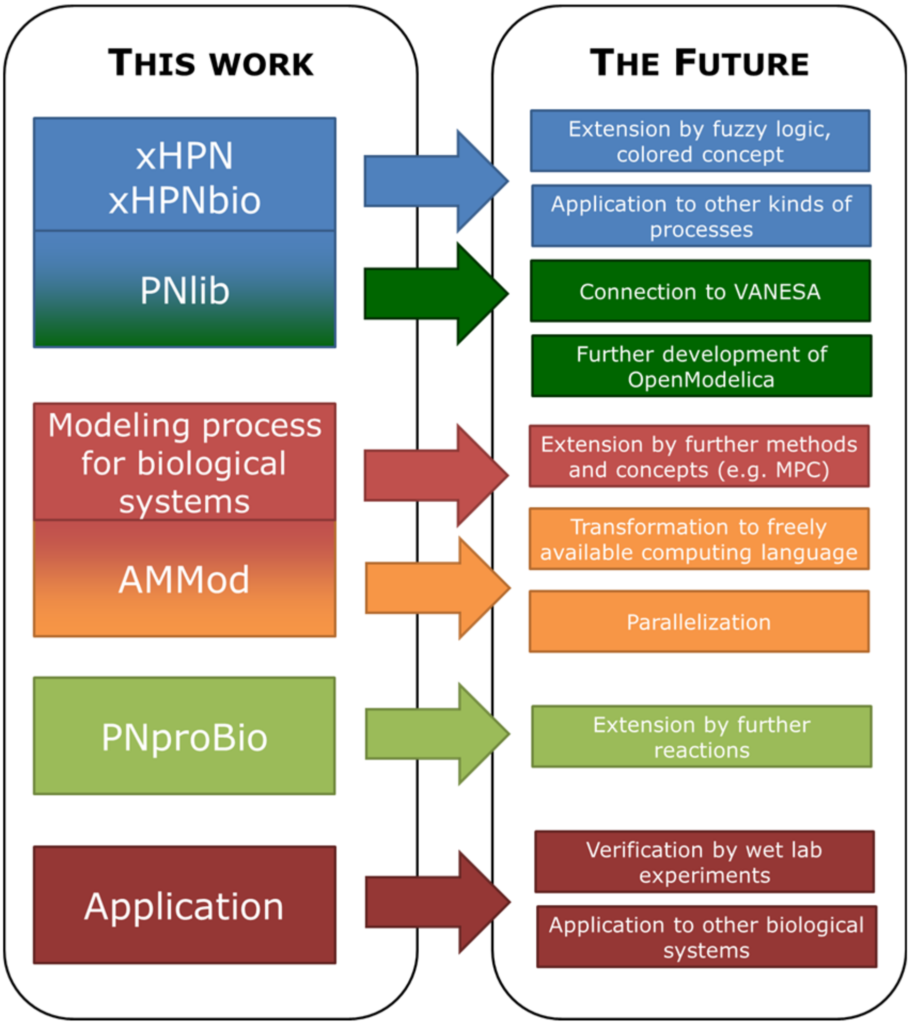

THIS WORK
THE FUTURE
xHPN
xHPNbio
PNlib
Modeling process for biological systems
AMMod
PNproBio
Application
Extension by fuzzy logic, colored concept
Application to other kinds of processes
Connection to VANESA
Further development of OpenModelica
Extension by further methods and concepts (e.g. MPC)
Transformation to freely available computing language
Parallelization
Extension by further reactions
Verification by wet lab experiments
Application to other biological systems

Figure 8.1: Further development possibilities beyond this work

During the work, it became increasingly clear that the mathematical modeling concept xHPN, especially developed to adress the demands of biological processes, is so powerful but also so universal and generic that it is an ideal **all-round-tool** for modeling and simulation of nearly all kinds of processes such as business processes, production processes, logistic processes, work flows, traffic flows, data flows, multi-processor systems, communication protocols, and functional principals. Hence, the abbreviation xHPN is chosen generally to emphasize that it is not restricted to biological applications. This developed formalism and the PNlib could be used for modeling and simulation of these mentioned processes and, in addition, the mathematical concept as well as the library could be extended by fuzzy logic (e.g. (Chen et al. 1990)) and the color concept (e.g. (Jensen 1987)) to further enhance the range of application fields. In the future, it is planned to establish a model-based process consulting for small and medium enterprises at the University of Applied Sciences Bielefeld based on the methods developed in this work.

In addition, a further aim is to provide all the tools as **open source**. This demands a further development of the open source Modelica-tool **OpenModelica** to get the PNlib to work with it because some Modelica features are not supported so far. The University of Applied Sciences Bielefeld is already closely involved in the further development of the OpenModelica-Tool (Braun 2010, Braun et al. 2010, Braun et al. 2011, Ochel et al. 2011, Krems et al. 2010).

Additionally, the mathematical methods required for the modeling process have to be transferred from the commercial language Matlab to an open source alternative like C++ or Python. Then, a new connection between OpenModelica and the open source tool for the analysis methods has to be established and a user-friendly GUI for the tools could be developed.

Moreover, the graphical modeling of Petri nets has to be improved because the input of arc weights is inconvenient due to the fact that arcs cannot have properties in Modelica. For this purpose the PNlib could be connected to **VANESA**, an open source tool for visualization and analysis of networks in systems biology applications developed at the faculty of technology at the Bielefeld University (Janowski 2008). The graphical modeling of Petri nets is performed in VANESA with an easy-to-use interface and, thereafter, the constructed Petri net is translated to the Modelica language and compiled and simulated with a Modelica-tool. The simulation results are then sent back to VANESA for display and animation of the token development. Beyond that, VANESA offers the possibility to load networks from several databases and convert them to nets. These nets can then be transformed to Petri nets. Furthermore, VANESA contains various qualitative methods from graph theory which are also usable on Petri nets and, in addition, some Petri net specific analysis methods have been already integrated (Brinkrolf 2011). The connection of PNlib and VANESA has already commenced (Proß et al. 2012a, Proß et al. 2012b).

Moreover, the functionality of AMMod could also be further expanded by methods for parameter estimation, sensitivity analysis, and process optimization. In addition, methods for new analysis technics like the design of experiments, model based experimental design, model reduction, and uncertainty analysis could be developed and implemented. Especially the process optimization procedure offers a good basis for further development. Till now, it achieves an open-loop control strategy, i.e. the strategy is given based on the model and no feedback from the real system is included. This feedback could be also considered by realizing a **closed-loop control**, particularly the so-called **model predictive control (MPC)**, to further improve the biological process. Therefore, the model is used to calculate the process states of the real system in the future depending on current measurements. Based on these predictions and an expert system in the background, the process parameters can be adapted appropriately. With additional methods from artificial intelligence, a **self-learning system** could be established. The vision is that this MPC of fermentation processes proceeds fully automatically as it is depicted in Figure 8.2. Thereby, the process data is measured by specific sensors and the process parameters are properly adapted by actuators depending on the control data gained from mathematical methods and the process model.

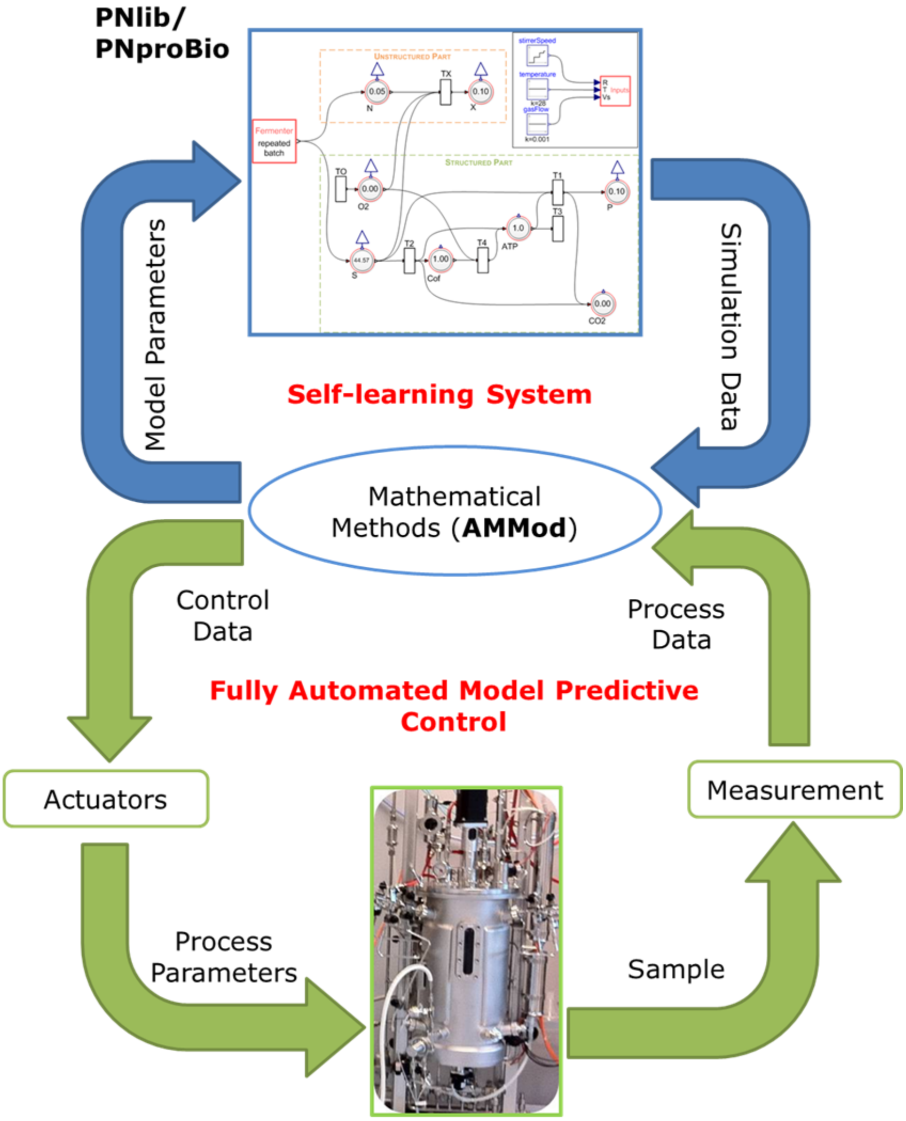


Figure 8.2: The vision: A fully automated model predictive control of fermentation processes

Furthermore, the computing time of mostly all mathematical methods used in the modeling process has to be reduced to make them applicable on large models. A possibility to do this is **parallelization**. Thereby, the algorithm is divided into separate parts which can be executed in parallel by a simultaneous computer or a computer cluster. Especially ES and CMAES can be easily parallelized by simultaneous evaluation of the individuals of one generation. In the same manner, the simulations required for local and global sensitivity analysis could be performed in parallel.

