BIOMODELKIT

A Framework for Modular Biomodel-Engineering

DISSERTATION

zur Erlangung des akademischen Grades

doctor rerum naturalium

(Dr. rer. nat.)

genehmigt durch die Fakultät für Naturwissenschaften

der Otto-von-Guericke-Universität Magdeburg

von Dipl.-Ing. Mary-Ann Blätke

geb. am 14.12.1985 in Schönebeck

Gutachter:

Prof. Dr. Wolfgang Marwan

Prof. Dr. Andrzej M. Kierzek

eingereicht am: 11. Juli 2016

verteidigt am: 26. April 2017

Abstract

BioModelKit – A Framework for Modular Biomodel-Engineering
Dipl.-Ing. Mary-Ann Blätke

Systems biology employs models to increase the systems-level understanding of living organisms. Models allow to analyse and predict the behaviour of a biological system and thus help us to understand the underlying mechanisms and the impact of perturbations by e.g. drug treatment. Integrating the quantities of data produced by high-throughput *omic* technologies and disperse knowledge into models is a challenge. Other challenges arise from the integration of different biomolecular network types, missing model documentations, and the representation of multiscale aspects. The application of traditional modelling approaches results into large and complex monolithic models. As suggested by classical engineering, modularization is a concept to reduce the complexity of a system.

This thesis aims to establish a modular modelling framework for biomodel engineering, called BioModelKit, where modules are designed for the purpose of model composition. A module is a molecule-centred Petri net model representing the functionality and interactions of a biomolecule. We purposefully defined mechanistic and causal module types, to capture the different types of networks and *omic* fields. Next to the underlying model, each module is equipped with rich annotations. Defined interfaces among the modules allow the modular composition of models without manual adjustments and thus, the integration of different network types and *omic* data. The algorithmic model mutation of composed models allows mimicking the effect of gene knock-outs and structural mutations of a biomolecule. Modularly composed models can also be transformed into spatial models to represent the movement of biomolecules, the cell geometry, compartments, and the distribution of molecules. Modules can be constructed using direct and reverse engineering approaches. Existing models, e.g SBML or Boolean models, can also be a source to obtain modules. The integration of *omic* data into mechanistically defined models by reverse engineered modules, as well as modules obtained from established models, greatly increases the general applicability of our framework, but also encourages the reuse of data and models.

The BioModelKit framework is supported by a relational database that acts as a file manager, but also explicitly stores the model and the annotation of each module, which facilitates the module versioning. Different module versions representing e.g. competitive hypothesis or abstraction levels of a molecular mechanism can be stored. Through a web-interface, the user can browse, search and inspect modules. Furthermore, the modular model composition, as well as the algorithmic model mutation and spatial transformation can be applied to an *ad hoc* chosen set of modules. Modules can also be submitted, curated and annotated through the website.

The case studies discussed in this thesis demonstrate the applicability and capabilities of our framework to cope with current challenges in computational systems biology. The proposed concepts can be particularly helpful for the integration of *omic* data into models, *in silico* rewiring of models in synthetic biology and *in silico* mutations studies, as well as for model-driven personalized medicine. In summary, the BioModelKit framework is an integrative, universally applicable and versatile framework for biomodel-engineering, which supports advanced modelling and simulation approaches including multiscale modelling, spatial modelling, automatic model generation and mutation at a genome-wide scale.

Zusammenfassung

BioModelKit – A Framework for Modular Biomodel-Engineering
Dipl.-Ing. Mary-Ann Blätke

System Biologie verwendet Modelle, um lebende Organismen in einem integrativen Kontext zu verstehen. Die Repräsentation von *"Omic"*-Daten, verstreuten Wissen, und verschiedenen Netzwerktypen stellen große Herausforderungen in der Modellierung dar. Weitere Herausforderungen ergeben sich aus der mangelhaften Dokumentation bestehender Modelle und der Darstellung von multiskalen Aspekten. Dabei erzeugen traditionelle Modellierungsansätze hochkomplexe monolithische Modelle. Eine klassische ingenieurwissenschaftliche Methode die Komplexität solcher Probleme zu reduzieren ist die Modularisierung.

Mit dieser Arbeit soll ein modulares Modellierungsframework für Biomodelle etabliert werden, genannt BioModelKit, in dem Module speziell für den Zweck der Modelkomposition entworfen werden. Ein Modul ist ein Petri Netz Modell, das die Funktionalität und Interaktionen eines Biomoleküls.Es wurden mechanistische und kausale Modultypen definiert, um die unterschlichen Netzwerktypen und *"Omic"* Level abzudecken. Neben dem unterliegenden Modell verfügt jedes Modul auch über weitreichende Annotationen. Die definierten Schnittstellen der Modulen ermöglichen die Modelkomposition ohne manuelle Korrekturen und somit die Integration verschiedener Netzwerktypen und *"Omic"*-Daten. Die Einführung der Modellmutation erlaubt es, den Effekt von Gen Knock-outs und strukturellen Mutationen eines Biomoleküls im Modell abzubilden. Die erzeugten Modelle lassen sich um räumliche Aspekte erweitern, sodass die Bewegung von Biomolekülen, Zellgeometrie, Kompartmentalisierung und räumlichen Verteilung der Biomoleküle dargestellt werden kann. Module können durch *"direct"* und *"reverse engineering"* Ansätze erzeugt werden, sowie durch die Transformation von bestehenden Modellen, z.B. SBML oder Boolsche Modelle. Die Integration von Modulen erzeugt durch die Modeltransformation und *"Omic"* Daten in mechanistisch Modelle durch *"reverse engineered"* Module steigert die universelle Anwendbarkeit unseres Frameworks, und fördert die Wiederverwendung von Daten und Modellen. Das BioModelKit Framework wird unterstützt durch eine Datenbank, welche explizit das Model und Annotationen der Module speichert. Dies ermöglicht verschiedene Modulversionen zu speichern, erzeugt aufgrund von z.B. konkurrierenden Hypothesen zu einem molekularen Mechanismus oder verschiedenen Abstraktionsebenen. Über die Webschnittstelle hat der Nutzer Zugriff auf die Module. Module können für die Veröffentlichung über die Webschnittstelle eingereicht oder kuriert werden. Die Webschnittstelle dient auch der Modelkomposition, sowie der Anwendung der algorithmischen Modelmutation und räumlichen Transformation. Die Fallstudien in dieser Arbeit verdeutlichen die Anwendbarkeit unseres Frameworks und dessen Stärken im Zusammenhang mit den aktuellen Herausforderungen in der System Biologie. Die vorgestellten Konzepte können besonders hilfreich sein, um *"Omic"*-Daten zu integrieren, synthetische biologische Modelle zuerstellen und *in silico* Modellmutagenese durchzuführen, sowie um die modell-gestützte personalisierte Medizin anzuwenden. Das BioModelKit Framework ein integratives, universelles und vielseitiges Framework zur Erstellung von Biomodellen, welches auch Aspekte der multiskalen Modellierung adressiert.

Statement of Authorship

Herewith, I do solemnly declare that I have written the research thesis entitled

"BioModelkit - A Framework for Modular Biomodel-Engineering"

by myself. I certify that I have not used this research thesis previously. All tools and resources have been acknowledged completely.
Furthermore, I certify that I have not submitted this or another research thesis at any institution to acquire the academic degree *doctor rerum naturalium (Dr. rer. nat.)*.

Hiermit erkläre ich, dass ich die von mir eingereichte Dissertation zu dem Thema:

Biomodelkit - A Framework for Modular Biomodel-Engineering

selbstständig verfasst habe. Diese Dissertation wurde von mir zu vor noch nicht verwendet. Alle Hilfsmittel und Quellen wurden vollständig angegeben.
Weiterhin erkläre ich, dass ich weder diese noch eine andere Arbeit zur Erlangung des akademischen Grades doctor rerum naturalium (Dr. rer. nat.) an anderen Einrichtungen eingereicht habe.

Magdeburg, 11. Juli 2016
Dipl.-Ing. Mary-Ann Blätke

Publications

M. A. Blätke. "Petri-Netz Modellierung mittels eines modularen und hierarchischen Ansatzes mit Anwendung auf nozizeptive Signalkomponenten". Diploma Thesis. Otto–von–Guericke–University Magdeburg, 2010.

M. A. Blätke and W. Marwan. "Modular and hierarchical modelling concept for large biological Petri nets applied to nociception". In: *Proc. 17th German Workshop on Algorithms and Tools for Petri Nets (AWPN 2010)*. Vol. 643. CEUR Workshop Proceedings. CEUR-WS.org, 2010, pp. 42–50.

M. A. Blätke, S. Meyer, C. Stein, et al. "Petri net modeling via a modular and hierarchical approach applied to nociception". In: *Proc. 1st Int. Workshop on Biological Processes & Petri Nets (BioPPN), satellite event of Petri Nets 2010*. Braga, Portugal, 2010, pp. 135–146.

M. A. Blätke, M. Heiner, and W. Marwan. *Tutorial – Petri nets in systems biology*. Tech. rep. Otto–von–Guericke–University Magdeburg, 2011.

M. A. Blätke, S. Meyer, and W. Marwan. "Pain signaling – A case study of the modular Petri net modeling concept with prospect to a protein-oriented modeling platform". In: *Proc. 2nd International Workshop on Biological Processes & Petri Nets (BioPPN), satellite event of Petri NETS 2011*. Vol. 724. CEUR Workshop Proceedings. CEUR-WS.org, 2011, pp. 117–134.

M. A. Blätke, M. Heiner, and W. Marwan. "Predicting Phenotype from Genotype Through Automatically Composed Petri Nets". In: *Proc. 10th International Conference on Computational Methods in Systems Biology (CMSB 2012), London*. Vol. 7605. LNCS/LNBI. Springer, 2012, pp. 87–106.

M. A. Blätke and W. Marwan. "A database-supported modular modelling platform for systems and synthetic biology". In: *Proc. 3rd International Workshop on Biological Processes & Petri Nets (BioPPN), satellite event of Petri NETS 2012*. Vol. 852. CEUR Workshop Proceedings. CEUR-WS.org, 2012, pp. 18–19.

M. A. Blätke, A. Dittrich, C. Rohr, et al. "JAK–STAT Signalling as Example for a Database-Supported Modular Modelling Concept". In: *Proc. 10th International Conference on Computational Methods in Systems Biology (CMSB 2012), London*. Vol. 7605. LNCS/LNBI. Springer, 2012, pp. 362–365.

W. Marwan and M. A. Blätke. "A module-based approach to biomodel engineering with Petri nets". In: *Proc. Winter Simulation Conference (WSC 2012), Berlin*. IEEE, 2012, pp. 3383–3394.

M. A. Blätke, A. Dittrich, C. Rohr, et al. "JAK/STAT signalling - an executable model assembled from molecule-centred modules demonstrating a module-oriented database concept for systems and synthetic biology". In: *Molecular BioSystem* 9.6 (2013), pp. 1290–1307.

M. A. Blätke, C. Rohr, M. Heiner, et al. "A Petri-Net-Based Framework for Biomodel Engineering". In: *Large-Scale Networks in Engineering and Life Sciences*. Modeling and Simulation in Science, Engineering and Technology. Springer International Publishing, 2014, pp. 317–366.

F. Liu, M. A. Blätke, M. Heiner, et al. "Modelling and simulating reaction–diffusion systems using coloured Petri nets". In: *Computers in Biology and Medicine* 53 (2014), pp. 297–308.

M. Beccuti, M. A. Blätke, M. Falk, et al. "*Dictyostelium discoideum*: Aggregation and Synchronisation of Amoebas in Time and Space". In: vol. 4. 11. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015, pp. 138–226.

M. A. Blätke, M. Heiner, and W. Marwan. "BioModel Engineering with Petri Nets". In: *Algebraic and Discrete Mathematical Methods for Modern Biology*. Elsevier Inc., 2015. Chap. 7, pp. 141–193.

M. A. Blätke and C. Rohr. "A coloured Petri net approach for spatial Biomodel Engineering based on the modular model composition framework Biomodelkit". In: *Proc. 6th Int. Workshop on Biological Processes & Petri Nets (BioPPN 2015), satellite event of Petri Nets 2015*. Vol. 1373. CEUR Workshop Proceedings. CEUR-WS.org, 2015, pp. 37–54.

Supervised Master Thesis

H. Bongartz. “Experimentelle Parameterisierung und Validierung eines Petri-Netz Modells der IL-6-induzierten JAK/STAT- Signaltransduktion”. Masterthesis. Otto–von–Guericke–University Magdeburg, 2011.