In conclusion, this work presents a new powerful and universally applicable modeling environment that can be used for nearly all kinds of biological processes. In addition, the developed methods and tools are also applicable for many kinds of processes from other application fields, e.g. business processes, production processes, and logistic processes. The xHPN formalism and the corresponding processes have been defined precisely and are so powerful that most formalisms are included regardless of the modeling approach (qualitative vs. quantitative, continuous vs. discrete, deterministic vs. stochastic). Furthermore, the wrapping technique offers on the one hand the possibility to provide biologists an intuitive and familiar adapted biological view of the model and on the other the flexibility and generality of the xHPN formalism for the hybrid simulation. Moreover, the developed mathematical methods provide a means to estimate the model parameters from the given experimental data when direct experimental determination is not possible. In addition, these methods enable the optimization of underlying processes in order to achieve an open-loop control.

APPENDIX

A1 ALGORITHMS

Algorithm A1 (Hooke-Jeeves method)

Input: start value x^0, objective function $Q(x)$, step size control parameter $\rho \in (0,1)$, minimum step size $\varepsilon > 0$, abort criteria $\mathcal{A}$

Start: Set $\delta = \rho \cdot x^0$, $x = x^0$, $Q_x = Q(x^0)$ and $\Delta = \rho$

Iteration:

1. Exploratory move: Evaluate $(z, Q_z) = exploratoryMove(x, Q_x, \delta)$
2. Pattern search: If $Q_z < Q_x$, set $\theta = x$, $x = z$, $z = 2 \cdot z - \theta$, $Q_x = Q_z$, calculate $Q_z = Q(z)$ and evaluate $(z, Q_z) = exploratoryMove(z, Q_z, \delta)$
   If $Q_z > Q_x$, go to step 1; otherwise, for $l = 1,2, \dots, n$ if $abs(z_i - x_i) > 0.5 \cdot abs(\delta_i)$, go to step 2.
3. Reduce step size $(Q_z > Q_x)$: If $\Delta \geq \varepsilon$, set $\Delta = \Delta \cdot \rho$, $\delta = \delta \cdot \rho$ and go to step 1; otherwise, stop.

Stop: If at least one of the abort criteria $a \in \mathcal{A}$ is fulfilled or the iteration stops.

Procedure *exploratoryMove*:

```
Input: $x, Q_x, \delta$
Output:$y, Q_y$
set $y = x$ and $Q_y = Q_x$
**for** $i = 1, 2, \dots, n$ **do**
    Set $m = x + \delta_i e_i$, where $e_i$ is the $i$th unit vector, and evaluate $Q_m = Q(m)$.
    If $Q_m < Q_x$, set $y = m$ and $Q_y = Q_m$.
    Otherwise, set $m = x - \delta_i e_i$, where $e_i$ is the $i$th unit vector and evaluate $Q_m = Q(m)$.
    If $Q_m < Q_x$, set $y = m$ and $Q_y = Q_m$.
**end**
```

Algorithm A2 (Nelder-Mead simplex method)

Input: start value x^0, objective function $\mathrm{Q}(x)$, reflection parameter $\rho = 1$, expansion parameter $\chi = 2$, contraction parameter $\gamma = 1/2$, shrinkage parameter $\sigma = 1/2$, abort criteria $\mathcal{A}$

Start: Generate an initial simplex around x^0 by adding 5% of each component $x_l^0, l = 1,2,\dots,n$ to x^0. These n vectors and x^0 are then the $n + 1$ vertices of the initial simplex. (It uses 0.00025 as component l if $x_l^0 = 0.$)

Iteration:

1. Order: Order the $n + 1$ vertices from the lowest objective function value $Q_1 = Q(x_1)$ to the highest one $Q_{n+1} = Q(x_{n+1})$ such that
$$Q_1 \le Q_2 \le \cdots \le Q_{n+1}$$
2. Reflect: Compute the reflection point r
$$r = (1 + \rho)m - \rho x_{n+1} = 2m - x_{n+1},$$
where
$$m = \frac{1}{n}\sum_{l=1}^{n} x_i$$
and evaluate $Q_r = Q(r)$.
If $Q_1 \le Q_r < Q_n$, accept r and terminate the iteration.
3. Expand: If $Q_r < Q_1$, calculate expansion point s
$$s = m + \rho\chi(m - x_{n+1}) = m + 2(m - x_{n+1})$$
and evaluate $Q(s) = Q_s$.
   a. If $Q_s < Q_r$, accept s and terminate the iteration.
   b. Otherwise, accept r and terminate the iteration.
4. Contract: If $Q_r \ge Q_n$, perform a contraction between m and the better of x_{n+1} and r.
   a. Outside: If $Q_r < Q_{n+1}$, calculate
   $$c = m + \gamma(r - m) = m + \frac{r - m}{2}$$
   and evaluate $Q_c = Q(c)$. If $Q_c < Q_r$, accept c and terminate the iteration; otherwise, go to step 5 (perform a shrink).
   b. Inside: If $Q_r \ge Q_{n+1}$, calculate
   $$cc = m + \gamma(x_{n+1} - m) = m + \frac{x_{n+1} - m}{2}$$
   and evaluate $Q_{cc} = Q(cc)$. If $Q_{cc} < Q_{n+1}$, accept cc and terminate the iteration; otherwise, go to step 5 (perform a shrink).
5. Perform a shrink step: Calculate the n points
$$v_l = x_1 + \sigma(x_l - x_1) = x_1 + \frac{x_l - x_1}{2}, \qquad l = 2, 3, \dots, n + 1$$

and evaluate $Q(v_l), l = 2, 3, \ldots, n+1$. The simplex at the next iteration is $x_1, v_2, \ldots, v_{n+1}$.

Stop: If at least one of the abort criteria $a \in \mathcal{A}$ is fulfilled.

Algorithm A3 (DIRECT method)

Input: objective function $Q(x)$, abort criteria $\mathcal{A}$, global/local search weight parameter $\varepsilon > 0$

Start: Normalize the search space to be the unit hypercube and evaluate the function value $Q(c_1)$ at its midpoint c_1. Set $Q_{min} = Q(c_1)$.

Iteration:

1. Identify the set S of potentially optimal rectangles.
2. Select any rectangle $j \in S$.
3. Perform the procedure for dividing rectangles. Update Q_{min}.
4. Set $S = S \setminus \{j\}$. If $S \neq \emptyset$ go to step 3.
5. Go to step 1.

Stop: If at least one of the abort criteria $a \in \mathcal{A}$ is fulfilled.

Procedure for dividing rectangles:

1. Identify the set I of dimensions with the maximum side length. Let δ be equal to one-third of this maximum side length.
2. Evaluate the objective function at the points $c \pm \delta e_i \; \forall i \in I$, where c is the midpoint of the rectangle and e_i is the ith unit vector.
3. Divide the rectangles into thirds along the dimensions in I, starting with the lowest value of $w_i = \min\big(Q(c + \delta e_i),\; Q(c - \delta e_i)\big)$ and continuing to the dimension with the highst w_i.

Algorithm A4 (Evolution strategy)

```
Set $g = 0$
Initialize $A(0) = \{a_1(0), a_2(0), \ldots, a_\mu(0)\} \in I^\mu$
        where $a_k = (x_k, s_k)$
        with $s_k = (\sigma_i, m_j,\ \forall i = 1, 2, \ldots, n,\ \forall j = 1, 2, \ldots, n(n-1)/2)$ or $s_k = (\sigma_i,\ \forall i = 1, 2, \ldots, n)$
Evaluate $A(0)$: $\{\Phi(a_1(0)), \Phi(a_2(0)), \ldots, \Phi(a_\mu(0))\}$
        where $\Phi(a_k(0)) = Q(x_k(0))$
while not $\iota(A(g))$ do
    Recombine strategy parameters: $s'_k(g) = r'_s(A(g)),\ k = 1, 2, \ldots, \lambda$
    Recombine objective parameters: $x'_k(g) = r'_p(A(g)),\ k = 1, 2, \ldots, \lambda$
    Mutate strategy parameters: $s''_k(g) = m'_s(s'_k(g)),\ k = 1, 2, \ldots, \lambda$
    Mutate objective parameters: $x''_k(g) = m'_p(x'_k(g), s''_k(g)),\ k = 1, 2, \dots, \lambda$
    Evaluate $P''(g) = (a''_k(g),\ k = 1, 2, \dots, \lambda)$, where $a''_k(g) = (x''_k(g), s''_k(g))$:
    $\{\Phi(a''_1(g)), \Phi(a''_2(g)), \dots, \Phi(a''_\lambda(g))\}$, where $\Phi(a''_k(g)) = Q(x''_k(g))$
    Select: $A(g+1) =$ **if** $(\mu, \lambda)$-selection **then** $s_{(\mu,\lambda)}(A''(g))$ **else** $s_{(\mu+\lambda)}(A(g) \cup A''(g))$
    Set $g = g + 1$
end while
```

Algorithm A5 (Covariance matrix adaption evolution strategy)

Input: objective function $Q(x)$, abort criteria $\mathcal{A}$, start distribution mean $m^0 \in X \subseteq \mathbb{R}^n$, and start step size $\sigma^0 \in \mathbb{R}_+$

Start: Set parameters λ, μ, w_k $(k = 1, 2, \dots, \mu)$, c_σ, c_1, and c_μ to their default values according to Table A1.

Set $m = m^0$, $\sigma = \sigma^0$, $q_\sigma = 0$, $q_c = 0$, $C = I$, and $g = 0$.

Iteration:

1. Mutation to generate new offspring:
   Perform an eigendecomposition of the covariance matrix $C = BD^2B$, where the columns of B are an orthonormal basis of eigenvectors and the diagonal elements of the diagonal matrix D are the square roots of the corresponding eigenvalues.
   Evaluate $x_k = m + z_k$, where $z_k = \sigma BD\mathcal{N}(0, I)$.
2. Selection of the μ best individuals such as $Q(x_{1:\lambda}) \leq Q(x_{2:\lambda}) \leq \dots \leq Q(x_{\mu:\lambda})$.
3. Recombination of the μ best individuals
   Set $m_{old} = m$ and $m = \sum_{k=1}^{\mu} w_k x_{k:\lambda}$.
4. Step size control:
   Calculate evolution path $q_\sigma = (1 - c_\sigma)q_\sigma + \sqrt{c_\sigma(2 - c_\sigma)\mu_{eff}} \cdot C^{-\frac{1}{2}} \cdot \left(\frac{m - m_{old}}{\sigma}\right)$.
   Set $\sigma_{old} = \sigma$ and $\sigma = \sigma_{old} \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|q_\sigma\|}{E(\|N(0,I)\|)} - 1\right)\right)$,
   where $E(\|N(0,I)\|) = \sqrt{n}\left(1 - \frac{1}{4n} + \frac{1}{21n^2}\right)$.
5. Covariance matrix adaptation:
   Calculate evolution path $q_c = (1 - c_c)q_c + h_\sigma\sqrt{c_c(2 - c_c)\mu_{eff}} \cdot \left(\frac{m - m_{old}}{\sigma_{old}}\right)$.
   Set
   $$C = (1 - c_1 - c_\mu)C + c_1(q_c q_c^t + \delta(h_\sigma)C) + c_\mu \sum_{k=1}^{\mu} w_k \left(\frac{x_{k:\lambda} - m_{old}}{\sigma_{old}}\right)\left(\frac{x_{k:\lambda} - m_{old}}{\sigma_{old}}\right)^t,$$
   where
   $$h_\sigma = \begin{cases} 1 & \text{if} \frac{\|q_\sigma\|}{\sqrt{1 - (1 - c_\sigma)^{2(g+1)}}} < (1.4 + \frac{2}{n+1} E(\|N(0,I)\|) \\ 0 & \text{otherwise} \end{cases}$$

$$\delta(h_\sigma) = (1 - h_\sigma)c_c(2 - c_c)$$

Stop: If at least one of the abort criteria $a \in \mathcal{A}$ is fulfilled.

The default values for the exogenous strategy parameters of the algorithm are given in the following table.

Table A1: Default exogenous strategy parameters (Hansen 2009)

Parameter	Value
λ	$4 + \lfloor 3\, ln(n) \rfloor$
μ	$\lfloor \lambda/2 \rfloor$
w_k	$\frac{ln(\lambda/2 + 0.5) - ln(k)}{\sum_{j=1}^{\mu}(ln(\lambda/2 + 0.5) - ln(j))}$ $k = 1, 2, \dots, \mu$
c_σ	$\frac{\mu_{eff} + 2}{n + \mu_{eff} + 5}$
d_σ	$1 + 2\, max\left(0, \sqrt{\frac{\mu_{eff} - 1}{n + 1}} - 1\right) + c_\sigma$
c_c	$\frac{4 + \mu_{eff}/n}{n + 4 + 2\mu_{eff}/n}$
c_1	$\frac{2}{(n + 1.3)^2 + \mu_{eff}}$
c_μ	$min\left(1 - c_1, \frac{2\mu_{eff} - 4 + \frac{2}{\mu_{eff}}}{(n + 2)^2 + \mu_{eff}}\right)$

Algorithm A6 (Hybrid algorithm)

Input: objective function $Q(x)$, abort criteria $\mathcal{A}$, method specific parameters

Start: Set $g = 0$ and initialize $A(0) = \{a_1(0), a_2(0), \dots, a_\mu(0)\} \in I^\mu$ where $a_k = (x_k, s_k)$

with $s_k = (\sigma_i, m_j,\ \forall i = 1, 2, \dots, n,\ \forall j = 1, 2, \dots, n(n-1)/2)$

or $s_k = (\sigma_i,\ \forall i = 1, 2, \dots, n,)$.

Evaluate $A(0)$: $\{\Phi(a_1(0)), \Phi(a_2(0)), \dots, \Phi(a_\mu(0))\}$ where $\Phi(a_k(0)) = Q(x_k(0))$

and $x_{min}(0)$ with $Q(x_{min}(0)) \le Q(x_k(0))\ \forall k = 1, 2, \dots, \mu$.

Diversification:

```
    while not $S1$ or $S2$ or $S3$ do
        Recombine strategy parameters: $s'_k(g) = r'_s(A(g)),\ k = 1, 2, \dots, \lambda$
        Recombine objective parameters: $x'_k(g) = r'_p(A(g)),\ k = 1, 2, \dots, \lambda$
        Mutate strategy parameters: $s''_k(g) = m'_s(s'_k(g)),\ k = 1, 2, \dots, \lambda$
        Mutate objective parameters: $x''_k(g) = m'_p(x'_k(g), s''_k(g)),\ k = 1, 2, \dots, \lambda$
        Evaluate $A''(t) = (a''_k(g),\ k = 1, 2, \dots, \lambda)$, where $a''_k = (x''_k(g), s''_k(g))$:
        $\{\Phi(a''_1(g)), \Phi(a''_2(g)), \dots, \Phi(a''_\lambda(g))\}$, where $\Phi(a''_k(g)) = Q(x''_k(g))$
        Select: $A(g+1) =$ if $(\mu, \lambda)$-selection then $s_{(\mu,\lambda)}(A''(g))$ else $s_{(\mu+\lambda)}(A(g) \cup A''(g))$
        Evaluate $x_{min}(g+1)$ with $Q(x_{min}(g+1)) \le Q(x_k(g+1))\ \forall k = 1, 2, \dots, \mu$.
        Set $g = g + 1$
    end while
Intensification:
    Perform A1 Algorithms
Algorithm A1 or Algorithm A2 with the start value $x^0 = x_{min}(g+1)$.
```

In the same manner CMAES of Algorithm A5 can be used in the diversification phase.

Algorithm A7 (Gillespie's direct method)

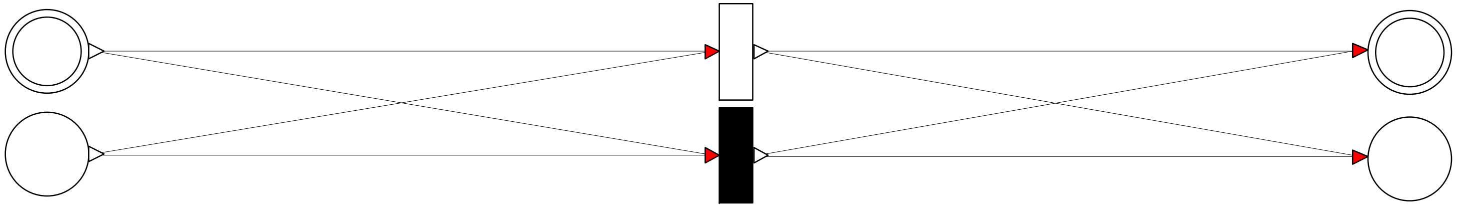
1. Set the initial numbers of molecules and set $t = 0$.
2. Calculate the hazard function h_j for all j.
3. Choose the reaction R_k randomly with the probabilities $h_k / \sum_j h_j$
4. Generate the time $\tau \sim Exp(\sum_j h_j)$ when reaction R_k occurs.
5. Update the number of molecules affected by reaction R_k.
6. If $t = T_{max}$, stop; otherwise, set $t = t + \tau$ and go to step 2.

Algorithm A8 (First reaction method)

1. Set the initial numbers of molecules and set $t = 0$.
2. Calculate the hazard function h_j for all j.
3. Generate a putative time $\tau_j \sim Exp(h_j)$ for all j.
4. Let R_k the reaction whose putative time τ_k is the smallest.
5. Update the number of molecules affected by reaction R_k.
6. If $t = T_{max}$, stop; otherwise, set $t = t + \tau_k$ and go to step 2.

A2 Connector Variables of the PNlib

Table A2: Variables of the PNlib connectors. Orange variables are only part of discrete communications while blue variables are only part of continuous communications



PlaceOut		TransitionIn		TransitionOut		PlaceIn	
INPUT		INPUT		INPUT		INPUT	
Bool	**active**	**Real**	**t**	**Real**	**t**	**Bool**	**active**
Are the output transitions active?		Markings of input places		Markings of output places		Are the input transitions active?	
Bool	**fire**	**Integer**	**tint**	**Integer**	**tint**	**Bool**	**fire**
Do the output transitions fire?		Integer markings of input places		Integer markings of output places		Do the input transitions fire?	
Real	**arcWeight**	**Real**	**minTokens**	**Real**	**maxTokens**	**Real**	**arcWeight**
Weights of output arcs		Minimum capacites of input places		Maximum capacities of output places		Weights of input arcs	
Integer	**arcWeightint**	**Integer**	**minTokensint**	**Integer**	**maxTokensint**	**Integer**	**arcWeightInt**
Integer weights of output arcs		Integer minimum capacites of input places		Integer maximum capacites of input places		Integer weights of input arcs	
Bool	**disTransition**	**Bool**	**enable**	**Bool**	**enable**	**Bool**	**disTransition**
Types of output transitions (discrete/stochastic or continuous)		Is the transition enabled by input places?		Is the transition enabled by output places?		Types of input transitions (discrete/stochastic or continuous)	
Real	**instSpeed**	**Bool**	**tokenInOut**	**Bool**	**disPlace**	**Bool**	**enabledByInPlaces**
Instantaneous speeds of continuous output transitions		Do the input places have a discrete token change?		Types of output places (discrete or continuous)		Are the input transitions enabled by all their input places?	
Real	**prelimSpeed**	**Integer**	**arcType**	**Bool**	**emptied**	**Real**	**instSpeed**
Preliminary speeds of continuous output transitions		Types of input arcs (normal, test, inhibition, or read)		Are the continuous output places emptied?		Instantaneous speeds of continuous input transitions	
Real	**maxSpeed**	**Bool**	**disPlace**	**Real**	**decreasingFactor**	**Real**	**prelimSpeed**

Maximum speeds of continuous output transitions		Types of input places (discrete or continuous)		Factors of continuous output places for decreasing the speed		Preliminary speeds of continuous input transitions	
		Real	**testValue**	**Real**	**speedSum**	**Real**	**maxSpeed**
		Test values of test or inhibitor arcs		Output speeds of continuous output places		Maximum speeds of continuous input transitions	
		Integer	**testValueint**				
		Integer test values of test or inhibitor arcs					
		Integer	**normalArc**				
		Double arc: test and normal arc or inhibitor and normal arc					
		Bool	**fed**				
		Are the continuous input places fed?					
		Real	**decreasingFactor**				
		Factors of continuous input places for decreasing the speed					
		Real	**speedSum**				
		Input speeds of continuous input places					
OUTPUT		**OUTPUT**		**OUTPUT**		**OUTPUT**	
Real	**t**	**Bool**	**active**	**Bool**	**active**	**Real**	**t**
Marking of the place		Is the transition active?		Is the transition active?		Marking of the place	
Integer	**tint**	**Bool**	**fire**	**Bool**	**fire**	**Integer**	**tint**
Integer marking of the place		Does the transition fire?		Does the transition fire?		Integer marking of the place	
Real	**minTokens**	**Real**	**arcWeight**	**Real**	**arcWeight**	**Real**	**maxTokens**
Minimum capacity of the place		Weights of input arcs		Weights of output arcs		Maximum capacity of the place	
Integer	**minTokensint**	**Integer**	**arcWeightint**	**Integer**	**arcWeightint**	**Integer**	**maxTokensint**
Integer minimum capacity of the place		Integer weights of input arcs		Integer weights of output arcs		Integer maximum capacity of the place	
Bool	**enable**	**Bool**	**disTransition**	**Bool**	**disTransition**	**Bool**	**enable**
Which of the output transitions are enabled by the place?		Type of the transition (discrete/stochastic or continuous)		Type of the transition (discrete/stochastic or continuous)		Which of the input transitions are enabled by the place?	
Bool	**tokenInOut**	**Real**	**instSpeed**	**Bool**	**enabledByInPlaces**	**Bool**	**disPlace**
Does the place have a discrete token change?		Instantaneous speed of a continuous transition		Is the transition enabled by all input places?		Type of the place (discrete or continuous)	
Integer	**arcType**	**Real**	**prelimSpeed**	**Real**	**instSpeed**	**Bool**	**emptied**
Types of output arcs (normal, test, inhibition, or read)		Preliminary speed of a continuous transition		Instantaneous speed of a continuous transition		Is the continuous place emptied by output transitions?	
Bool	**disPlace**	**Real**	**maxSpeed**	**Real**	**prelimSpeed**	**Real**	**decreasingFactor**