L. Jehrke. “Modulare Modellierung und graphische Darstellung boolescher Netzwerke mit Hilfe automatisch erzeugter Petri-Netze und ihre Simulation am Beispiel eines genregulatorischen Netzwerkes”. Masterthesis. Otto–von–Guericke–University Magdeburg, 2014.

M. Soldmann. “Transformation monolithischer SBML-Modelle biomolekularer Netzwerke in Petri Netz Module”. Masterthesis. Otto–von–Guericke–University Magdeburg, 2014.

Contents

1 Introduction **21**

1.1 Motivation 21

1.2 Contributions 23

1.3 Fundamental Basics 26

1.4 Organisation of Thesis 26

2 Petri Nets **29**

2.1 The Basics 29

2.1.1 Qualitative Petri Net Properties 33

2.2 Petri Net Framework 35

2.2.1 Stochastic Petri Nets 35

2.2.2 Continuous Petri Nets 36

2.2.3 Generalised Hybrid Petri Nets 37

2.3 Coloured Petri Nets 38

2.4 Tools 42

3 BioModelKit (BMK) Framework **45**

3.1 BMK - Module Definition 48

3.1.1 Module Definition and Terminology 48

3.1.2 Module Types 53

3.1.3 Module Transformation in Petri Nets 59

3.1.4 Relation Between Graph Properties Emerging from the Module Definition 65

3.2 BMK - Module Annotation 70

3.3 BMK - Modular Model Composition 78

3.3.1 Modular Model Composition 78

3.3.2 Instantiation of Modularly Composed Models . 80

3.3.3 Algorithmic Model Mutation 89

3.3.4 Spatial Model Transformation 93

3.4 BMK - Module Construction 116

3.4.1 Direct Engineering Approach 116

3.4.2 Reverse Engineering Approach 116
3.4.3 Transformation of Boolean Gene Regulatory Network Models into Modules and a Modular Network 118
3.4.4 Modular Transformation of SBML Models . . . 125

4 BioModelKit (BMK) Web-Tool 133
4.1 BMK - Web-Interface 134
4.2 BMK - Database 143
4.3 Relation between BMK Datbase and Module Definition 151

5 Case Studies 157
5.1 JAK-STAT Signalling 157
5.2 Signal Components involved in Nociception 161
5.3 Phosphate Regulatory Network in Enterobacteria . . . 163

6 Examples of Application 165
6.1 Integration of *Omic* Data 167
6.2 Synthetic Biology 167
6.3 Mutation Studies 168
6.4 Personalised Medicine 169

7 Conclusion & Outlook 171
7.1 Conclusion 171
7.2 Outlook 175

List of Figures

2.1 Petri net formalism. 30
2.2 Regulatory arcs. 30
2.3 Coarse Nodes and Logical Nodes. 33
2.4 Conceptual Framework. 36
2.5 Coloured Petri net basics. 40

3.1 The BMK framework. 45
3.2 Molecular mechanism of the running example. 47
3.3 Petri net modules of the running example. 63
3.4 Definition of MODULE element. 70
3.5 Definition of INFOLIST element. 72
3.6 Definition of PLACELIST element. 73
3.7 Definition of TRANSITIONLIST element. 74
3.8 Definition of PARAMETERLIST element. 75
3.9 Definition of PUBREFLIST element. 75
3.10 Definition of DBREFLIST element. 76
3.11 A modularly composed model of the running example. 79
3.12 Model instantiation using coloured Petri nets. 86
3.13 Discrimination of component instances involved in homodimerization and intermolecular interactions. 88
3.14 Model mutation applied to the running example. 92
3.15 Representation of biomolecular mechanisms and movement in a coherent model. 94
3.16 Definition of coordinate places. 97
3.17 Local constraint of interaction. 98
3.18 Movement of components as single entities. 100
3.19 Movement of single interaction site complexes. 103
3.20 Movement of multi interaction site complexes. 106
3.21 Spatial model (Part I). 110
3.22 Spatial model (Part II). 111
3.23 Definition of nested compartments. 113
3.24 Gene prototype module. 117
3.25 mRNA prototype module. 118
3.26 Allelic influence modules. 118
3.27 Boolean model. 120
3.28 Modular Petri net transformation of a Boolean network. 122
3.29 Boolean model mutation. 123
3.30 Boolean Gene modules of the Boolean model for hierarchical differentiation of myeloid progenitors. 124

3.31 State space of the hierarchical differentiation of myeloid progenitors. 125
3.32 XSD of additional SBML elements for the modular Petri net transformation. 128
3.33 KEGG map for inflammatory mediator regulation of TRP channels. 129
3.34 Modular Petri net transformation of the SBML model for PKCKϵ. 131
3.35 Modifications of the PKC SBML model. 132

4.1 BMK database. 133
4.2 Home view. 134
4.3 Navigation scheme of the BMK web-interface. 135
4.4 Module list view. 136
4.5 Module view. 137
4.6 Place view. 139
4.7 Transition view. 140
4.8 Concepts of the relational model. 143
4.9 Entity relationship model of the BMK database. 145

5.1 Molecular model of IL-6 induced JAK/STAT signalling. 158
5.2 The gp130 protein module. 159
5.3 Simulation of the modularly composed JAK/STAT signalling model and comparison with experimental data. 160
5.4 Modular nociceptive signalling model. 162
5.5 Phosphate regulatory network in enterobacteria. 163
5.6 Modular phosphate regulatory network model. 164

6.1 Generation of alternative models. 165
6.2 Alternative models of the JAK-STAT pathway. 166
6.3 Synthetic and synthetically rewired networks by combing modules with altered interface networks. 167
6.4 *In silico* mutagenesis. 169
6.5 Design of patient-specific models. 170

List of Tables

3.1 Examples of functional units. 50
3.2 Molecular Structure of the modules in the running example. 58
3.3 Molecular events of the modules in the running example. 59
3.4 Mapping of places and molecular states. 64
3.5 Mapping of transitions and molecular events. 64
3.6 Graph properties of the running example. 67
3.7 P- and T-invariants of the running example. 68
3.8 External reference databases. 77
3.9 Mutation types. 89

To my loved ones.

1
Introduction

1.1 Motivation

In the mid-1990s, systems biology [31] emerged as a new scientific discipline in bioscience research aiming to achieve a systems-level understanding of living organisms. The application of the obtained knowledge facilitates advances in various fields, including biotechnology and medicine. In this context, systems-level approaches presume a holistic viewpoint to investigate complex interactions among components and their resulting behaviour. Such investigations require the in-depth analysis of molecular and genetic findings. Based on those requirements, systems biology mainly encouraged the development of high-throughput '*omic*' technologies, such as genomics, proteomics, and metabolomics. The main challenges arising in systems biology are the complexity of investigated systems, the tremendous quantities of data produced by *omic* technologies, and scattered pieces of knowledge; which all need to be integrated. [110] Two approaches tackling these challenges are databases and models integrating complex data and knowledge. Besides, models enable the analysis and prediction of the systems behaviour to explain the underlying molecular mechanisms and estimate the impact of perturbations, e.g. mutations, drug treatments, environmental changes, in a biological system *in silico*.

In many cases, models restrict themselves to a particular network type, e.g. protein-protein interactions networks, gene regulatory networks, metabolic networks or signalling networks. But for a true systems-level understanding which systems biology aims at, models need to integrate information on different classes of biomolecules (the genome, the transcriptome, the proteome, the metabolome, etc.) and molecular interaction (e.g. protein-protein or protein-DNA interactions) to obtain a comprehensive view of cellular regulation based on dynamic models with predictive power.

With increasing amounts of data and unravelled magnitude of interactions, models, as monolithic entities, become larger and more complex. Also, the underlying molecular mechanism of a model cannot be considered as hard-wired. Several factors like environmental (experimental) conditions, the cell type, the impact stimulus sensing and responses, or the history of an individual cell might alter the gene expression pattern and thus, the wiring of a molecular network

by adding, deleting or modifying components (see [17] for example). Integrating all these information in a monolithic model is a tremendous challenge. Understanding a model, maintaining and reusing it requires a readily accessible and compact model representation, which also must include reasonable annotations to document and reference a model [54].

There exists a large variety of modelling methods, such as ordinary differential equations, Boolean networks and Petri nets to name only a few, which have already been applied to model numerous biomolecular systems (see [87] for a review). Among them, Petri nets [4] are particularly suitable for modelling molecular networks. Their operational semantics allows for the unambiguous representation of concurrent, asynchronous and dynamic behaviour, which is characteristic for most biomolecular systems. Since the introduction of Petri nets to model metabolic pathways by Reddy et al. [19], Petri nets have been successfully used in many case studies (see [82] for a review). The attractiveness of Petri nets results from:

- the intuitive and graphically appealing modelling language;
- the operational semantics to directly execute a model;
- the option to hierarchically structure models of complex systems;
- the variety of mathematically founded qualitative and quantitative analysis techniques, including different simulation paradigms (stochastic, deterministic, hybrid), structural and behavioural properties; and
- the support by a wealth of computer tools for modelling, simulation and analysis, see also Section 2.4 for popular Petri net tools.

Coloured Petri nets [13] are an extension of standard Petri nets by the properties of a programming language, which even allow to represent complex molecular multiscale networks in a compact and scalable way.

Since the number of models representing functional aspects of a living cell is steadily increasing [53], it is advisable to store and organise models in databases. Storing models in databases facilitates to access models publicly, to reuse and to exchange them in the scientific community. There already exits a number of databases providing quantitative models of biomolecular networks, such as the CellML repository [84], JWS Online [49], SenseLab ModelDB [42], the Database of Quantitative Cellular Signalling (DOCQS) [44], SigPath [45], and BioModels Database [62]. Typically, these databases permit to store models, supplementary files, and related metadata to retrieve models and to assist model versioning. However, models stored in a database are not necessarily well-documented or peer-reviewed, i.e., curated by experts [54].

Modularisation is a fundamental concept in engineering to reduce the complexity of problems by decomposing a system into functional parts with interfaces. Functional parts, called modules, can be created independently of each other; they can be reused and recombined in different regimes, and are in general easier to handle and to maintain.

In systems biology, there are two types of modularisation approaches used: the *a posteriori* decomposition of monolithic models into modular parts is currently the most common one, while the composition of models from *ab initio* designed modular elements has been established only recently [97, 131].

A posteriori decomposition has been performed by modularising existing models according to specific criteria. These criteria might be either defined functionally (physiological phenomenon, common genetic or signal-transduction units, etc.) [55] or arise from the structural features of the network (units without retroactive effects [55], strongly connected components [10], or network invariants [80], etc.). However, *a posteriori* decomposition mainly aims at the analysis rather than at the modelling of complex systems. With the approaches introduced by [92, 103] or [125], the resulting modules can be reused after appropriate manual adjustment.

In the context of synthetic biology Cooling et al. [97] suggested a modular approach, where *in silico* biomodels are composed of *ab initio* built standard virtual parts (SVPs) of rather a general structure that refer to standard biological parts (SBPs) and bio-environmental processes. The SVPs are stored as ordinary differential equations (ODEs) with input and output variables in the CellML database [84] for reuse purposes. But because of the general structure of those modules, input and output variables have to be manually matched to compose synthetic network models.

Likewise, the composition of new models from existing models stored in the above-mentioned computational model databases is hampered by various granularities, different modelling formalisms, the lack of strict vocabulary, naming convention, and model ontologies. Also, the lack of accurate and compatible model interfaces hampers the coupling of individual models.

1.2 Contributions

The aim of this thesis was to establish an integrative modular modelling framework for biomodel engineering, named BioModelKit, where modules in the form of Petri nets are designed for the purpose of model composition and organised in a database.

In particular, the contributions of this thesis are as follows.

Contribution 1: Modular Modelling Concept

The BioModelKit (BMK) framework originally developed during my preceding Diploma thesis was validated in terms of applicability and usefulness. It uses a modular approach to compose coherent models. Modules are biomolecule-centred Petri net models, which are specifically designed for the purpose of model composition. The use of shared interface networks within modules prevents false or error-prone coupling of modules and highlights interactions of molecules

that do occur in the modularly composed model. This allows the generation of composed models by incorporation of alternative, e.g. tissue-specific interactions of biomolecules. The rigorous reuse and recombination of modules greatly facilitates the construction of complex models. Modules can be easily added to or removed from a modularly composed model. Modules in a modularly composed model can also be exchanged by alternative module versions representing competing hypotheses or different abstraction levels, reengineered or rewired modules for example.

Modules can be classified according to their biomolecular component or process they represent. The definition of different module types provides at the same time standardisation and extensive flexibility for the integration of genomic, transcriptomic, proteomic, and metabolomic components in a modularly composed model. Additional module types that represent causal interactions are useful when molecular mechanisms of causal interactions between model components are not or not sufficiently known.