Type of the place (discrete or continuous)		Maximum speed of a continuous transition		Preliminary speed of a continuous transition		Factor for decreasing the speed of continuous input transitions	
Real	**testValue**			**Real**	**maxSpeed**	**Real**	**speedSum**
Test values of test or inhibitor arcs				Maximum speed of a continuous transition		Output speed of a continuous place	
Integer	**testValueint**						
Integer test values of test or inhibitor arcs							
Integer	**normalArc**						
Double arc: test and normal arc or inhibitor and normal arc							
Bool	**fed**						
Is the continuous place fed by input transitions?							
Real	**decreasingFactor**						
Factor for decreasing the speed of continuous output transitions							
Real	**speedSum**						
Input speed of a continuous place							

A3 SUPPLEMENTS TO THE AMMOD-TOOL

A3-1 PRA

Curve Fitting Toolbox^{TM} of Matlab (MathWorks 2011) has the ability to smooth the experimental data as described in Section 5.1. This can be done by the prompt

```
[cfun,gof,output] = fit(t,ymsd,'smoothingspline')
```

called from within a Matlab procedure, whereby the vector t comprises the time points and $ymsd$ the corresponding measurements. The approximated data and the corresponding derivative can then be obtained by the prompts

```
   y = feval(cfun,t);
dydt = differentiate(cfun,t);
```

Another possibility is to perform the fitting through a GUI opened by the prompt `cftool.` After preprocessing the data with smoothing splines, functional relationships between the considered biological compounds and their reaction rates can be analyzed. Thereby, the smoothed reaction rates are derived from the first derivations of the smoothing splines. The RA can also be performed with Curve Fitting Toolbox^{TM} in Matlab by comparing two data vectors obtained from smoothing with each other. Thereby, the entries of these vectors have to refer to the same points in time. One vector is chosen to be the X Data and the other is assigned to the Y Data. Afterwards, the relationship between these two data vectors is analyzed by creating a fit. Several equation types are available to find a function which describes the relationship; custom equations can be defined, too.

A3-2 PESA

To use the PESA-option on the main menu of the AMMod-tool, the experimental data have to be prepared in an Excel map. One sheet of this Excel map has to contain the initial conditions of the experiments; the structure is shown in Table A3. The example comprises data of five fermentation experiments with changing initial nitrogen ($N0$) and glucose ($S0$) concentrations. The first column *sheet* determines the Excel sheet with the experimental data which correspond to the initial conditions $N0$ and $S0$ and the second column *weight* defines the sheet with the weights of each data point. If the word “no” is entered in the weight column, all data points are weighted equally. The columns after these two columns comprise the initial conditions.

Table A3: Excel sheet with the initial conditions of experiments

sheet	weight	N0	S0
data1	W1	0.13	35
data2	W2	0.25	30
data3	W3	0.28	45
data4	W4	0.28	20
data5	W5	0.12	30

The sheets with the experimental data have to be structured as shown in Table A4. The first column comprises the points in time when the studied substances have been measured. The following columns contain the measured values of these substances: glucose (S), oxygen $(O2)$, xanthan (P), nitrogen (N), and biomass (X).

Table A4: Excel sheet with an experimental data set

time	S	O2	P	N	X
0.00	35.00	2.00E-04	0.10	0.13	0.10
5.50	34.45	1.87E-04	0.28	0.12	0.13
15.00	33.06	1.77E-04	0.75	0.11	0.22
30.00	29.10	1.91E-04	2.11	0.07	0.44
40.00	25.02	1.88E-04	3.61	0.04	0.64
50.00	20.33	1.86E-04	5.53	0.02	0.79
60.00	15.65	1.85E-04	7.67	0.00	0.86
70.00	11.10	1.84E-04	9.88	0.00	0.88
80.00	6.61	1.84E-04	12.10	0.00	0.89
90.00	2.13	1.84E-04	14.32	0.00	0.89
100.00	0.00	1.83E-04	15.38	0.00	0.89

The weights are entered in a separate sheet in the form of a matrix shown in Table A5. In this example, the weights are the reciprocal of the quadratic maximum amount of each substance.

Table A5: Excel sheet with a weight matrix which comprises a weight for each data point

8.16E-04	2.50E+07	4.23E-03	59.17	1.26
8.16E-04	2.50E+07	4.23E-03	59.17	1.26
8.16E-04	2.50E+07	4.23E-03	59.17	1.26
8.16E-04	2.50E+07	4.23E-03	59.17	1.26
8.16E-04	2.50E+07	4.23E-03	59.17	1.26
8.16E-04	2.50E+07	4.23E-03	59.17	1.26
8.16E-04	2.50E+07	4.23E-03	59.17	1.26

8.16E-04	2.50E+07	4.23E-03	59.17	1.26
8.16E-04	2.50E+07	4.23E-03	59.17	1.26
8.16E-04	2.50E+07	4.23E-03	59.17	1.26
8.16E-04	2.50E+07	4.23E-03	59.17	1.26

It should be noted that all mentioned sheets have to be provided within one Excel map.

Settings that have to be made on the PESA GUI:

- **Simulink Model**: Select the Simulink model which corresponds to the Modelica model whose parameters should be estimated or for which a sensitivity analysis should be performed.
- **Dymola Block**: Name of the Dymola block in the Simulink model.
- **Input File**: Select the text file that contains the inputs of the Modelica model.
- **Data File**: Select the Excel file with the experimental data structured as mentioned above.
- **Initial Sheet**: Select the name of the Excel sheet with the initial conditions.
- **Method**: Select an optimization algorithm of those introduced in Section 3.2:
    - Hooke-Jeeves
    - Nelder-Mead
    - DIRECT
    - ES
    - CMAES
    - Hybrid

  or a sensitivity analysis method from those introduced in Section 3.3:
    - Local
    - FAST
    - eFAST
- **Method Parameters**: When a PE or SA method is chosen, the respective method-specific parameters appear and can be set.
- **Abort Criteria**: When a PE or SA method is chosen, the respective method-specific abort criteria appear and can be set.
- **Objective Function**: One of the objective functions in Eq. 5-11 to Eq. 5-14 can be chosen.
- **Model Parameters**: If the settings above have been made, the parameters of the model can be loaded to the GUI by clicking *load parameters*. Afterwards, the parameters which should be optimized can be selected. Additionally, a minimum and a maximum value can

be determined for each parameter. Local PE and SA methods require a start value which can be also entered into the GUI. All these values can also be loaded from an Excel file by clicking *load start*, *load min*, and *load max*, respectively. Then an Excel file has to be chosen from a dialog, the selected Excel file appears, and the respective values can be marked. Thereby, the first column has to contain the parameter names and the second the respective values. Afterwards, the selection is confirmed by clicking *OK* and the values appear in the GUI.

- **scale**: If the scale option is chosen, the parameters are scaled for the optimization to the interval [0,1] by the defined minimum and maximum values. For the simulation, the parameters are then re-scaled by the equation

$$p_{sim} = p^l + (p^u - p^l) \cdot p_{opt}. \qquad \text{Eq. A1}$$

- **START**: Starts the PE or SA method.
- **STOP**: Stops the PE or SA method when the current iteration is finished.

A4 Supplements to the Application

A4-1 Pseudo Experimental Data

Table A6: Data set 1 of the unstructured model in Figure 7.7

time	S	O2	P	N	X
0.00	35.00000	0.00020	0.10000	0.13000	0.10000
5.50	34.44110	0.00019	0.29307	0.12458	0.13294
15.00	33.01345	0.00018	0.78760	0.11079	0.21665
30.00	28.91399	0.00019	2.24482	0.07318	0.44510
40.00	24.62684	0.00019	3.89286	0.04041	0.64410
50.00	19.57404	0.00019	6.08707	0.01514	0.79756
60.00	14.40804	0.00018	8.60533	0.00388	0.86592
70.00	9.31151	0.00018	11.23984	0.00074	0.88501
80.00	4.25147	0.00018	13.90244	0.00011	0.88882
90.00	0.00000	0.00018	16.14781	0.00002	0.88938
100.00	0.00000	0.00018	16.14781	0.00002	0.88938

Table A7: Data set 2 of the unstructured model in Figure 7.7

time	S	O2	P	N	X
0.00	30.00000	0.00020	0.10000	0.25000	0.10000
5.50	28.99639	0.00017	0.34012	0.23461	0.19347
10.00	27.51758	0.00015	0.69549	0.21201	0.33071
12.00	26.56968	0.00014	0.92532	0.19763	0.41803
15.00	24.70132	0.00011	1.38626	0.16971	0.58761
18.00	22.21479	0.00008	2.02528	0.13390	0.80507
20.00	20.22229	0.00006	2.56898	0.10688	0.96916
25.00	14.61558	0.00016	4.35065	0.04420	1.34984
30.00	9.25877	0.00017	6.56417	0.01142	1.54887
40.00	0.00000	0.00017	11.24895	0.00034	1.61618
50.00	0.00000	0.00017	11.24895	0.00034	1.61618

Table A8: Data set 3 of the unstructured model in Figure 7.7

time	S	O2	P	N	X
0.00	45.00000	0.00020	0.10000	0.28000	0.10000
5.50	44.53396	0.00019	0.28194	0.27660	0.12065
15.00	43.43544	0.00018	0.69702	0.26785	0.17378
30.00	40.19073	0.00019	1.83474	0.23731	0.35927
40.00	35.50434	0.00018	3.33845	0.18575	0.67236
50.00	26.27887	0.00017	6.26821	0.08257	1.29897
60.00	14.64387	0.00017	11.02104	0.00849	1.74890
70.00	4.20827	0.00017	16.38084	0.00022	1.79912
80.00	0.00000	0.00017	18.60166	0.00004	1.80022
90.00	0.00000	0.00017	18.60166	0.00004	1.80022
100.00	0.00000	0.00017	18.60166	0.00004	1.80022

Table A9: Data set 4 of the unstructured model in Figure 7.7

time	S	O2	P	N	X
0.00	20.00000	0.00020	0.10000	0.28000	0.10000
5.50	18.29181	0.00015	0.40994	0.24855	0.29097
10.00	14.94226	0.00008	1.04803	0.18850	0.65569
12.00	12.60940	0.00004	1.52990	0.14865	0.89766
15.00	8.35287	0.00000	2.54730	0.08328	1.29469
18.00	4.10522	0.00000	3.87824	0.03478	1.58922
20.00	1.56199	0.00000	4.87668	0.01643	1.70063
25.00	0.00000	0.00016	5.57023	0.00943	1.74317
30.00	0.00000	0.00017	5.57023	0.00943	1.74317
40.00	0.00000	0.00017	5.57023	0.00943	1.74317
50.00	0.00000	0.00017	5.57023	0.00943	1.74317

Table A10: Data set 5 of the unstructured model in Figure 7.7

time	S	O2	P	N	X
0.00	30.00000	0.00020	0.10000	0.12000	0.10000
5.50	29.37397	0.00019	0.30067	0.11310	0.14193
15.00	27.66680	0.00017	0.85827	0.09482	0.25293
30.00	22.70978	0.00019	2.61081	0.04881	0.53232
40.00	18.11011	0.00019	4.49995	0.02007	0.70690
50.00	13.28085	0.00019	6.77984	0.00560	0.79472
60.00	8.51452	0.00019	9.21539	0.00117	0.82168
70.00	3.79404	0.00019	11.69204	0.00019	0.82759
80.00	0.00000	0.00018	13.69445	0.00004	0.82852
90.00	0.00000	0.00018	13.69445	0.00004	0.82852
100.00	0.00000	0.00018	13.69445	0.00004	0.82852

Table A11: Weightings of data points for PE of the unstructured model in Figure 7.7

Data Set	S	O2	P	N	X
1	0.00082	25000000	0.00384	59.17160	1.26423
2	0.00111	25000000	0.00790	16.00000	0.38284
3	0.00049	25000000	0.00289	12.75510	0.30857
4	0.00250	25000000	0.03223	12.75510	0.32909
5	0.00111	25000000	0.00533	69.44444	1.45677

A4-2 Method-specific Parameters

Table A12: Settings of the PE methods for the unstructured model in Figure 7.7

Method	Parameters
ES	$\lambda = 35$, $\mu = 5$, $\varepsilon_{ES} = 10^{-6}$, $g_{max} = 1000$
CMAES	$\varepsilon_{CMAES} = 10^{-8}$, $g_{max} = 1500$
Hybrid1 (ES+Nelder-Mead simplex method)	$\lambda = 35$, $\mu = 5$, $g_{max,global} = 1000$, $g_{max,local} = 3000$, $\varepsilon_{ES} = 10^{-8}$, $\rho_H = 0.01$
Hybrid2 (ES+Hooke-Jeeves method)	$\lambda = 35$, $\mu = 5$, $g_{max,global} = 1000$, $g_{max,local} = 1000$, $\varepsilon_{ES} = 10^{-8}$, $\rho_H = 0.01$, $\rho_{HJ} = 0.7$, $\varepsilon_{HJ} = 10^{-6}$
Hybrid3 (CMAES+Nelder-Mead simplex method)	$g_{max,global} = 1500$, $g_{max,local} = 3000$, $\varepsilon_{CMAES} = 10^{-8}$, $\rho_H = 0.001$
Hybrid4 (CMAES+Hooke-Jeeves method)	$g_{max,global} = 1500$, $g_{max,local} = 1000$, $\varepsilon_{CMAES} = 10^{-8}$, $\rho_H = 0.001$, $\rho_{HJ} = 0.7$, $\varepsilon_{HJ} = 10^{-6}$

DIRECT	$\varepsilon_D = 0.0001$, $g_{max} = 250$

Table A13: Settings of the PO methods for the metabolically structured model in Figure 7.12

Method	Parameters
ES	$\lambda = 28$, $\mu = 4$, $\varepsilon_{ES} = 10^{-5}$, $g_{max} = 500$
CMAES	$\varepsilon_{CMAES} = 10^{-5}$, $g_{max} = 500$
Hybrid1 (ES+Nelder-Mead simplex method)	$\lambda = 28$, $\mu = 4$, $g_{max,global} = 500$, $g_{max,local} = 2000$, $\varepsilon_{ES} = 10^{-5}$, $\rho_H = 0.01$
Hybrid2 (ES+Hooke-Jeeves method)	$\lambda = 28$, $\mu = 4$, $g_{max,global} = 500$, $g_{max,local} = 500$, $\varepsilon_{ES} = 10^{-5}$, $\rho_H = 0.01$, $\rho_{HJ} = 0.7$, $\varepsilon_{HJ} = 10^{-6}$
Hybrid3 (CMAES+Nelder-Mead simplex method)	$g_{max,global} = 500$, $g_{max,local} = 2000$, $\varepsilon_{CMAES} = 10^{-5}$, $\rho_H = 0.001$
Hybrid4 (CMAES+Hooke-Jeeves method)	$g_{max,global} = 500$, $g_{max,local} = 500$, $\varepsilon_{CMAES} = 10^{-5}$, $\rho_H = 0.001$, $\rho_{HJ} = 0.7$, $\varepsilon_{HJ} = 10^{-6}$
DIRECT	$\varepsilon_D = 10^{-4}$, $g_{max} = 250$

Table A14: Adapted settings of the PO methods for the modified metabolically structured model in Figure 7.12

Method	Parameters
ES	$\lambda = 35$, $\mu = 5$, $g_{max} = 1000$
CMAES	$g_{max} = 2000$
Hybrid1 (ES+Nelder-Mead simplex method)	$\lambda = 35$, $\mu = 5$, $g_{max,global} = 100$, $g_{max,local} = 3000$
Hybrid2 (ES+Hooke-Jeeves method)	$\lambda = 35$, $\mu = 5$, $g_{max,global} = 1000$
Hybrid3 (CMAES+Nelder-Mead simplex method)	$g_{max,global} = 2000$, $g_{max,local} = 3000$, $\rho_H = 0.0001$
Hybrid4 (CMAES+Hooke-Jeeves method)	$g_{max,global} = 1000$, , $\rho_H = 0.0001$
DIRECT	-

A4-3 Components and Parameters of the Models

Table A15:Components of the metabolically structured model in Figure 7.12

Name	Type	Description	Properties
N	Continuous place	Nitrogen concentration	$startMarks = N0$ (Initial nitrogen concentration)
X	Continuous place	Biomass concentration	$startMarks = 0.1$ (Initial biomass concentration)
S	Continuous place	Glucose concentration	$startMarks = S0$ (Initial glucose concentration)
ATP	Continuous place	ATP concentration	$startMarks = 1$ (Initial ATP concentration)
P	Continuous place	Xanthan concentration	$startMarks = 0.1$ (Initial xanthan concentration)
$CO2$	Continuous place	CO_2 concentration	$startMarks = 0$ (Initial CO_2 concentration)
Cof	Continuous place	Cofactor concentration	$startMarks = 1$ (Initial cofactor concentration)
$O2$	Continuous place	O_2 concentration	$startMarks = O2^*$ (Initial oxygen concentration)
TX	Continuous transition	Biomass production from nitrogen, glucose, and oxygen according to Eq. 7-28	inputs: $N, S, O2$ outputs: X maximum speed: $r_x = \mu_{max} \cdot X \cdot N \cdot \left(1 - \frac{S}{S_{max}}\right) \cdot \left(1 - \frac{\mu_{eff}}{\mu_{eff,max}}\right)$ input arc weights: $\left(\frac{1}{Y_{XN}}, \frac{1}{Y_{XS}}, \frac{1}{Y_{XO}}\right)$ output arc weights: (1)
$T1$	Continuous transition	Xanthan production from glucose	inputs: $S, ATP, O2$ outputs: $P, Cof, CO2$ maximum speed: $r_1 = \left(\frac{1 + 4Y_{ATPCof}}{3.58 + 4Y_{ATPP}}\right) r_4 \left(1 - \frac{N}{N_{max}}\right)$ input arc weights: $(5.49 \cdot 180, 10.89, 0.3)$ output arc weights: $(923.2, 3.58, 0.6)$
$T2$	Continuous transition	Glucose catabolism	inputs: S outputs: $ATP, Cof, CO2$ maximum speed: $r_2 = \frac{1}{12}\left(1 - 3.58\left(\frac{1 + 4Y_{ATPCof}}{3.58 + 4Y_{ATPP}}\right)\right) r_4$ input arc weights: (180) output arc weights: $(3, 12, 6)$

$T3$	Continuous transition	Maintenance energy	inputs: ATP outputs: - maximum speed: $r_3 = (Y_{ATPP} - 10.89)r_1$ input arc weights: (1) output arc weights: -
$T4$	Continuous transition	Oxidative phosphorylation	Inputs: $Cof, O2$ Outputs: ATP maximum speed: $r_4 = k_4 \cdot O_2 \cdot X$ input arc weights: $(1, 1)$ output arc weights: $\left(Y_{ATPCof}\right)$ $Y_{ATPCof} = k_4' \cdot O_2$
TO	Continuous transition	Oxygen mass transfer	Inputs: - Outputs: $O2$ maximum speed: $r_{OTR} = k_L a \cdot (O_2^* - O_2)$ input arc weights: - output arc weights: (1) $k_L a = 3.08 \cdot 10^{-3} \cdot V_s^{0.43} \cdot R^{1.75} \cdot \mu_{eff}^{-0.39}$
$Fermenter$	Fermenter wrapper	Mode of fermentation Section 6.2	Inputs: - Outputs: N, S Parameters: $mode\ =\ \text{repeated batch fixed feeding number}$ $Vstart\ =\ 4$ (start volume) $Cadd\ =\ \{Nadd, Sadd\}$ (substrate concentration in volume addition) $n\ =\ 1$ (number of feedings) $feedingTimes = \{ft\}$ (feeding point in time) $Vadd\ =\ 1$ (volume addition)
$Inputs$	Inputs wrapper	Wrapper for the input factors stirrer speed (R), temperature (T), and gas flow (Vs)	
R	Steps wrapper	Speed of the stirrer	Parameters: $offset = 3.3$ (initial stirrer speed) $nSteps = 2$ (number of steps) $timePoints = \{22.5, 25\}$ (times when the stirrer speed is increased) $heights = \{4.2, 1.7\}$ (increase of stirrer speed)