Contribution 2: Module Annotation

Each module is documented as suggested by [54]. The module annotation is encoded in the BMK mark-up language and includes a main reference and information about the modelling process, involved persons, the meaning of nodes used in the Petri net graph, references to publications or other relevant databases, kinetics, and initial conditions. The rich annotation facilitates the understanding of the biomolecular content of a module and provides sufficient information for the execution of the underlying model in a simulation environment.

Contribution 3: Module Construction Approaches

Next to the classical direct engineering approach, modules can be obtained by the transformation of Boolean models of genetic networks or models encoded in the SBML format. The automatic generation of such modules that was implemented during this thesis allows to import existing models into the module repository of the BMK framework and make those models and their parts reusable by integrating them into modularly composed models. The use of prototype modules and causal interaction modules in combination with reverse engineering approaches fosters the generation of new modules through different means and even by automatically compiling genome-wide data sets (*omics* data sets) obtained by high-throughput approaches.

Contribution 4: Algorithmic model mutation

We have also implemented algorithmic mutation of modules and of models derived thereof, which allows mimicking different kinds of mutations such as gene deletions or protein structure mutations. The algorithmic model mutation is designed to deliver exclusively models which are biochemically realistic. Moreover, annotations provided for the nodes of the underlying Petri net model of a module can be

read during the process of model mutation to generate very specific mutations.

Contribution 5: Spatial model transformation
In addition, a modularly composed model can be transformed into a spatial model to account for geometric and spatial properties of a cell, as well as its physical structure with respect to membranes, compartments, pools, etc. Spatially transformed models can be simulated using Snoopy [120] or Marcie [129], and are computationally manageable.

Contribution 6: Web tool including a model database
A web-tool consisting of a database and web-interface supports the BMK framework and provides public access to the concepts and methods introduced in this thesis. The database takes over the handling of modules and organises the module repository by explicitly storing the underlying model structure and related annotations of a module. The database also keeps track of the module versioning. The web-interface accesses the database and provides diverse functionality to the user, e.g. browsing, searching and inspecting modules, but also composing models from a set of chosen modules, applying the algorithmic model mutation or spatial transformation of a module. Trough the web-interface the user can also submit and curate modules.

Prospects.
The BMK framework might be beneficial for several applications, such as the automatic generation of models based on high-throughput data sets, the rewiring and reengineering of models for synthetic biology purposes, and the performance of extensive mutation studies. In particular, integrating rewired modules and high-throughput data using reverse engineering approaches facilitates the generation of personalised, respectively individualised, models. Such models might be of importance in personalised medicine in the near future.

In summary, the proposed BMK framework covers all major types of modelling paradigms (discrete, deterministic, stochastic, hybrid) in systems biology, and allows to integrate all common types of biomodels including Boolean networks. Therefore, the BMK framework is widely if not universally applicable in the field of biomodel engineering in a very versatile manner.

All of the approaches introduced in this thesis have been implemented with the help of Snoopy [120], Charlie [141], and Marcie [129]; and successfully tested for their executability. In particular, this thesis had a significant impact on the consistent further development of Snoopy [120].

1.3 Fundamental Basics

This thesis incorporates concepts and theories from different fields to establish the BMK framework and a supportive web-tool. Throughout this thesis, we chose *Petri nets* as modelling language to create modules as defined in the BMK framework. But we also relate to *Boolean networks* representing gene regulatory processes as an input for the module construction. The web-tool is implemented using HyperText Markup Languages (HTML), Cascading Style Sheets (CSS), Hypertext Preprocessor (PHP), and Javascript. As a relational database management system *MySQL* has been used to design and administrate the database holding modules defined in the BMK framework. Modelling biomolecular systems at a macro-molecular level also requires knowledge about:

- *Biochemistry* - Study of chemical components and processes vital in live organisms with a focus on the roles, functions and structures of the involved components.
- *Genetics* - Study of the effect of genetic differences on organisms.
- *Molecular Biology* - Study of the molecular fundamentals of the processes of replication, transcription, translation, and cell function to understand live.

We also incorporate general concepts employed in systems and computational biology to model and analyse biomolecular systems. This includes standards like the Minimum Information Requested In the Annotation of biochemical Models (MIRIAM) and the Systems Biology Markup Language (SBML). Working with SBML requires knowledge about the concepts of the eXtensible Markup Languages (XML) [22].

A basic understanding of these concepts and theories is needed to conceive the complex relations in the BMK framework. As far as necessary and relevant, we provide a basic introduction to the used concepts and give references for further reading.

1.4 Organisation of Thesis

Since the definition of the modular modelling concept mainly relies on Petri nets, we start with a short but comprehensive introduction of the Petri net framework in Chapter 2. In the following, we will define the modular modelling concept used in the BMK framework in Chapter 3 and then introduce the supportive web-tool, including the BMK database and web-interface in Chapter 4. We separately present additional case studies in Chapter 5 and discuss examples of application in Chapter 6 to allow a continuous reading of the formal concepts and their implementation in the chapters before. At the end in Chapter 7, we provide a conclusion and a short outlook.

Since the BMK framework integrates several concepts in a complex context to establish a general and versatile framework for biomodel engineering, the complete overall picture may not become evident

before the end of this thesis, when the elements are discussed in relation to each other.

Below, we summarise the content of each chapter in more detail:

Chapter 2 We give a short introduction into Petri nets, which is the modelling language of choice used in this thesis due to their operational semantics and outstanding facilities to model and analyse biological systems.

Chapter 3 We introduce the BMK framework for biomodel engineering, where models are composed of a set of molecule-centred models, called modules. We first give a definition of our modularization concept and the underlying Petri net model of a module. Also, we describe the extension of a module with annotations for documentation and reference purposes by proposing the BMK mark-up language. We then explain the modular model composition and variations of it, including the algorithmic mutation of models and spatial extension of modularly composed models. Last, we describe different approaches for the module construction, including direct and reverse engineering approaches, as well as the modular transformation of Boolean and SBML models.

Chapter 4 We present the BMK web-tool consisting of a database and a web-interface to support the automatisation of the BMK framework. Here, the database explicitly stores the underlying model and annotations of a module and keeps track of the module versioning. Through the web-interface, the user can access the modules stored in the database by querying for specific modules or browsing through the module repository. The user can submit and curate modules through the BMK web-tool. The BMK web-tool also provides features to create modularly composed models and to apply the algorithmic model mutation or spatial model transformation to the modularly composed models.

Chapter 5 We give three case studies on JAK-STAT signalling, nociception, and the phosphate regulatory network in enterobacteria to demonstrate the proposed modularization concept.

Chapter 6 We first provide a general overview on generation of alternative modularly composed models by steadily reusing modules in variable combination. Next, we explain more detailed approaches on how to apply the BMK framework to integrate *omic* data into modularly composed models, and to support the investigation of synthetic and synthetically rewired networks, as well as the mutation studies. Last, we also provide a short prospect on applications in personalised medicine.

Chapter 7 We summarise the features and merits of the BMK framework for the modular composition of biomodels and give an outlook.

2
Petri Nets

Petri nets are a formal modelling language to describe distributed systems in any field of application from technical to biological systems. Carl Adam Petri invented Petri nets in August 1939 at the age of 13 and finally documented them as a part of his dissertation "Kommunikation mit Automaten (communication with automata)" in 1962. Using Petri nets to model a system allows representing concurrent, asynchronous and non-deterministic processes in an unambiguous way. In this chapter, we will explain only the basics of Petri nets, a more comprehensive introduction on Petri nets can be found in [133, 138]. In particular, we refer to [82] for terminology and formal definitions on Petri nets. These references do also argue on the applicability and advantageous of Petri nets for modelling and analysing biological systems supported by several examples.

2.1 The Basics

Mathematically, a Petri net is a well-defined directed graph using two sets of nodes, places and transitions, which are connected by directed arcs, see Figure 2.1. Places are the passive nodes of a Petri net representing conditions (objects), where transitions are active nodes representing events (activities). In-going arcs of a transition indicate all conditions (places) acting as a source for the encoded event. Out-going arcs of a transition denote, which conditions (places) are products of the event. Thus, arcs indicate in which direction a process evolves. The arc-weight of an arc defines the quantity that is required or produced of a condition (place) by an event (transition). Tokens define the value of each condition modelled by a place. The token distribution over all places is called marking and represents the current state of the modelled system.

The Petri net formalism employs operational semantics to execute the modelled events and processes. To run an event encoded by a transition all pre-places of a transition must host a sufficient amount of tokens according to the arc-weight, called enabledness. If an enabled transition runs its event, called firing, it deletes the tokens on its pre-places according to the arc-weights and produces new tokens on its post-places, again according to the arc-weights, see Figure 2.1(D). Thus, the firing of a transition updates the current marking to a

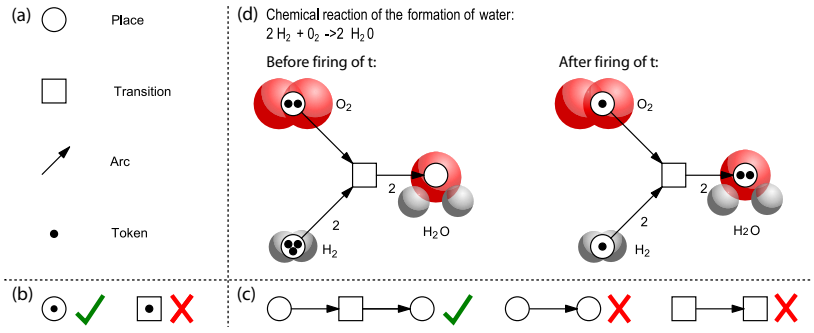


Figure 2.1: Petri net formalism. (A) Elements of a Petri net are places, transitions, arcs and tokens. (B) Only places can host tokens. (C) Nodes of the same type cannot be connected. (D) The Petri net models the chemical formation of water through oxygen (atoms shown in red) and hydrogen (atoms shown in grey). The stoichiometry is given by the arc-weights of the reaction. A transition is enabled and may fire if pre-places host sufficient amounts of tokens. (adapted from [133])

newly reachable one. The firing process itself is an atomic event and consumes no time. The repeated firing of transitions translates into the behaviour of the Petri net. The set of all reachable markings from the initial marking constitutes the state space of the Petri net. Accordingly, qualitative Petri nets ($\mathcal{QPN}$) are defined as [82]:

Definition 2.1 (Qualitative Petri net)**.** A qualitative Petri net is a 5-tuple $\mathcal{N} = \{P, T, F, f, m_0\}$, where:

- P is a finite, non-empty set of places.
- T is a finite, non-empty set of transitions.
- F is a finite set of directed arcs.
- f: $F \to \mathbb{N}_0$ is a function that assigns a non-negative integer to each arc $f(x, y) \in F$; $x, y \in P \cup T$.
- m_0: $P \to \mathbb{N}_0$ gives the initial marking.

So far, there exist several extensions of the $\mathcal{QPN}$ [82]. Regulatory arcs are one of them, which either increase the expressiveness of the

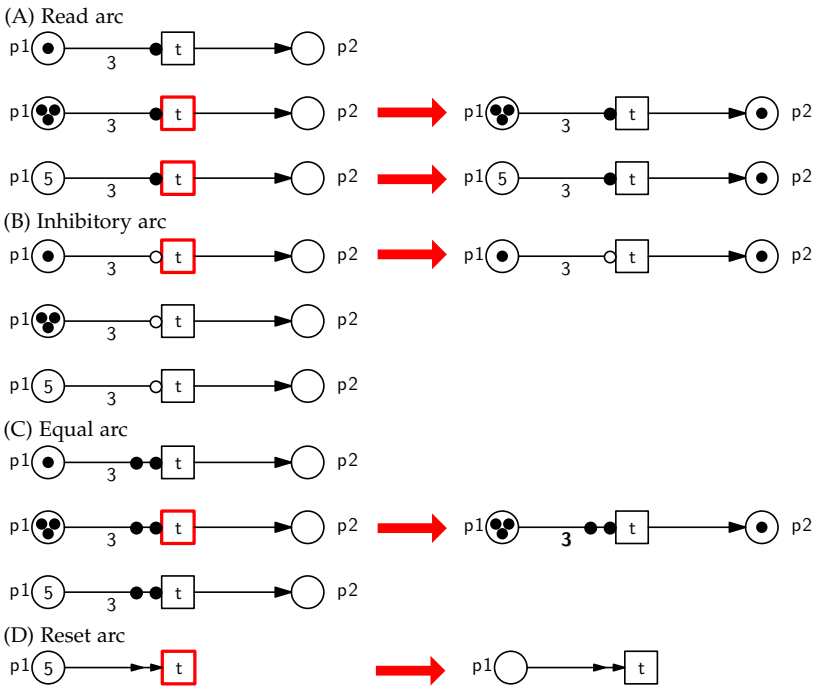


Figure 2.2: Regulatory arcs. Left side shows net before firing of transition t (only transitions marked in red are enabled). Right side shows net after firing of enabled transitions. (A) Read arc - transition t can only fire, if the marking on place p_1 is equal to or greater than the arc-weight. (B) Inhibitory arc - transition t can only fire, if the marking on place p_1 is less than the arc-weight. (C) Equal arc - transition t can only fire, if the marking on place p_1 is equal to the arc-weight. In (A)-(C) token remain on place p_1 after firing of transition t. (D) Reset arc - transition t can fire independent of the marking on place p_1. The firing of transition t deletes all tokens on place p_1.

Petri net formalism or allow a more compact representation. Here, we consider four types of regulatory arcs: read arcs, inhibitory arcs, equal arcs, and reset arcs. All of the regulatory arcs are only allowed to lead from a place p to a transition t. Read, inhibitory, and equal arcs introduce additional constraints to the enabledness of a transition t. In the case of these three arc types, the marking of the respective place $m(p)$ is not affected by the firing of transition t.

- The read arc [21] requires that the marking of place p is not less than the arc-weight to enable transition t, see Figure 2.2(A).
- The inhibitory arc [7] reverses the logic of the read arc. The marking of place p must be less than the arc-weight to enable transition t, see Figure 2.2(B).
- The equal arc [104] narrows the precondition implied by the read arc. Accordingly, the connected transition t may only fire if the marking of place p is equal to the arc-weight, see Figure 2.2(C).
- The reset arc [9] adds no precondition to transition t, the firing of transition t empties place p, see Figure 2.2(D).

Based on Defintion 2.1 for $\mathcal{QPN}$, we define qualitative Petri nets extended by regulatory arcs ($\mathcal{XPN}$) as [82]:

Definition 2.2 (Extended Petri net)**.** A qualitative Petri net is a 5-tuple $\mathcal{N} = \{P, T, F, f, m_0\}$, where:

- P is a finite, non-empty set of places.
- T is a finite, non-empty set of transitions.
- F is a finite set of directed arcs. F is defined as the union of five disjunctive arc sets, i.e. $F := F_{SA} \cup F_{RA} \cup F_{IA} \cup F_{EA} \cup F_{XA}$ with:
  - $F_{SA} \subseteq (P \times T) \cup (T \times P)$ is the set of standard arcs,
  - $F_{RA} \subseteq P \times T$ is the set of read arcs,
  - $F_{IA} \subseteq P \times T$ is the set of inhibitory arcs,
  - $F_{EA} \subseteq P \times T$ is the set of equal arcs, and
  - $F_{XA} \subseteq P \times T$ is the set of reset arcs.
- f: $F \to \mathbb{N}_0$ is a function that assigns a non-negative integer to each arc $f(x, y) \in F$; $x, y \in P \cup T$.
- m_0: $P \to \mathbb{N}_0$ gives the initial marking.

For simplicity we do not further distinguish $\mathcal{QPN}$ and $\mathcal{XPN}$, and refer to $\mathcal{XPN}$ also as $\mathcal{QPN}$. Based on the Definition 2.1 and 2.2, we introduce notations about the marking and pre-sets, respectively post-sets, of nodes. The notation $m(p)$ refers to the number of tokens on place p in the marking m. Place p is clean (empty, unmarked) in m if $m(p) = 0$. A set of places is called clean if all places are clean, otherwise the set is marked. The pre- and postset of a node $x \in P \cup T$, is defined as:

- Preset: $\bullet x := \{y \in P \cup T \mid f(y, x) \neq 0\}$
- Postset: $x\bullet := \{y \in P \cup T \mid f(x, y) \neq 0\}$

For places and transitions, we get four types of sets:

- $\bullet t$ - pre-places of transition t
- $t\bullet$ - post-places of transition t
- $\bullet p$ - pre-transitions of place p
- $p\bullet$ - post-transitions of place p

For a set of nodes $X \subseteq P \cup T$, $\bullet X := \bigcup_{x \in X} \bullet x$ is the set of pre-nodes and $X\bullet := \bigcup_{x \in X} \bullet x$ is the set of post-nodes.

Based on the Definition 2.2 and the notation of pre- and post-sets of nodes, the firing rule of a $\mathcal{QPN}$ can be defined as:

Definition 2.3 (Firing Rule)**.** Let $\mathcal{N} = (P, T, F, f, m_0)$ be a Petri net:

- A transition t is enabled in marking m, written as $m\,[t\rangle$, if
  - $\forall p \in \bullet t : m(p) \geq f(p,t)$, if $f(p,t) \in F_{SA}$,
  - $\forall p \in \bullet t : m(p) \geq f(p,t)$, if $f(p,t) \in F_{RA}$,
  - $\forall p \in \bullet t : m(p) < f(p,t)$, if $f(p,t) \in F_{IA}$, and
  - $\forall p \in \bullet t : m(p) = f(p,t)$, if $f(p,t) \in F_{EA}$;

  else transition t is disabled.
- A transition t, which is enabled in m, may fire.
- When t in m fires, a new marking m' is reached, written as $m\,[t\rangle\,m'$, with $\forall p \in P$:

$$m'(p) = \begin{cases} m(p) - f(p,t) + f(t,p) & \text{, if } f(p,t) \in F_{SA} \\ m(p) + f(t,p) & \text{, if } f(p,t) \in F_{RA} \\ m(p) + f(t,p) & \text{, if } f(p,t) \in F_{IA} \\ m(p) + f(t,p) & \text{, if } f(p,t) \in F_{EA} \\ f(t,p) & \text{, if } f(p,t) \in F_{XA} \end{cases}$$

Additional Visual Features

In addition to the standard Petri net elements and regulatory arcs, some Petri net tools, e.g. Snoopy [120], support two additional distinguished features for the design and systematic construction of larger Petri nets:

- Coarse Nodes allow to structure a Petri net hierarchically, see Figure 2.3(A)-(B). There are two types of coarse nodes. Coarse transitions (two centric squares) stand for transition-bordered subnets (i.e. subnets having only transitions as an interface to the supernet). Likewise, coarse places (two centric circles) stand for place-bordered subnets (i.e. subnets having only places as an interface to the supernet).
- Logical nodes (shaded in grey), also called fusion nodes, allow to represent identical copies of one and the same node, i.e. logical nodes with the same name are logically identical, see Figure 2.3(C)-(D). Logical nodes can exist in two forms: logical places and logical transition. They can be applied to connect remote parts of a net or to split nodes with a high connectivity into copies of identical logical nodes to sort out superimposed arcs.

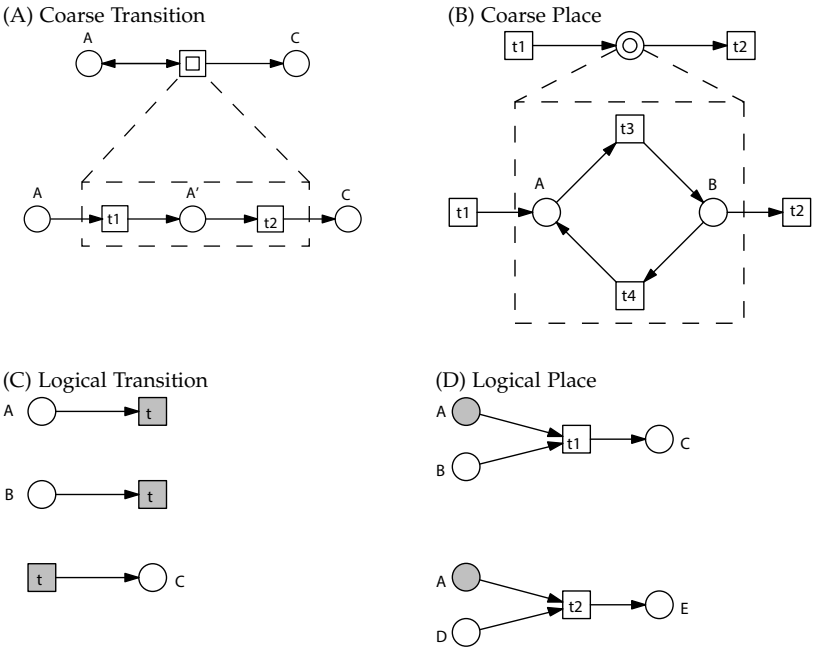


Figure 2.3: Coarse Nodes and Logical Nodes. Coarse transitions (A) and coarse places (B) yield hierarchically structured nets. Logical transitions (C) and logical places (D) connect remote parts of a net.

Both, coarse nodes and logical nodes allow to structure complex Petri nets in a compact and neatly arranged way.

2.1.1 *Qualitative Petri Net Properties*

In the following, we list elementary graph properties of a Petri net to reflect the modelling approach and qualitatively characterize the graph structure, see [82] for detailed explanations of the following notions:

- A Petri net is:
  - ...*pure* if $\forall x, y \in P \cup T : f(x,y) \neq 0 \Rightarrow f(y,x) = 0$.
  - ...*ordinary* if $\forall x, y \in P \cup T : f(x,y) = 1$.
  - ...*homogeneous* if $\forall p \in P : t, t' \in p\bullet \Rightarrow f(p,t) = f(p,t')$.
  - ...*free of blocking multiplicity* if $\forall p \in P : \bullet p \neq \emptyset \wedge min\{f(t,p) \mid \forall t \in \bullet p\} \geq max\{f(p,t) \mid \forall t \in p\bullet\}$.
  - ...*conservative* if $\forall t \in T : \sum_{p \in \bullet t} f(p,t) = \sum_{p \in t\bullet} f(t,p)$.
  - ...*static conflict free* if $\forall t, t' \in T : t \neq t' \Rightarrow \bullet t \cap \bullet t' = \emptyset$.
  - ...*connected* if it holds for every two nodes x and y, $x, y \in P \cup T$, that there is an undirected path between x and y.
  - ...*strongly connected* if it holds for every two nodes x and y, $x, y \in P \cup T$, that there is a directed path from x to y.
- A Petri net has
  - ...*input transitions* (transitions without pre-places) if $\bullet t = \emptyset$.
  - ...*output transitions* (transitions without post-places) if $t\bullet = \emptyset$.
  - ...*input places* (places without pre-transitions) if $\bullet p = \emptyset$.
  - ...*output places* (places without post-transitions) if $p\bullet = \emptyset$.

In Addition to the elementary graph properties, place invariants (P-invariants) and transition invariants (T-invariants) are essential structural features of a Petri net graph, which have a major impact

on the model behaviour. In general, an invariant is a predicate of a system, which is not changed by the processes involved in the system. Concerning places, a P-invariant represents a set of places over which the weighted sum of tokens is constant and independent of any firing. In the case of transitions, a T-invariant stands for a set of transitions. The partially ordered firing of transitions in a T-invariant reproduces the initial state, which enabled the firing of the transitions in the T-invariant. The total effect of the T-invariant on a marking is zero. Below, we give the formal definitions for P- and T-invariants, consider also [82] for more details.

Definition 2.4 (P-invariants, T-invariants)**.**

- The incidence matrix of N is a matrix $\mathbb{C} : P \times T \to \mathbb{Z}$, indexed by P and T, such that $\mathbb{C}(p,t) = f(t,p) - f(p,t)$.
- A place vector (transition vector) is a vector $x : P \to \mathbb{Z}$, indexed by P ($y : T \to \mathbb{Z}$, indexed by T)
- A place vector (transition vector) is called P-invariant (T-invariant) if it is a non-trivial non-negative integer solution of the linear equation system $x \cdot \mathbb{C} = 0$ ($\mathbb{C} \cdot y = 0$).
- The set of nodes corresponding to an invariant's non-zero entries are called the support of this invariant x, written as $supp(x)$.
- An invariant x is called minimal, if $\nexists$ invariant z: $supp(z) \subset supp(x)$, i.e., its support does not contain the support of any other invariant z, and the greatest common divisor of all non-zero entries of x is 1.
- A net is covered by P-invariants (T-invariants), if every place (transition) belongs to a P-invariant (T-invariant).

The general behaviour of a Petri net can be characterized by three properties:

- *Boundedness* - For each place it holds that: Whatever happens, the maximum number of tokens on this place is bounded by a constant. This constraint precludes overflow by an unlimited increase of tokens.
- *Liveness* - For every transition it holds that: Whatever happens, it will always be possible to reach a state, where this transition gets enabled. In a live net, all transitions can contribute to the net behaviour forever, which precludes dead states, i.e., states where none of the transitions is enabled.
- *Reversibility* - For every state it holds that: Whatever happens, the net will always be able to reach this state again. So the net has the capability of self-reinitialization.