T	Step wrapper	Temperature of the fermenter	Parameters: $offset = temp0$ (initial temperature) $startTime = tt$ (time when the temperature is increased) $height = tempHeight$ (increase of temperature)
Vs	Constant wrapper	Gas flow	Parameters: $k = 0.001$ (constant gas flow rate)

Table A16:Parameters of the metabolically structured model in Figure 7.12

Parameter	Description	Value
Y_{XN}	Yield of biomass from nitrogen	6.073
Y_{XS}	Yield of biomass from glucose	$Y_{XS}(T) = a_{XS} + b_{XS}T$ $a_{XS} = 0.34$, $b_{XS} = -0.0061$
Y_{XO}	Yield of biomass from oxygen	$Y_{XO}(T) = \{C_{1,XO}(T - T_{min,XO}) \cdot [1 - \exp(C_{2,XO}(T - T_{max,XO}))]\}^2$ $C_{1,XO}$ =0.571, $C_{2,XO}$ =-0.205, $T_{min,XO}$ = 24.35, $T_{max,XO} = 37.16$
Y_{ATPP}	Consumption of ATP for Xanthan	34
k_4	Rate constant of maximum speed for transition T_4	$k_4(T) = \{C_{1,4}(T - T_{min,4}) \cdot [1 - \exp(C_{2,4}(T - T_{max,4}))]\}^2$ $C_{1,4}$ =0.94, C_{24} =1.399, $T_{min,4}$ =19.45, $T_{max,4}$ =34.25
k'_4	Constant for yield of ATP from cofactors (Y_{ATPCof})	$k'_4(T) = a_4 + b_4T$ $a_4 = -2913$, $b_4 = 2048$
μ_{max}	Rate constant for growth (T_X)	$\mu_{max}(T) = \{C_{1,X}(T - T_{min,X}) \cdot [1 - \exp(C_{2,X}(T - T_{max,X}))]\}^2$ $C_{1,X} = 0.0491$, $C_{2,X} = -0.245$, $T_{min,X} = 25$, $T_{max,X} = 37.09$
O_2^*	Saturated oxygen concentration	0.0002
a	Constant for viscosity (Eq. 7-12)	0.08
b	Constant for viscosity (Eq. 7-12)	0.3
S_{max}	Glucose inhibition constant of growth	50
N_{max}	Nitrogen inhibition constant of xanthan production	0.3
$\mu_{eff,max}$	Viscosity inhibition constant of growth	2

LIST OF FIGURES

LIST OF TABLES

List of Abbreviations

A

AG	**A**ctive **G**ene
AMMod	**A**nalysis of **M**odelica **Mod**els
ANOVA	**AN**alysis **O**f **VA**riance

B

BLT	**B**lock **L**ower **T**riangular form
Bound A,B, C, D	Bounds of process parameters

C

c	**c**ontinuous
C1	Speed calculation
C2	Marks calculation
C3	Activation
C4	Enabling
C5	Firability
C6	Preliminary speed calculation
C7	Decreasing factor calculation
C8	Instantaneous speed calculation
C/N ratio	**C**arbon to **N**itrogen **ratio**
CMAES	**C**ovariance **M**atrix **A**daption **E**volution **S**trategy
Cof	Cofactors

D

d	**d**iscrete
D1	Activation
D2	Output Enabling
D3	Input Enabling
D4	Firability
D5	Token/marks recalculation
DAE	**D**ifferential **A**lgebraic **E**quation
DAEs	**D**ifferential **A**lgebraic **E**quation **s**ystem

DASSL	**D**ifferential **A**lgebraic **S**ystem **S**o**l**ver
dft1	Time from fermentation start to first feeding at $t_1 = dft1$
dft2	Time from first feeding at t_1 to second feeding at $t_2 = t_1 + dft2$
dft3	Time from second feeding at t_2 to third feeding at $t_3 = t_2 + dft3$
dft4	Time from third feeding at t_3 to fourth feeding at $t_4 = t_3 + dft4$
dft5	Time from fourth feeding at t_4 to fifth feeding at $t_5 = t_4 + dft5$
DHS	**D**eterministic **H**ybrid **S**imulation
DIRECT	**DI**viding **RECT**angles

E

eFAST	**e**xtended **F**ourier **A**mplitude **S**ensitivity **T**est
EP	**E**xtracellular **P**roteins
ES	**E**volution **S**trategy

F

FA	**F**orming Bases **A**mino Acids
FAST	**F**ourier **A**mplitude **S**ensitivity **T**est
ft	**f**eeding **t**ime

G

GUI	**G**raphical **U**ser **I**nterface

H

HDMR	**H**igh **D**imensional **M**odel **R**epresentation
HJ	**H**ooke-**J**eeves method
HFPNe	**H**ybrid **F**unctional **P**etri **N**et with **e**xtensions
hybrid DAEs	**hybrid D**ifferential **A**lgebraic **E**quation **s**ystem
Hybrid1	ES+Nelder-Mead simplex method
Hybrid2	ES+Hooke-Jeeves method
Hybrid3	CMAES+Nelder-Mead simplex method
Hybrid4	CMAES+Hooke-Jeeves method

I

IG	**I**nactive **G**ene
IP	**I**ntracellular **P**roteins

M

ma	**m**aximum, **a**bsolute objective function
MM	**M**athematical **M**odeling
MOP1	**M**odified **O**ptimization **P**roblem **1**
MOP1C	**M**odified **O**ptimization **P**roblem **1** with Bound **C**
MOP1D	**M**odified **O**ptimization **P**roblem **1** with Bound **D**
MOP2	**M**odified **O**ptimization **P**roblem **2**
MOP2C	**M**odified **O**ptimization **P**roblem **2** with Bound **C**
MOP2D	**M**odified **O**ptimization **P**roblem **2** with Bound **D**
MP	**M**odel **P**rediction
MPC	**M**odel **P**redictive **C**ontrol
MR	**M**odel **R**eduction
ms	**m**aximum, **s**quared objective function

N

N	Nitrogen concentration
N0	Initial nitrogen concentration
Nadd	**N**itrogen **add**ition
NFA	**N**on-**F**orming Bases **A**mino Acids
NLP	**N**on-**L**inear Programming **P**roblem
NMS	**N**elder-**M**ead **S**implex method

O

OAT	**o**ne-**a**t-a-**t**ime
O2	Oxygen concentration
O20	Initial oxygen concentration
ODE	**O**rdinary **D**ifferential **E**quation
OP1	**O**ptimization **P**roblem **1**
OP1A	**O**ptimization **P**roblem **1** with Bound **A**
OP1B	**O**ptimization **P**roblem **1** with Bound **B**
OP1C	**O**ptimization **P**roblem **1** with Bound **C**
OP2	**O**ptimization **P**roblem **2**
OP2A	**O**ptimization **P**roblem **2** with Bound **A**

OP2B	**O**ptimization **P**roblem **2** with Bound **B**
OP2C	**O**ptimization **P**roblem **2** with Bound **C**

P

P	Xanthan concentration (**P**roduct) **P**rotein
PC	**C**ontinuous **P**lace
PD	**D**iscrete **P**lace
PE	**P**arameter **E**stimation
PESA	**P**arameter **E**stimation and **S**ensitivity **A**nalysis
PNlib	**P**etri **N**et **lib**rary
PNproBio	**P**etri **N**ets for **pro**cess modeling of **Bio**logical systems
PO	**P**rocess **O**ptimization
POSA	**P**rocess **O**ptimization and **S**ensitivity **A**nalysis
PRA	**P**reprocessing and **R**elationship **A**nalysis

R

RA	**R**elationship **A**nalysis

S

S	Glucose concentration Carbon source concentration
s	**s**tochastic
S0	Initial glucose concentration
SA	**S**ensitivity **A**nalysis
SaaS	Software as a Service of Cell Illustrator
Sadd	Glucose addition
Sadd11	Nitrogen concentration of feeding at time $t_1 = dft1$
Sadd12	Glucose concentration of feeding at time $t_1 = dft1$
Sadd21	Nitrogen concentration of feeding at time $t_2 = t_1 + dft2$
Sadd22	Glucose concentration of feeding at time $t_2 = t_1 + dft2$
Sadd31	Nitrogen concentration of feeding at time $t_3 = t_2 + dft3$
Sadd32	Glucose concentration of feeding at time

	$t_3 = t_2 + dft3$
Sadd41	Nitrogen concentration of feeding at time $t_4 = t_3 + dft4$
Sadd42	Glucose concentration of feeding at time $t_4 = t_3 + dft4$
Sadd51	Nitrogen concentration of feeding at time $t_5 = t_4 + dft4$
Sadd52	Glucose concentration of feeding at time $t_5 = t_4 + dft4$
sa	**s**um, **a**bsolute objective function
SECG	Simulation Engine Code Generator of Cell Illustrator
SHS	**S**tochastic **H**ybrid **S**imulation
SPN	**S**tochastic **P**etri **N**et
ss	**s**um, **s**quared objective function

T

TC	**C**ontinuous **T**ransition
TD	**D**iscrete **T**ransition
temp0	Initial **temp**erature
tempHeight	Increase of temperature
TS	**S**tochastic **T**ransition
tt	**t**ime of **t**emperature increase

X

X	Biomass concentration Cell concentration
xHPN	e**x**tended **H**ybrid **P**etri **N**et
xHPNbio	e**x**tended **H**ybrid **P**etri **N**ets for **bio**logical Applications

Y

YXN	Yield of biomass from nitrogen
YXS	Yield of biomass from carbon
YXO	Yield of biomass from oxygen
YPS	Yield of xanthan from carbon
YATPP	Utilized amount of ATP per xanthan unit
YATPCof	Yield of ATP from cofactors

LIST OF REFERENCES

A

Alla H, David R (1998) Continuous and hybrid Petri nets, Journal of Circuits, Systems, and Computers, 8:159- 188

Amanullah A, Satti S, Nienow AW (1998) Enhancing xanthan fermentations by different modes of glucose feeding. Biotechnology progress 14(2):265–269

Amengual A (2009) A specification of hybrid Petri net semantics for HISim simulator. International Computer Science Institute & Universitat de les Illes Balears

B

Bäck T (1996) Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford University Press, USA

Bäck T, Schwefel H (1993) An overview of evolutionary algorithms for parameter optimization. Evolutionary computation 1(1):1–23

Backhaus K, Erichson B, Plinke W, Weiber R (2003) Multivariate Analysemethoden. Eine anwendugsorientierte Einführung. Springer Verlag, Berlin et al.