These properties can be formulated independently of the special function of the network under consideration. Below, we give the formal notations of these notions, see [82] for detailed explanations.

Definition 2.5 (Boundedness)**.**

- A place p is k-bounded if there exists a positive integer number k, which represents an upper bound for marking of this place in all reachable markings of the Petri net: $\exists k \in \mathbb{N}_0 : \forall m \in [m_0\rangle : m(p) \leq k$.
- A Petri net is k-bounded if all its places are k-bounded.
- A Petri net is structurally bounded if it is bounded in any initial marking.

Definition 2.6 (Liveness)**.**

- A transition t is dead in the marking m if it is not enabled in any marking m' reachable from: $\nexists m' \in [m\rangle : m'(t)$.
- A transition t is live, if it is not dead in any marking reachable from m_0.
- A marking m is dead, if there is no transition enabled in m.
- A Petri net is deadstate-free, if there are no reachable dead markings.
- A Petri net is live, if each transition is live.

Definition 2.7 (Reversibility)**.** A Petri net is reversible if the initial marking can be reached again from each reachable marking: $\forall m \in [m_0\rangle : m_0 \in [m\rangle$.

2.2 *Petri Net Framework*

So far, we considered qualitative Petri nets ($\mathcal{QPN}$) as time-free, i.e. neither are transitions associated with time nor are tokens assigned with a sojourn time. Thus, $\mathcal{QPN}$ consider all possible behaviour under any timing. The extension of $\mathcal{QPN}$ in a quantitative manner by integrating time aspects allows the introduction of extended stochastic Petri nets ($\mathcal{SPN}$), extended continuous Petri nets ($\mathcal{CPN}$), and generalised hybrid Petri nets ($\mathcal{HPN}$), see Figure 2.4.

2.2.1 *Stochastic Petri Nets*

$\mathcal{SPN}$ are an extension of $\mathcal{QPN}$ [82]. In an $\mathcal{SPN}$, see Figure 2.4, each transition of the respective $\mathcal{QPN}$ is associated with a firing rate. The firing rate is a random variable defined by an exponential distribution. The semantics of an $\mathcal{SPN}$ are thus equivalent to a continuous-time Markov chain (CTMC), which can be obtained from the reachability graph of the underlying $\mathcal{QPN}$, where arcs connecting two states are labelled with the transition rate, see also [89] for more details. The $\mathcal{QPN}$ is an abstraction of the $\mathcal{SPN}$, meaning all qualitative properties valid for the $\mathcal{QPN}$ are valid for the $\mathcal{SPN}$ as well, and vice versa. An $\mathcal{SPN}$ is defined as [89]:

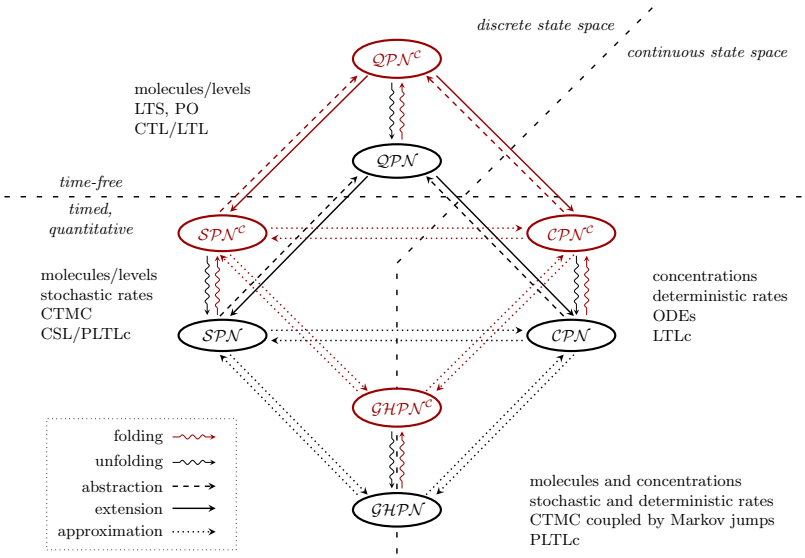


Figure 2.4: Conceptual Framework. The standard low-level Petri net formalism comprises four net classes, qualitative Petri nets ($\mathcal{QPN}$), stochastic Petri nets ($\mathcal{SPN}$), continuous Petri nets ($\mathcal{CPN}$), and generalised hybrid Petri nets ($\mathcal{HPN}$). The Petri net classes differ in their type of state space and their relation in respect to time. Each Petri net class can be derived from one of the others by abstraction, extension, or approximation. Also, all standard Petri net classes can be projected to the high-level coloured Petri net framework. Coloured Petri nets can be obtained by folding of the corresponding standard Petri net class, respectively unfolding a coloured Petri nets yields a standard Petri nets. (taken from [121])

Definition 2.8 (Stochastic Petri Net)**.** A stochastic Petri net is a 6-tuple $\mathcal{N} = \{P, T, F, f, v, m_0\}$, where:

- P is a finite, non-empty set of places.
- T is a finite, non-empty set of transitions.
- F is a finite set of directed arcs. F is defined as the union of five disjunctive arc sets, i.e. $F := F_{SA} \cup F_{RA} \cup F_{IA} \cup F_{EA} \cup F_{XA}$ with:
  - $F_{SA} \subseteq (P \times T) \cup (T \times P)$ is the set of standard arcs,
  - $F_{RA} \subseteq P \times T$ is the set of read arcs,
  - $F_{IA} \subseteq P \times T$ is the set of inhibitory arcs,
  - $F_{EA} \subseteq P \times T$ is the set of equal arcs, and
  - $F_{XA} \subseteq P \times T$ is the set of reset arcs.
- f: $F \to \mathbb{N}_0$ is a function that assigns a non-negative integer to each arc $f(x, y) \in F$; $x, y \in P \cup T$.
- v: $T \to H$ is a function that assigns a stochastic hazard function $h(t)$ to each transition $t \in T$, whereby $H := \bigcup_{t \in T} \{h(t) \mid h(t) : \mathbb{N}_0^{\bullet t} \to \mathbb{R}^+\}$ is the set of all stochastic hazard functions, and $v(t) = h(t)$ for all transitions T.
- m_0: $P \to \mathbb{N}_0$ gives the initial marking.

2.2.2 *Continuous Petri Nets*

In the $\mathcal{CPN}$, see Figure 2.4, each transition of the respective $\mathcal{QPN}$ is associated with a deterministic rate, and the discrete values of the places are replaced with continuous values. Thus, the $\mathcal{CPN}$ gives a structured description of a set of ordinary differential equations (ODEs), which describes the overall behaviour of a modelled system. Accordingly, the state space of a $\mathcal{CPN}$ is continuous and linear [67]. An $\mathcal{CPN}$ is defined as [89]:

Definition 2.9 (Continuous Petri Net)**.** A continuous Petri net is a 6-tuple $\mathcal{N} = \{P, T, F, f, v, m_0\}$, where:

- P is a finite, non-empty set of places.
- T is a finite, non-empty set of transitions.
- F is a finite set of directed arcs. F is defined as the union of five disjunctive arc sets, i.e. $F := F_{SA} \cup F_{RA} \cup F_{IA} \cup F_{EA} \cup F_{XA}$ with:
  - $F_{SA} \subseteq (P \times T) \cup (T \times P)$ is the set of standard arcs,
  - $F_{RA} \subseteq P \times T$ is the set of read arcs,
  - $F_{IA} \subseteq P \times T$ is the set of inhibitory arcs,
  - $F_{EA} \subseteq P \times T$ is the set of equal arcs, and
  - $F_{XA} \subseteq P \times T$ is the set of reset arcs.
- f: $F \to \mathbb{R}_0^+$ is a function that assigns a non-negative real value to each arc $f(x, y) \in F$, $x, y \in P \cup T$.
- v: $T \to H$ is a function that assigns a firing-rate function $h(t)$ to each transition $t \in T$, whereby $H := \bigcup_{t \in T} \{h(t) \mid h(t) : \mathbb{R}^{\bullet t} \to \mathbb{R}\}$ is the set of all stochastic hazard functions, and $v(t) = h(t)$ for all transitions T.
- m_0: $P \to \mathbb{R}_0^+$ gives the initial marking.

2.2.3 *Generalised Hybrid Petri Nets*

In the $\mathcal{HPN}$, see Figure 2.4, the aspects of $\mathcal{SPN}$ and $\mathcal{CPN}$ are joined. Nodes in a $\mathcal{HPN}$ can either be considered as part of an $\mathcal{SPN}$ or $\mathcal{CPN}$. There are strict rules on how to connect nodes of different origin, see Definition 2.10. The resulting state space is a combination of a discrete state space of an $\mathcal{SPN}$ and the continuous state space of an $\mathcal{CPN}$, where the CTMC is connected through Markov jumps. An $\mathcal{HPN}$ is defined as [130]:

Definition 2.10 (Generalised Hybrid Petri Net)**.** A hybrid Petri net is a 6-tuple
$\mathcal{N} = \{P, T, F, f, V, m_0\}$, where:

- P is a finite, non-empty set of places with $P = \{P_{cont} \cup P_{disc}\}$, whereby P_{cont} is the set of continuous places to which non-negative real values can be assigned and P_{disc} is the set of discrete places to which non-negative integer values can be assigned.
- T is a finite, non-empty set of transitions with $T = \{T_{cont} \cup T_{stoch}\}$, whereby T_{cont} is the set of continuous transitions, which fire continuously over time and P_{stoch} is the set of stochastic transitions, which fire stochastically with exponentially distributed waiting times.
- F is a finite set of directed arcs. F is defined as the union of five disjunctive arc sets, i.e. $F := F_{SA} \cup F_{RA} \cup F_{IA} \cup F_{EA} \cup F_{XA}$ with:
  - $F_{SA,cont} \subseteq (P_{cont} \times T) \cup (T \times P_{cont})$ is the set of standard arcs,
  - $F_{SA,disc} \subseteq (P \times T) \cup (T \times P)$ is the set of standard arcs,
  - $F_{RA} \subseteq P \times T$ is the set of read arcs,
  - $F_{IA} \subseteq P \times T$ is the set of inhibitory arcs,
  - $F_{EA} \subseteq P \times T$ is the set of equal arcs, and

- $F_{XA} \subseteq P \times T_{stoch}$ is the set of reset arcs.

- f: is a function that assigns the following values to each arc $f(x,y) \in F$, $x,y \in P \cup T$:
  - $F_{SA,cont} \rightarrow \mathbb{R}_0^+$;
  - $F_{SA,disc} \rightarrow \mathbb{N}_0$;
  - $F_{RA} \rightarrow \mathbb{R}^+$ if $P \in P_{cont}$ or $F_{RA} \rightarrow \mathbb{N}^+$ if $P \in P_{disc}$;
  - $F_{EA} \rightarrow \mathbb{R}_0^+$ if $P \in P_{cont}$ or $F_{EA} \rightarrow \mathbb{N}_0^+$ if $P \in P_{disc}$; and
  - $F_{IA} \rightarrow \mathbb{R}^+ \cup \{0^+\}$ if $P \in P_{cont}$ or $F_{IA} \rightarrow \mathbb{N}^+$ if $P \in P_{disc}$, where 0^+ means a very small positive real number but not zero.

- V is a set of functions with $V = \{v_{cont}, v_{stoch}\}$
  - v_{cont}: $T_{cont} \rightarrow H_{cont}$ is a function that assigns a firing-rate function $h_{cont}(t)$ to each transition $t \in T_{cont}$, whereby $H_{cont} := \bigcup_{t \in T_{cont}} \{h_{cont}(t) \mid h_{cont}(t) : \mathbb{R}^{\bullet t} \rightarrow \mathbb{R}\}$ is the set of all stochastic hazard functions, and $v_{cont}(t) = h_{cont}(t)$ for all transitions T_{cont}.
  - v_{stoch}: $T_{stoch} \rightarrow H_{stoch}$ is a function that assigns a stochastic hazard function $h_{stoch}(t)$ to each transition $t \in T_{stoch}$, whereby $H_{stoch} := \bigcup_{t \in T_{stoch}} \{h_{stoch}(t) \mid h_{stoch}(t) : \mathbb{N}_0^{\bullet t} \rightarrow \mathbb{R}^+\}$ is the set of all stochastic hazard functions, and $v_{stoch}(t) = h_{stoch}(t)$ for all transitions T_{stoch}.

- $m_0 = m_{0,cont} \cup m_{0,stoch}$: gives the initial marking for continuous places P_{cont} and discrete places P_{disc}, whereby $m_{0,cont} : P_{cont} \rightarrow \mathbb{R}_0^+$ and $m_{0,cont} : P_{cont} \rightarrow \mathbb{N}_0$

2.3 *Coloured Petri Nets*

Coloured Petri nets [11, 12, 13] are the coloured extension of the standard Petri net classes. Like the hierarchical structuring using coarse nodes, the colouring concept facilitates a compact model design. Coloured Petri nets extend the strength of standard Petri nets with the expressiveness of a programming language allowing the definition of data types. In addition to the elements of a standard Petri net, a coloured Petri net model is characterised by colour sets, each defined by a datatype. As common for programming languages, a data type is a set of values obeying certain properties, e.g. integer numbers, Boolean values or strings [15].

As explained in [121]: "In a coloured Petri net, a colour set is assigned to each place. A place in a coloured Petri net may host distinguishable coloured tokens, whose colours are comprised in the colour set assigned to the place. Each arc carries an arc-expression. Its result type is a multiset over the colour sets of the connected places, see [122] for an explanation on multisets. Thus, variables associated with a transition originate either from the guard of the transition or the arc-expressions of in- or out-going arcs. An arc-expression can only be evaluated if values with suitable data types are assigned to each variable. This process is called binding [70]. A particular binding of a transition defines a transition instance. Each transition is equipped with a guard, which is a Boolean expression over defined

variables. The guard determines whether a transition instance for a particular binding exists or not. If the guard is not specified in any other way, it is set to the default value "*true*", which is not explicitly given. If the guard of a transition is evaluated to false, the transition can not be enabled. Both the guard and the arc-expression of the in-going arcs of a transition for a particular binding must be evaluated to true to enable the corresponding transition instance. Only enabled transition instances may fire. The firing of an enabled transition instance removes coloured tokens from the pre-places and adds coloured tokens to the post-places according to the specified arc-expressions."

For each of the four standard Petri net classes, see previous Section and Figure 2.4 exists a coloured Petri net equivalent, see Figure 2.4. A coloured qualitative Petri net ($\mathcal{QPN}^{\mathcal{C}}$) is defined as:

Definition 2.11 (Coloured Qualitative Petri Net)**.** A hybrid Petri net is a 9-tuple $\mathcal{N}(G_{col}) = \{P^{G_{col}}, T^{G_{col}}, F^{G_{col}}, W^{G_{col}}, \psi^{G_{col}}, g^{G_{col}}, f^{G_{col}}, m_0^{G_{col}}\}$, where:

- P is a finite, non-empty set of places.
- T is a finite, non-empty set of transitions.
- F is a finite set of directed arcs. F is defined as the union of five disjunctive arc sets, i.e. $F := F_{SA} \cup F_{RA} \cup F_{IA} \cup F_{EA} \cup F_{XA}$ with:
  - $F_{SA,cont} \subseteq (P_{cont} \times T) \cup (T \times P_{cont})$ is the set of standard arcs,
  - $F_{SA,disc} \subseteq (P \times T) \cup (T \times P)$ is the set of standard arcs,
  - $F_{RA} \subseteq P \times T$ is the set of read arcs,
  - $F_{IA} \subseteq P \times T$ is the set of inhibitory arcs,
  - $F_{EA} \subseteq P \times T$ is the set of equal arcs, and
  - $F_{XA} \subseteq P \times T_{stoch}$ is the set of reset arcs.
- W is a finite, non-empty set of colour sets.
- ψ: $P \to W$ is a colour function that assigns to each place $p \in P$ a colour set $\psi(p) \in W$.
- g: $T \to EXP$ is a guard function that assigns to each transition $t \in T$ a guard expression of Boolean type.
- f: $F \to EXP$ is an arc function that assigns to each arc $f(x, y) \in F$, $x, y \in P \cup T$ an arc-expression of a multiset type $\psi(p)_{MS}$.
- m_0: $P \to EXP$ is an initialization function that assigns to each place $p \in P$ an initialization expression of a multiset $\psi(p)_{MS}$.

The definitions of the coloured Petri net equivalents for the remaining three Petri net classes $\mathcal{SPN}$, $\mathcal{CPN}$ and $\mathcal{HPN}$ have to be defined accordingly. For the sake of simplicity, they are not explicitly given, see [121] and [122] for more details.

Colour sets can be defined using different data types, depending on the data type, we distinguish simple and compound colour sets:

- Simple types:
  - *int* (non-negative integers), e.g. colour set $CS_1 := 1, 2, 3$
  - *string* (sequences of printable ASCII characters), e.g. colour set $CS_2 := a, b, c$

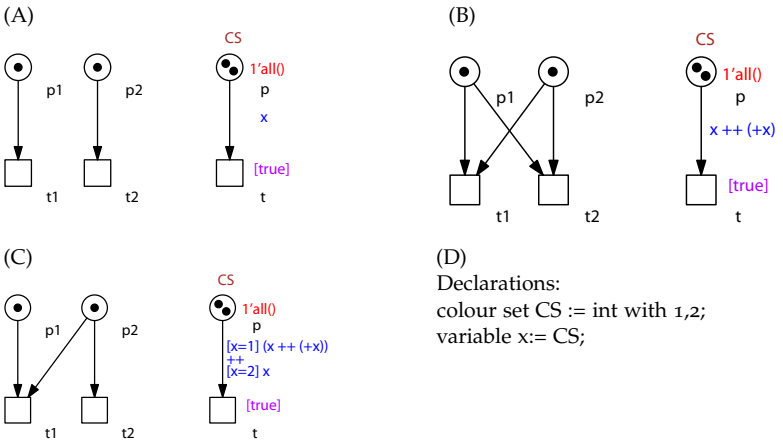


Figure 2.5: Coloured Petri net basics. For each of the three cases (A)-(C), a colour set "CS" has been defined with two colours of type integer: "1" and "2" and a variable x bound to the colour set "CS", see (D). The colour "1" represents the subnet with $p1$ and $t1$, whereas colour "2" represents the subnet with $p2$ and $t2$. In cases (A)-(C) the left net shows the standard Petri net and the right net illustrates the coloured Petri net equivalent. The additional notations of the coloured Petri net are given as follows: arc-expressions in blue, markings in red, colour sets in maroon, and guards in purple. (A) The net to be folded consists of two identical isolated subnets, resulting into the arc-expression x. (B) The net to be folded has two extra arcs ($p1$ to $t2$; $p2$ to $t1$), resulting into the arc-expression $x++(+x)$. The operator "+" in $(+x)$ returns the successor of x in an ordered finite colour set. The operator "++" is the multiset addition operator. (C) The net to be folded has one extra arc ($p2$ to $t1$), resulting into the arc-expression $[x = 1](x++(+x))++[x = 2]x$. The expression in square brackets $[\ldots]$ defines a predicate (if-statement) followed by an instruction, see text for more details. (adapted from [121])

  - *bool* (Boolean values are true and false), e.g. colour set $CS_3 := true, false$
  - ...
- Compound types:
  - *product* (tuple of previously declared colour sets)

$$\text{e.g. } CS_{1,2}^{product} := CS_1 \times CS_2 \equiv \begin{matrix}(1,a),(1,b),(1,c),\\(2,a),(2,b),(2,c),\\(3,a),(3,b),(3,c)\end{matrix}$$

  - *union* (disjoint union of previously declared colour sets)

$$\text{e.g. } CS_{1,2}^{union} := CS_1, CS_2 \equiv 1,2,3,a,b,c$$

Coloured and standard Petri nets can be converted into each other by folding and unfolding. The unfolding of the coloured Petri net model, as well as the computation of its state space, allows for the application of all analysis techniques of the standard Petri net classes [70, 91].

Figure 2.5 demonstrates the notions and use of coloured Petri nets on a set of small examples, see also text below for more details about the examples.

Example. Figure 2.5 shows three small examples of standard Petri nets and their coloured equivalents. For each case (A)-(C) the net on the left shows the standard Petri net and the net on the right the coloured Petri net equivalent [1]. Each of the standard Petri nets consists of two places $p1$ and $p2$, as well as two transitions $t1$ and $t2$. The examples only differ in the arcs connecting the places and transitions. For the coloured Petri net equivalents, we defined a colour set "CS" of type *int* (see above) with two colours "1" and "2" and a variable x, which is bound to colour set "CS", see Figure 2.5(D). The graph of each coloured Petri nets consists only of a coloured place p bound to the colour set "CS" (maroon notation), a coloured transition t with its guard set to "*true*" (purple notation) and an arc connecting

Note: In contrast to standard Petri nets, the graph of a coloured Petri net [1] has additional annotations with the following colour code:

- arc-expressions (blue)
- guards (purple)
- markings (red)
- colour sets (maroon)

the coloured place and transition. The differences in the connection between places and transitions in the corresponding standard Petri nets are realised through arc-expressions (blue notations). Thus, colour "1" represents the subnet with $p1$ and $t1$, whereas colour "2" represents the subnet with $p2$ and $t2$. Since the places $p1$ and $p2$ are each marked with one token, the marking of the coloured place p is set to $1'all()$ (red notation). The symbol "$'$" in the marking expression separates coefficient (number of tokens) and colour. The function $all()$ returns all colours from the colour set bound to the corresponding place. In this case, colours "1" and "2" are returned. In total, the coloured place p has one token of colour "1" and another token of colour "2". The tokens or number directly displayed on a coloured place always indicates the total number of tokens over all place instances; here two tokens are shown. The arc-expressions for the coloured Petri nets are derived as follows:

- Figure 2.5(A), the net to be folded consists of two identical isolated subnets, resulting into the arc-expression x. Depending on the colour that is bound to variable x, either one token of colour "1" (representing the firing of transition $t1$) or "2" (representing the firing of transition $t2$) must be subtracted from the coloured place p.
- Figure 2.5(B), the net to be folded has two extra arcs ($p1$ to $t2$; $p2$ to $t1$) compared to Figure 2.5(A), resulting into the arc-expression $x++(+x)$. The operator "$+$" in $(+x)$ returns the successor of x in an ordered finite colour set. If x is bound to colour "1", the expression $(+x)$ is evaluated to colour "2", otherwise if x is bound to colour "2", it returns colour "1". The operator "$++$" is the multiset addition operator, see [122] for more explanations on multisets. Here, the multiset addition operator creates a set of differently coloured tokens, which run along the corresponding arc. Thus, if x is bound to colour "1", the arc-expression $x++(+x)$ claims that one token of colour "1" and "2" must be subtracted from the coloured place p (representing the enabling of transition $t1$). Vice versa, if x is bound to colour "2", the arc-expression $x++(+x)$ claims that one token of colour "2" and "1" must be subtracted from the coloured place p (representing the enabling of transition $t2$).
- Figure 2.5(C), the net to be folded has one extra arc ($p2$ to $t1$) compared Figure 2.5(A), resulting into the arc-expression $[x = 1](x++(+x))++[x = 2]x$. The expression in square brackets $[\ldots]$ defines a predicate. A predicate is a Boolean expression, which subtracts a subset of colours from the actual colour set. The predicate $[x = 1]$ subtracts colour "1" form the colour set CS, where the predicate $[x = 2]$ subtracts colour "2". An arc-expression follows each predicate. If the predicate is evaluated to true for a particular binding, then the corresponding arc-expression is applied. The notation of a predicate followed by an arc-expression is similar to an if-statement followed by an instruction. Here, the multiset addition operator "$++$" allows to list a number of different predicates with their corresponding arc-expression. Thus,

if x is bound to colour "1", the notation $(x + +(+x))$ is valid. Accordingly, a token of colour "1" and "2" must be subtracted from the coloured place p (representing the enabling of transition $t1$), see also Figure 2.5(B). Otherwise, if x is bound to colour "2", the notation x is valid. Accordingly, a token of colour "2" must be subtracted from the coloured place p (representing the enabling of transition $t2$), see also Figure 2.5(A).

2.4 *Tools*

As already mentioned in the introduction, Petri nets are supported by a wealth of computer tools. Most popular Petri net tools in the context of biological applications are Snoopy [120] with its close friends Charlie [141] and Marcie [129], GreatSPN [144], CPN Tools [43], Cell Illustrator [102], and the two analysis tools INA [28] and TINA [57]. A comparison of these tools can be found in [120, 129, 141].