Bard Y (1974) Nonlinear parameter estimation. Academic Press, New York

Bell M, Pike MC (1966) Remark on algorithm 178 [E4] direct search. Communications of the ACM 9(9):684–685

Beyer H (1995) Toward a theory of evolution strategies: On the benefits of sex. Evolutionary Computation 3(1):81–111

Beyer H (2001) The theory of evolution strategies. Springer Verlag, Berlin et al.

Beyer H, Schwefel H (2002) Evolution strategies–A comprehensive introduction. Natural computing 1(1):3–52

Brinkrolf C (2011) Realisierung einer neuen Petri-Netz-Simulations- und Analyse-Umgebung in der Systembiologie. Master thesis, Bielefeld University

Blanch HW, Clark DS (1997) Biochemical Engineering. Marcel Dekker, New York

Braun W (2010) Weiterentwicklung der Ereignisbehandlung im OpenModelica Compiler zur Simulation hybrider Modelle. Diploma thesis, University of Applied Sciences Bielefeld

Braun W, Bachmann B, Proß S (2010) Synchronous Events in the OpenModelica Compiler with a Petri Net Library Application. Proceedings of 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools: 63–70

Braun W, Ochel L, Bachmann B (2011) Symbolically Derived Jacobians Using Automatic Differentiation - Enhancement of the OpenModelica Compiler. Proceedings of 8th International Modelica Conference:495-501

BTU Cottbus (2011) Charlie Website. A Tool for the Analysis of Place/Transition Nets. http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Charlie (called 7.8.2012)

C

Calder M, Vyshemirsky V, Gilbert D, Orton R (2006) Analysis of signalling pathways using continuous time Markov chains. Transactions on Computational Systems Biology VI 4220:44–67

Chan K, Saltelli A, Tarantola S (1997) Sensitivity analysis of model output: variance-based methods make the difference. Proceedings of 29th conference on Winter simulation:261-268

Chelouah R, Siarry P (2003) Genetic and Nelder-Mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions. European Journal of Operational Research 148(2):335–348

Chen M, Hofestädt R (2003) Quantitative Petri net model of gene regulated metabolic networks in the cell. In Silico Biology 3(3):347–365

Chen S, Ke J, Chang J (1990) Knowledge representation using fuzzy Petri nets. Knowledge and Data Engineering, IEEE Transactions on 2(3):311–319

Chmiel H, Briechle S (2008) Bioprozesstechnik: Einführung in die Bioverfahrenstechnik. Spektrum Akademischer Verlag, Heidelberg

Cukier RI, Fortuin CM, Shuler KE, Petschek AG, Schaibly JH (1973) Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients. I Theory. The Journal of Chemical Physics 59:3873–3876

Cukier RI, Levine HB, Shuler KE (1978) Nonlinear sensitivity analysis of multiparameter model systems. Journal of Computational Physics 26(1):1–42

Cukier RI, Schaibly JH, Shuler KE (1975) Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients. III. Analysis of the approximations. The Journal of Chemical Physics 63:1140–1149

D

Dassault Systèmes AB (2011) Dymola-Dynamic Modeling Laboratory-User Manual Volume 2, Lund, Sweden

David R, Alla H (1987) Continuous petri nets. Proceedings of 8th European Workshop on Application and Theory of Petri nets:275-294

David R, Alla H (2001) On Hybrid Petri Nets. Discrete Event Dynamic Systems: Theory and Applications(11): 9–40

David R, Alla H (2005) Discrete, continuous, and hybrid Petri nets. Springer Verlag, Berlin Heidelberg

Dixon LC, Szegö GP (1978) The global optimization problem: an introduction. Towards Global Optimization 2:1–15

Doi A, Fujita S, Matsuno H, Nagasaki M, Miyano S (2004) Constructing biological pathway models with hybrid functional Petri nets. In Silico Biology 4(3):271–291

Drath R (2002) Description of hybrid systems by modified petri nets. Modelling, Analysis and Design of Hybrid Systems, LNCIS 279:15-36

Dubois E, Alla H, David R (1994) Continuous Petri net with maximal speeds depending on time. Proceedings of the Fourth International Conference on Computer Integrated Manufacturing and Automation Technology:32–39

Dunn I (2003) Biological reaction engineering: dynamic modelling fundamentals with simulation examples. Wiley-VCH Verlag GmbH & Co. KGaA, Weinheim

F

Fabricius SM (2001) Extensions to the Petri Net Library in Modelica. ETH Zurich, Switzerland

Finkel D (2003) DIRECT optimization algorithm user guide. Center for Research in Scientific Computation, North Carolina State University

Fritzson PA (2004) Principles of object-oriented modeling and simulation with Modelica 2.1. Wiley-IEEE Press

Funahashi H, Yoshida T, Taguchi H (1987) Effect of glucose concentrations on xanthan gum production by xanthomonas campestris. Journal of fermentation technology 65(5):603–606

G

Garcia-Ochoa F, Gomez E (1998) Mass transfer coefficient in stirred tank reactors for xanthan gum solutions. Biochemical engineering journal 1(1):1–10

García-Ochoa F, Santos V, Alcon A (1995) Xanthan gum production: an unstructured kinetic model. Enzyme and Microbial Technology 17(3):206–217

Garcia-Ochoa F, Santos V, Alcon A (1996) Simulation of xanthan gum production by a chemically structured kinetic model. Mathematics and Computers in Simulation 42(2-3):187–195

Garcia-Ochoa F, Santos V, Alcon A (1998) Metabolic structured kinetic model for xanthan production. Enzyme and Microbial Technology 23(1-2):75–82

Garcia-Ochoa F, Santos V, Alcon A (2004a) Structured kinetic model for Xanthomonas campestris growth. Enzyme and Microbial Technology 34(6):583–594

Garcia-Ochoa F, Santos V, Alcon A (2004b) Chemical structured kinetic model for xanthan production. Enzyme and Microbial Technology 35(4):284–292

Garcia-Ochoa F, Santos V, Casas JA, Gomez E (2000) Xanthan gum: production, recovery, and properties. Biotechnology Advances 18(7):549–579

Gibson M, Bruck J (2000) Efficient exact stochastic simulation of chemical systems with many species and many channels. The journal of physical chemistry A 104(9):1876–1889

Gillespie DT (1977) Exact stochastic simulation of coupled chemical reactions. The journal of physical chemistry 81(25):2340–2361

Goss P, Peccoud J (1998) Quantitative modeling of stochastic systems in molecular biology by using stochastic Petri nets. Proceedings of the National Academy of Sciences of the United States of America 95(12):6750-6755

H

Hansen N (2006) The CMA evolution strategy: a comparing review. Towards a new evolutionary computation:75–102

Hansen N (2009) Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed. Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers:2389-2395

Hansen N, Kern S (2004) Evaluating the CMA evolution strategy on multimodal test functions. Parallel Problem Solving from Nature-PPSN VIII. Springer Verlag Berlin Heidelberg

Hansen N, Ostermeier A (2001) Completely derandomized self-adaptation in evolution strategies. Evolutionary computation 9(2):159–195

Hanzalek Z (2003) Continuous Petri nets and polytopes. IEEE International Conference on Systems, Man and Cybernetics:1513-1520

Heiner M, Gilbert D, Donaldson R (2008) Petri nets for systems and synthetic biology. Proceedings of 8th International Conference on Formal Methods for Computational Systems Biology:215–264

Herajy M, Heiner M (2010) Hybrid Petri Nets for Modelling of Hybrid Biochemical Interactions. Proceedings of 17th German Workshop on Algorithms and Tools for Petri Nets:66–79

Hofestädt R (1994) A Petri net application to model metabolic processes. Systems Analysis Modelling Simulation 16(2):113–122

Hofestädt R, Thelen S (1998) Quantitative modeling of biochemical networks. In Silico Biology 1(1):39–53

Hooke R, Jeeves T (1961) Direct Search Solution of Numerical and Statistical Problems. Journal of the ACM 8(2):212–229

J

Janowski S (2008) An integrative bioinformatics solution to visualize and examine biological networks. Master Thesis, Bielefeld University

Jensen K (1987) Coloured petri nets. Petri nets: central models and their properties: 248–299, Springer Verlag, Berlin Heidelberg

Johnsson C, Årzén K-E (1999) Grafchart and grafcet: A comparison between two graphical languages aimed for sequential control applications, Preprints 14th World Congress of IFAC(A): 19-24

Jones D, Perttunen C, Stuckman B (1993) Lipschitzian optimization without the Lipschitz constant. Journal of Optimization Theory and Applications 79(1):157–181

Júlvez J, Mahulea C (2012) SimHPN: A MATLAB toolbox for simulation, analysis and design with hybrid Petri nets. Nonlinear Analysis: Hybrid Systems 6(2):806-817

K

Koda M, Mcrae GJ, Seinfeld JH (1979) Automatic sensitivity analysis of kinetic mechanisms. International Journal of Chemical Kinetics 11(4):427–444

Kolda T, Lewis R, Torczon V (2003) Optimization by Direct Search: New Perspectives on Some Classical and Modern Methods. Optimization 45(3):385–482

Krems M, Bachmann B, Braun W (2010) Enhancement of the OpenModelica Compiler – Analytical calculation of the Jacobian matrix. OpenModelica Annual Workshop

L

Lagarias J, Reeds J, Wright M, Wright P (1999) Convergence properties of the Nelder-Mead simplex method in low dimensions. SIAM Journal on Optimization 9(1):112–147

Le Bail J, Alla H, David R (1991) Hybrid petri nets. Proceedings of the European Control Conference: 1472-1477

Leela JK, Sharma G (2000) Studies on xanthan production from Xanthomonas campestris. Bioprocess Engineering 23(6):687–689

Lo Y, Yang S, Min D (1997) Effects of yeast extract and glucose on xanthan production and cell growth in batch culture of Xanthomonas campestris. Applied microbiology and biotechnology 47(6):689–694

Luong JH, Mulchandani A (1988) Kinetics of biopolymer synthesis: a revisit. Enzyme and Microbial Technology 10(6):326–332

M

Marcotte M, Taherian Hoshahili A, Ramaswamy HS (2001) Rheological properties of selected hydrocolloids as a function of concentration and temperature. Food Research International 34(8):695–703

Marino S, Hogue I, Ray C, Kirschner D (2008) A methodology for performing global uncertainty and sensitivity analysis in systems biology. Journal of Theoretical Biology 254(1):178–196

MathWorks (2011) Curve Fitting Toolbox™ User's Guide

Matsuno H, Doi A, Nagasaki M, Miyano S (2000) Hybrid Petri net representation of gene regulatory network. Pacific Symposium on Biocomputing 5:341-352

Matsuno H, Tanaka Y, Aoshima H, Doi A, Matsui M, Miyano S (2003) Biopathways representation and simulation on hybrid functional Petri net. In Silico Biology 3(3):389–404