In this thesis, we used the toolkit consisting of Snoopy [120], Charlie [141] and Marcie [129] for their outstanding facilities in modelling and analysing biochemical networks, which are summarised below:

Snoopy [120] offers a unifying Petri net framework constituted of a coloured and an uncoloured level of Petri net classes. The uncoloured level comprises qualitative (time-free) Place/Transition Petri nets ($\mathcal{QPN}$) as well as quantitative (time-dependent) Petri nets such as stochastic Petri nets ($\mathcal{SPN}$), continuous Petri nets ($\mathcal{CPN}$), and generalised hybrid Petri nets ($\mathcal{GHPN}$). The coloured level provides coloured counterparts of the uncoloured level, and comprises coloured qualitative Petri nets ($\mathcal{QPN}^{\mathcal{C}}$), coloured stochastic Petri nets ($\mathcal{SPN}^{\mathcal{C}}$), coloured continuous Petri nets ($\mathcal{CPN}^{\mathcal{C}}$) and coloured generalised hybrid Petri nets ($\mathcal{GHPN}^{\mathcal{C}}$). Petri nets of all Petri net classes within one level can be converted into each other. Meaning the resulting Petri nets share the same structure and are only distinct in their kinetic information. Through user-guided folding, a coloured Petri net can be obtained from an uncoloured Petri net. Coloured Petri nets can be automatically unfolded to uncoloured Petri nets. Again, changing the level only alters the presentation style but not the actual net structure of the Petri net. Snoopy supports the simultaneous use of different Petri net classes. The user-interface adapts automatically to the current Petri net class. Depending on the Petri net class built-in animations and simulations can be performed to investigate the behaviour of models. By changing the level and net classes, a model can be simultaneously investigated by deploying different mathematical modelling paradigms in various complementary ways [82, 111, 122]. Snoopy supports the hierarchical structuring of Petri nets and logical (fusion) nodes to ease the design and systematic construction of complex models, and thus, to master large networks. It also supports regulatory arcs shown in Figure 2.2, which increase the expressiveness of Petri nets. Snoopy supports the export to several analysis tools, among them are Charlie [141] and Marcie [129] (see Snoopy's website

www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy for a full list). Snoopy also provides other export formats e.g. to capture and document the Petri net graph, simulation results and extract the ODE from an $\mathcal{CPN}$. In particular, the human-readable text file formats ANDL (Abstract Net Description Language) [120] and CANDL (Coloured Abstract Net Description Language) [121] are an input format for the analysis tools Charlie [141] and Marcie [129]. Snoopy also allows the export and import of the standard format SBML (Systems Biology Markup Language), which makes it even more versatile for modelling and analysing biochemical models. Snoopy's rich modelling and simulation capabilities yield a very powerful and exceptional tool for Petri nets, particularly for the application in systems and synthetic biology, see [120] for a detailed review. Comparable tools to Snoopy are GreatSPN [144], CPN Tools [43], and Cell Illustrator [102], which all to some extent have the same capabilities, but none of them support as many net classes and features as Snoopy does, see [120] for a more detailed comparison.

Charlie [141] is a place/transition net analyser, which supports the analysis of standard Petri nets and extended Petri nets with (weighted) regulatory arcs (see Figure 2.2). Thus, Charlie supports the import of the four uncoloured Petri net classes ($\mathcal{QPN}$, $\mathcal{SPN}$, $\mathcal{CPN}$, $\mathcal{HPN}$) in the Snoopy-file format, as well as in the human-readable text file format ANDL (Abstract Net Description Language) [120], the file format APNN (Abstract Petri Net Notation) [24], and INA's PNT files (plain text file) [28]. Firing rates defined in the quantitative Petri net classes are ignored, only the net structure including regulatory arcs and the initial marking are considered during the analysis. The basic functionality of Charlie supports:

- the structural analysis of standard graph properties;
- the incidence matrix based analysis to e.g. compute p- and t-invariants and check for structural boundedness;
- the siphon/trap based analysis to compute siphons and traps and to check for the siphon-trap property (see [99] for more details about siphons and traps);
- the reachability/coverability graph construction to analysis the analytically constructed state space of the model, in the case of a bounded Petri net, the reachability graph is constructed, otherwise the coverability graph (see [82] for more details about reachability/coverability graphs);
- model checking of the computed state space based on Linear Time Logic (LTL) [14] or Computation Tree Logic [16]; and
- the search for a path in the computed state space.

Additional analysers can be added through Charlie's plug-in system, see [141] for more details. Thus, the analysis results provided by Charlie is valuable to verify and validate models, such as biochemical networks modelled by e.g. Snoopy [120]. The design of Charlie has been build on the experiences on the Integrated Net Analyser

(INA) [28], other than that there exist no comparable analysis tool, see [141] for more details.

Marcie [129] is a tool for the analysis of qualitative Petri nets (QPN) extended with read and inhibitor arcs and stochastic Petri nets $\mathcal{SPN}$, respectively generalised stochastic Petri nets $\mathcal{GSPN}$ with immediate transitions (no waiting time). $\mathcal{SPN}$ and $\mathcal{GSPN}$ can be augmented by rewards. Rewards specify additional measures for a probabilistic model. The models can be submitted through Marcie in the human-readable text file format ANDL (Abstract Net Description Language) [120] and the XML-based PNML (Petri Net Markup Language) format [90]. Marcie supports the analysis of qualitative and quantitative standard properties, as well as model checking based on established temporal logics, like Computation Tree Logic (CTL) [16], Continuous Stochastic Logic (CSL) [26], Continuous Stochastic Reward Logic (CSRL) [58] and Probabilistic Linear-Time Temporal Logic (PLTLc) [77]. The analysis engines for bounded Petri net models are based on Interval Decision Diagrams. They are complemented by simulative and approximative engines to allow for quantitative reasoning on unbounded models. Marcie features allow individualising the analysis workflow in various ways by defining and checking specific properties of a system based on temporal logics, which is necessary for an in-depth analysis of models in the context of systems and synthetic biology. The multi-threaded implementation of Marcie is a huge benefit for the quantitative model analysis, which is of particular interest in the case of complex networks, such as biochemical systems. Related tools are Prism [60], MRMC [112], Smart [35], and Möbius [86], which again only partly support the analysis capabilities of Marcie, see [129] for a more detailed comparison.

In summary, the tool-kit of Snoopy [120], Charlie [141] and Marcie [129] is perfectly geared together and especially powerful for the modelling and analysis of biochemical networks. All three tools have been downloaded several times and successfully used for teaching and research, particularly in the field of technical and biochemical systems [120, 129, 141].

3
BioModelKit (BMK) Framework

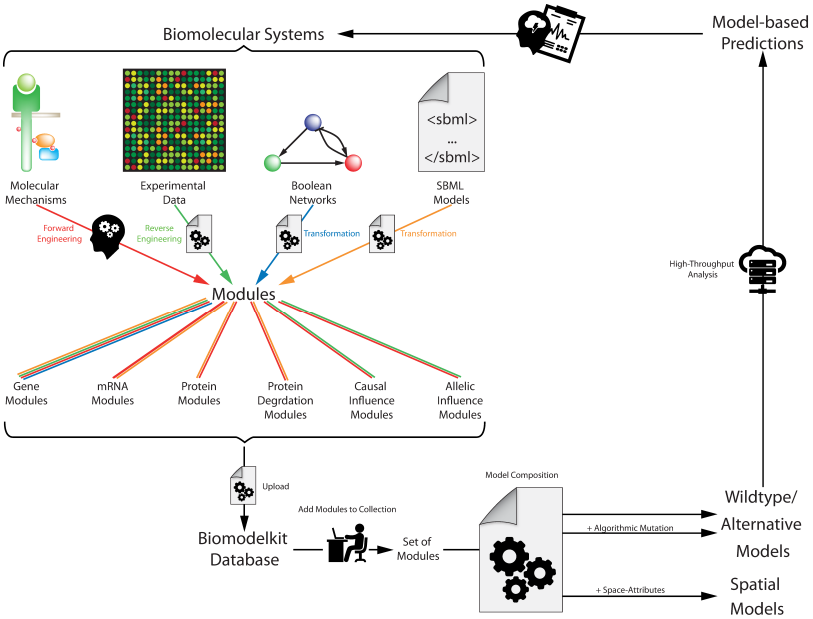


Figure 3.1: The BMK framework is a modular modelling framework for biomodel engineering based on Petri nets. Modules represent individual molecular components and their interactions. The module type depends on the biological classification of the represented component, process, and available mechanistic information. Depending on the module type and available resources forward or reverse engineering can be used to construct modules. Modules can also be generated by transforming existing models into modules. Modules are stored in a web-accessible database. Through a web-interface modules can be accessed and integrated into modularly composed models. During the composition, model mutation algorithms or a spatial transformation algorithm can be applied. By submitting the composed models to an analysis workflow, extensive *in silico* studies can be performed. (adapted from [139])

[1] BMKFR - BioModelKit framework

The BMKFR [1] is a tool for modular biomodel engineering [126], see also Figure 3.1. The main motivation behind the BMKFR was to develop a modelling framework, where modules are specifically designed for the purpose of model composition. The modularisation approach defined in the BMKFR was inspired by the natural composition of biomolecular processes, where molecular components with genetic information (genes, mRNAs, and proteins) are the natural building blocks. Thus, each of those molecular components is represented as a self-contained module, describing the functionality of the molecular components by using the formal language of Petri nets. We defined mechanistic module types according to the various types of genetic components. Causal module types were developed to capture also processes with missing mechanistic description [116, 124]. Interface networks, which are part of each module describe the interactions with other molecular components and are fundamental to automatically couple modules of interacting molecular components. The formalisation of the modularization concept of the BMKFR is explained in Section 3.1.

A module consists not only of its underlying model in the form of a Petri net, but also of a model annotation. The annotation is defined in the BMKml format [2], see Section 3.2, and documents the elements of the module and references considered during the construction process. The annotation of a module is vital to understand the modelled molecular mechanisms and to execute the module in a simulation environment.

[2] BMKml - BioModelKit mark-up language

The modular model composition using interface networks is explained in Section 3.3. Here, we also introduce the concept of algorithmic model mutation to generate alternative model structures with biochemically meaningful mutations. The algorithmic model mutation allows mimicking gene knock-outs and structural mutations of genetic components. The set of alternative models generated through algorithmic model mutation can be employed to identify models with desired behaviour. Such approaches might help to increase the understanding of the modelled system and thus to predict the possible phenotypes of real world mutants. Another concept introduced in this section is the spatial model transformation. Modular composed models can be extended with spatial information, which allows (1) mapping components to the cellular structure; (2) describing their movement or translocation; and (3) integrating geometric properties of a cell, membranes, compartment, pools etc. [139].

In Section 3.4, we describe how modules can be generated by forward engineering and reverse engineering approaches, as well as by transforming existing models, e.g. Boolean models or SBML models. In particular, reverse engineering approaches and the algorithmic transformation of models into modules might considerably increase the module repository and thus, the universal applicability of the BMKfr.

Forestalling, the BMKfr is supported by a web-tool, which consists of the BMKdb[3] and the BMKwi [4] [126]. Both will be introduced in Chapter 4. The organization of modules in the BMKdb allows the versioning of modules. Different module versions with competing hypotheses on the molecular mechanism or with alternative granularities can be stored in the BMKdb. Through the BMKwi the user can interact with the content stored in the BMKdb, which includes (1) browsing, searching and inspecting modules; (2) composing modular models, which includes the algorithmic model mutation and spatial extension of models; and (3) submitting and curating modules.