McKinnon K (1999) Convergence of the Nelder-Mead simplex method to a nonstationary point. SIAM Journal on Optimization 9(1):148–158

Modelica Association (2010) Modelica - A Unified Object-Oriented Language for Physical Systems Modeling Language Specification Version 3.2

Modelica Association (2011) www.modelica.org (called 7.8.2012)

Moles CG, Mendes P, Banga JR (2003) Parameter Estimation in Biochemical Pathways: A Comparison of Global Optimization Methods. Genome Research 13(11):2467–2474

Moraine RA, Rogovin P (1966) Kinetics of polysaccharide B 1459 fermentation. Biotechnology and Bioengineering 8(4):511–524

Moraine RA, Rogovin P (1973) Kinetics of the xanthan fermentation. Biotechnology and Bioengineering 15(2):225–237

Mosterman PJ, Otter M, Elmqvist H (1998) Modeling Petri nets as local constraint equations for hybrid systems using Modelica. Proceedings of SCS Summer Simulation Conference:314–319

Mulchandani A, Luong JH, Leduy A (1988) Batch kinetics of microbial polysaccharide biosynthesis. Biotechnology and Bioengineering 32(5):639–646

N

Nagasaki M, Saito A, Jeong E, Li C, Kojima K, Ikeda E, Miyano S (2010) Cell Illustrator 4.0: A computational platform for systems biology. In Silico Biology 10(1):5–26

Nagasaki M, Doi A, Matsuno H, Miyano S (2004) A Versatile Petri Net Based Architecture for Modeling and Simulation of Complex Biological Processes. Genome Informatics 15(1), 180-197

Nelder J, Mead R (1965) A simplex method for function minimization. The computer journal 7(4):308-313

Nocedal J, Wright SJ (1999) Numerical optimization. Springer-Verlag, New York Berlin Heidelberg

O

Ochel L, Bachmann B, Braun W (2011) Symbolic Calculation of partial derivatives for linearization of non-linear Modelica models in OpenModelica. OpenModelica Annual Workshop

Olsson H, Otter M, Mattsson S, Elmqvist H (2008) Balanced Models in Modelica 3.0 for Increased Model Quality. Proceedings of 6th International Modelica Conference:21-33

Otter M, Årzén KE, Dressler I (2005) StateGraph-a Modelica library for hierarchical state machines. Proceedings of 4th International Modelica Conference:569-578

P

Petri CA (1962) Kommunikation mit Automaten. Dissertation, Rheinisch-Westfälisches Institut für Instrumentelle Mathematik, Bonn

Petri Net World (2012) http://www.informatik.uni-hamburg.de/TGI/PetriNets/ (called 7.8.2012)

Petzold LR (1982) Description of DASSL: a differential/algebraic system solver. Proceedings of 10th international mathematics and computers simulation congress on systems simulation and scientific computation:65-68

Pinches A, Pallent L (1986) Rate and yield relationships in the production of xanthan gum by batch fermentations using complex and chemically defined growth media. Biotechnology and Bioengineering 28(10):1484–1496

Pons A, Dussap CG, Gros JB (1989) Modelling Xanthomonas campestris batch fermentations in a bubble column. Biotechnology and Bioengineering 33(4):394–405

Proß S, Bachmann B (2009) A Petri Net Library for Modeling Hybrid Systems in OpenModelica. Proceedings of 7th International Modelica Conference:454-462

Proß S, Bachmann B (2011a) An Advanced Environment for Hybrid Modeling and Parameter Identification of Biological Systems. Proceedings of 8th International Modelica Conference:557-571

Proß S, Bachmann B (2011b) An Advanced Environment for Hybrid Modeling of Biological Systems Based on Modelica. Journal of Integrative Bioinformatics(8):1–34

Proß S, Bachmann B (2012a) Hybrid Modelling and Process Optimization of Biological Systems, Proceedings of the MATHMOD Conference, Wien, Austria

Proß S, Bachmann B (2012b) PNlib – An Advanced Petri Library for Hybrid Process Modelling, Modelica Conference, Munich, Germany

Proß S, Bachmann B, Hofestädt R, Niehaus K, Ueckerdt R, Vorhölter FJ, Lutter P (2009) Modeling a Bacterium's Life: A Petri-Net Library in Modelica. Proceedings of 7th International Modelica Conference:463-472

Proß S, Janowski SJ, Bachmann B, Hofestädt R (2012a) A New Object-Oriented Petri Net Simulation Environment Based on Modelica, Winter Simulation Conference, Berlin, Germany

Proß S, Janowski SJ, Bachmann B, Kaltschmidt C, Kaltschmidt B (2012b) PNlib - A Modelica Library for Simulation of Biological Systems based on Extended Hybrid Petri Nets, 3rd International Workshop on Biological Processes & Petri Nets, Hamburg, Germany

Q

Quinlan A (1986) Kinetics of Secondary Metabolite Synthesis in Batch Culture When Two Different Substrates Limit Cell Growth and Metabolite Production: Xanthan Synthesis by Xanthomonas campestrisa. Annals of the New York Academy of Sciences 469(1):259–269

R

Rechenberg I (1971) Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. PhD thesis

Reddy VN, Mavrovouniotis ML, Liebman MN (1993) Petri net representations in metabolic pathways. Proceedings of 1st International Conference on Intelligent Systems for Molecular Biology: 328-336

Reinsch C (1967) Smoothing by spline functions. Numerische Mathematik 10(3):177–183

Reiß T (2002) Systeme des Lebens – Systembiologie. Bundesministerium für Bildung und Forschung. Bonn, Germany

Rohr C, Marwan W, Heiner M (2010) Snoopy—a unifying Petri net framework to investigate biomolecular networks. Bioinformatics 26(7):974

Roseiro JC, Esgalhado ME, Amaral Collaco MT, Emery AN (1992) Medium development for xanthan production. Process biochemistry 27(3):167–175

S

Saltelli A, Tarantola S, Campolongo F, Ratto M (2004) Sensitivity analysis in practice: a guide to assessing scientific models. John Wiley & Sons Inc

Saltelli A, Ratto M, Andres T, Campolongo F, Cariboni J, Gatelli D, Saisana M, Tarantola S (2008) Global sensitivity analysis. The primer. John Wiley

Saltelli A, Chan K, Scott EM (2000) Sensitivity analysis. Wiley New York

Saltelli A, Tarantola S, Chan K (1999) A quantitative model-independent method for global sensitivity analysis of model output. Technometrics 41(1):39–56

Schaibly JH, Shuler KE (1973) Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients. II Applications. The Journal of Chemical Physics 59:3879–3888

Schomburg I, Chang A, Schomburg D (2002) BRENDA, enzyme data and metabolic information. Nucleic acids research 30(1):47

Schwefel H (1975) Evolutionsstrategie und numerische Optimierung. PhD thesis

Schwefel H (1977) Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. Birkhäuser Verlag, Basel Stuttgart

Schwefel H (1987) Collective phenomena in evolutionary systems. Preprints of the 31st Annual Meeting of the International Society for General System Research 2:1025-1033

Schwefel H (1995) Evolution and Optimum Seeking. Sixth-Generation Computer Technology Series. Wiley, New York

Scowen R (1993) Extended BNF-a generic base standard. Software Engineering Standards Symposium

Sessego F, Giua A, Seatzu C (2008) HYPENS: A Matlab Tool for Timed Discrete, Continuous and Hybrid Petri Nets. PETRI NETS 2008, LNCS 5062:419-428

Shu C, Yang S (1991) Kinetics and modeling of temperature effects on batch xanthan gum fermentation. Biotechnology and Bioengineering 37(6):567–574

Shubert B (1972) A sequential method seeking the global maximum of a function. SIAM Journal on Numerical Analysis 9(3):379–388

Sieber V, Wittmann E, Buchholz S (2006) Polysaccharide. Angewandte Mikrobiologie:397–408, Springer Verlag, Berlin Heidelberg New York

Simonoff J (1996) Smoothing methods in statistics. Springer Verlag, New York

Sobol I (1993) Sensitivity analysis for non-linear mathematical models. Mathematical Modelling and Computational Experiment 1(1):407–414

Souw P, Demain A (1979) Nutritional studies on xanthan production by Xanthomonas campestris NRRL B1459. Applied and Environmental Microbiology 37(6):1186

Speers RA, Tung MA (1986) Concentration and temperature dependence of flow behavior of xanthan gum dispersions. Journal of Food Science 51(1):96–98

T

Tait MI, Sutherland IW, Clarke-Sturman AJ (1986) Effect of growth conditions on the production, composition and viscosity of Xanthomonas campestris exopolysaccharide. Microbiology 132(6):1483-1492

Tarjan R (1972) Depth-first Search and Linear Graph Algorithms, SIAM Journal on Computing 1(2):146–160

Torczon V (1997) On the convergence of pattern search algorithms. SIAM Journal on Optimization 7(1):1–25

U

Ueckerdt R (1978) Optimale lineare Modelle für das dynamische Verhalten mechanischer Systeme mit endlicher Anzahl von Freiheitsgraden. PhD thesis

University of Tokyo (2010) Cell IlustratorTM: User Guide

V

Vajda S, Valko P, Turanyi T (1985) Principal component analysis of kinetic models. International journal of chemical kinetics 17(1):55–81

Valk R (1978) Self-modifying nets, a natural extension of Petri nets. Automata, Languages and Programming 62:464–476

Vorhölter F, Schneiker S, Goesmann A, Krause L, Bekel T, Kaiser O, Linke B, Patschkowski T, Rückert C, Schmid J (2008) The genome of Xanthomonas campestris pv. campestris B100 and its use for the reconstruction of metabolic pathways involved in xanthan biosynthesis. Journal of biotechnology 134(1-2):33–45

Vuyst L, Loo J, Vandamme E (1987) Two step fermentation process for improved xanthan production by Xanthomonas campestris NRRL B 1459. Journal of Chemical Technology & Biotechnology 39(4):263–273

W

Weiss R, Ollis D (1980) Extracellular microbial polysaccharides. I. Substrate, biomass, and product kinetic equations for batch xanthan gum fermentation. Biotechnology and Bioengineering 22(4):859–873

Weyl H (1938) Mean motion. American Journal of Mathematics 60(4):889–896

Wiechert W (2002) Modeling and simulation: tools for metabolic engineering. Journal of biotechnology 94(1): 37–63

Wilkinson DJ (2006) Stochastic modelling for systems biology. Chapman & Hall/CRC

X

Xuewu Z, Xin L, Dexiang G, Wei Z, Tong X, Yonghong M (1996) Rheological models for xanthan gum. Journal of food engineering 27(2):203–209

Y

Yen J, Liao J, Lee B, Randolph D (1998) A hybrid approach to modeling metabolic systems using a genetic algorithm and simplex method. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 28(2):173–191