[3] BMKdb - BioModelKit database
[4] BMKwi - BioModelKit web-interface, accessible at `www.biomodelkit.com`

Running Example. To illustrate our approach, we employ a small running example describing the ligand-controlled expression of a receptor, see Figure 3.2. The binding of the ligand to the receptor activates the catalytic domain of the receptor, which in turn phosphorylates the tyrosine residue at the receptor. The phosphotyrosine residue acts as a docking site for an adaptor protein. After docking of the adaptor protein to the phosphotyrosine residue at the receptor, the catalytic domain of the receptor phosphorylates the tyrosine residue of the adaptor protein. The phosphorylation of the adaptor

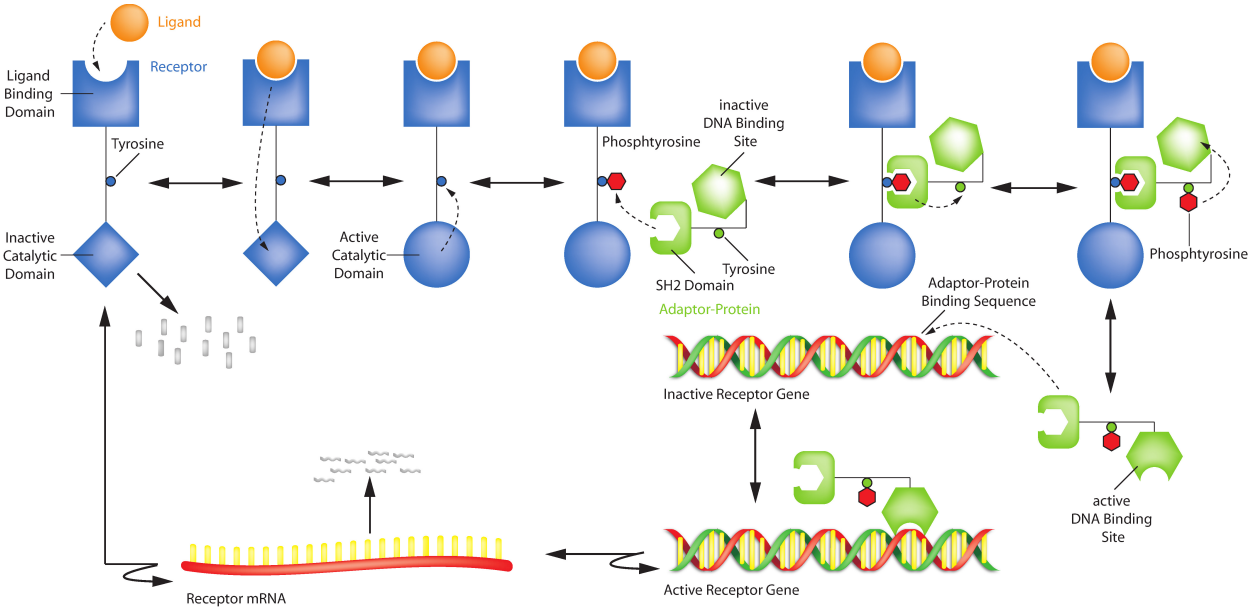


Figure 3.2: Molecular mechanism of the running example. The running example describes the regulated biosynthesis of a receptor protein (blue), which involves a ligand (orange) and an adaptor protein (green). The ligand bound to the receptor protein activates the catalytic domain. In turn, the tyrosine at the receptor protein gets phosphorylated. The SH2 domain of the adaptor protein binds to the phosphotyrosine of the receptor protein, which results in the phosphorylation of the tyrosine of the adaptor protein. The phosphotyrosine of the adaptor protein activates the DNA binding domain, which interacts with the receptor-encoding gene and initiates the transcription and translation of the receptor protein. Both the receptor protein and mRNA can be degraded.

protein exposes its DNA binding domain. Thus, the adaptor protein can bind to the regulatory region of the receptor gene and start the transcription of the receptor-encoding mRNA. The receptor mRNA is translated into the receptor protein. Both, the receptor protein and the mRNA can be degraded.

3.1 BMK - Module Definition

The formalisation of the modularisation concept is the firmament of the BMKfr. Only based on a consistent formal mathematical representation it is possible:

... to discriminate different module types;
... to define structural properties for the validation of the module structure;
... to define a suitable MIRIAM-compliant [54] annotation format for modules;
... to algorithmically process modules to generate modularly composed models;
... to algorithmically mutate composed models;
... to transform composed models into spatial models;
... to define a suitable database scheme for the BMKfr holding all information about a module including its annotations; and
... to develop a web-interface to make modules and the modular model composition, including the model mutation and spatial transformation of composed models accessible to the systems biology community.

In Section 3.1.1, we provide a formal definition of a module. Next, we introduce the different types of modules in the BMKfr, see Section 3.1.2. In Section 3.1.3, we define the relation between the introduced module terminology and the Petri net formalism. Furthermore, we specify interface subnetworks in the Petri net representation of a module to couple modules. Then, we demonstrate the relation between the module definition and standard graph properties of a Petri net, see Section 3.1.4.

3.1.1 Module Definition and Terminology

Below we introduce the formalization of the modularization concept and necessary terminology.

Component

A biomolecular component, in short component, is any chemical component that contributes to a biological process at a molecular level.

Genetic Component

Components with genetic information like genes, mRNAs, and proteins, are called genetic components.

Non-genetic Component

Components without genetic information like second-messengers, energy equivalents, ions, etc., are called non-genetic components.

Module

In our definition a MODULE M describes the functionality of a particular genetic component c_0, called MAIN COMPONENT, including its interactions with n_{IC} other components, $n_{IC} \geq 0$. The total set of components of a module M is given by $C(M) = \{c_0, \ldots, c_{n_{IC}}\}$.

Example: The module for the receptor protein in Figure 3.2 includes the receptor protein as the main component. Here, the ligand and the adaptor protein are interacting components.

Functional Unit

A FUNCTIONAL UNIT u is a defined part of a component $c \in C(M)$ with an assigned function that is of importance for the functionality of the component [5]. A genetic component $c \in C(M)$ consists of n_{FU} functional units, $n_{FU} \geq 1$. We assume that non-genetic components cannot be decomposed into more than one functional unit. Therefore, a non-genetic component is defined by only one functional unit. The total set of functional units of a component $c \in C(M)$ is given by $U(c) = \{u_1, \ldots, u_{n_{FU}}\}$. The summation of all sets $U(c)$ over the set of components $C(M)$ gives the total set of functional units in module M, $\mathcal{U}(M) = \bigcup_{c \in C(M)} U(c)$.

[5] E.g. a structural domain, a protein fold, a structural motif, a secondary structure, an amino acid sequence motif or a single amino acid of a protein, or a sequence motif, a codon, or a single base of the DNA/RNA (see Table 3.1)

Example: The receptor protein in Figure 3.2 consists of three functional units: ligand binding domain, tyrosine residue, and catalytic domain.

Molecular State

A MOLECULAR STATE s describes a specific physical constitution of a functional unit $u \in \mathcal{U}(M)$. A functional unit $u \in U(c)$ can adopt n_{MS} different molecular states, $n_{MS} \geq 1$. [6] The total set of molecular states of a functional unit $u \in U(c)$ is given by $S(u) = \{s_1, \ldots, s_{n_{MS}}\}$. The summation of all sets $S(u)$ over $u \in \mathcal{U}(M)$ defines the set $\mathcal{S}(M) = \bigcup_{u \in \mathcal{U}(M)} S(u)$.

[6] E.g. conformational state of a protein domain, covalent modification or non-covalent association of a component etc.

Example: The tyrosine residue of the receptor protein in Figure 3.2 can exist in three different molecular states: unphosphorylated, phosporylated, or phosphorylated and bound to the SH2 domain of the adaptor protein.

Each molecular state $s \in \mathcal{S}(M)$ can be mapped to a single functional unit or to a subset of functional units according to the relation $\lambda^u(s) = \{u \in \mathcal{U}(M) \mid s \in S(u)\}$. Accordingly, the mapping of a molecular state $s \in \mathcal{S}(M)$ to components is defined by the relation $\lambda^c(s) = \{c \in C(M) \mid s \in \bigcup_{U(c)} S(u)\}$. A molecular state $s \in \mathcal{S}(M)$ is called INTERACTION STATE s_{IS} if it can be mapped to more than one component $|\lambda^c(s)| > 1$ defining a complex $k = \lambda^c(s)$. The total set of interaction states is defined as $\mathcal{S}_{IS}(M) = \{s \in \mathcal{S}(M) \mid |\lambda^c(s)| > 1\}$, $\mathcal{S}_{IS}(M) \subseteq \mathcal{S}(M)$.

Example: The molecular state of the phosphorylated tyrosine residue bound to the SH2 domain of the adaptor protein, see receptor protein

Table 3.1: Examples of functional units. (A) Functional units of proteins, (B) functional units of genes (DNA) and mRNA.

Functional Unit	Description
(A) Protein	
structural domain	self-stabilizing domain as independently folding element of a protein, e.g. src-homology domain, pleckstrin homology domain, phosphotyrosine domain, etc.
protein fold	general protein architectures, e.g. helix bundle, beta barrel, Rosman fold, etc.
structural motif	short segment of the three-dimensional structure, e.g. β-hairpin, α-helix-hairpin, greek key, ω-loop, helix-loop-helix, zinc-finger, helix-turn-helix, etc.
secondary structures	defined structural elements, e.g. α-helix, π-helix, 3_{10}-helix, β-strain, β-turn, random-coil, etc.
amino acid sequence stretch	short segments of the amino acid sequence, e.g. N-glycosylation site motif, IQ motif, cytokine binding motif, phosphorylation motif, etc.
amino acid	single amino acid residue, e.g. tyrosine, serine, threonine, etc.
(B) Gene (DNA), mRNA	, etc.
DNA/RNA sequence motif	short segment of the DNA/RNA sequence, e.g. promotor, operator, enhance region, TATA-Box, etc.
DNA/RNA codon	three DNA/RNA bases encoding a specific amino acid
DNA/RNA base	single DNA/RNA base e.g. adenin, cytosin, guanin, thymin (DNA)/uracil(RNA), etc.

in Figure 3.2, represents an interaction state, which can be mapped to the receptor protein and the adaptor protein.

Each functional unit $u \in \mathcal{U}(M)$ can only exist in one specific molecular state at each time point, this molecular state is called ACTIVE MOLECULAR STATE, given by the relation $\alpha : s \rightarrow [0,1]$. If $\alpha(s) = 1$ the molecular state s is active, otherwise if $\alpha(s) = 0$ the molecular state s is inactive. In addition, the relation $\sum_{s \in S(u)} \alpha(s) = 1$ has to be true.

Example: If the ligand-binding domain of the receptor protein in Figure 3.2 is currently not occupied by a ligand, the molecular state of the unbound ligand binding domain is the active molecular state.

To initialise a module, we need to specify an initially active molecular state for each functional unit $u \in U(c_0)$ of the main component c_0 called HOME STATE $s_{0,u} \in S(u)$ with $\alpha(s_{0,u}) = 1$. We define that a home state $s_{0,u} \in S(u)$ is not allowed to be an interaction state, such that $s_{0,u} \notin \mathcal{S}_{IS}(M)$. This assumption is necessary to exclude inconsistencies during the modular model composition. The union of all home states is given by $S_0(M) = \bigcup_{u \in U(c_0)} s_{0,u}$.

Example: We assume that the receptor protein in Figure 3.2 is in its initial state catalytically inactive, not phosphorylated and not interacting with any other component. Thus, the unbound state of the ligand binding domain, the unphosphorylated state of the tyrosine residue and the inactive state of the catalytic domain have to be declared as home states. According to the assumption above, both the ligand-bound state of the ligand binding domain and the phosphorylated state of the tyrosine interacting with the adaptor protein are not allowed to be home states of the corresponding functional units.

Molecular Event

The relations between molecular states in $\mathcal{S}(M)$ are described by n_{ME} molecular events, $n_{ME} \geq 1$. A MOLECULAR EVENT e represents an

action changing the molecular states of the functional units. [7] The total set of molecular events is given by $\mathcal{E}(M) = \{e_1, \ldots, e_{n_{ME}}\}$.

[7] E.g. a bio-chemical reaction, a covalent or non-covalent binding reaction, a conformational change, a transport process etc.

Example: The transition from the unbound state to the ligand-bound state of the ligand-binding domain of the receptor protein in Figure 3.2 happens through the molecular event of ligand binding.

The stoichiometry with which a molecular state $s \in \mathcal{S}(M)$ mediates a molecular event $e \in \mathcal{E}(M)$ is given by the STOICHIOMETRIC COEFFICIENT $\nu(e,s) \in \mathbb{N}_0$ (educts) and $\nu(s,e) \in \mathbb{N}_0$ (products). The set of molecular states participating at a molecular event e can be distinguished into a set of educts, given by $\bullet e = \{s \in \mathcal{S}(M) \mid \nu(s,e) \neq 0\}$, and a set products, given by $e\bullet = \{s \in \mathcal{S}(M) \mid \nu(e,s) \neq 0\}$.

Example: The molecular event of ligand binding in Figure 3.2 involves three molecular states: the unbound ligand binding domain, the ligand-bound ligand binding domain and the ligand. The molecular states of the unbound ligand binding domain and the ligand are the educts of the molecular event. The molecular state of the ligand-bound ligand binding domain is the product of the molecular event. The Stoichiometric coefficients of all attending molecular states are set to "1".

Vice versa, $\bullet s = \{e \in \mathcal{E}(M) \mid \nu(e,s) \neq 0\}$ defines the set of molecular events, where the molecular state s is a product, and $s\bullet = \{e \in \mathcal{E}(M) \mid \nu(s,e) \neq 0\}$ defines the set of molecular events, where molecular state s is an educt.

Example: The molecular state of the unbound ligand binding domain of the receptor protein in Figure 3.2 is a product of the molecular event of ligand dissociation and educt of the molecular event of ligand binding.

A molecular event $e \in \mathcal{E}(M)$ can be defined as $e = \{(\varepsilon(\bullet e), \varepsilon(e\bullet)), \varepsilon(\bullet e) \rightarrow \varepsilon(e\bullet)\}$, where $\varepsilon(\bullet e) = \sum_{\bullet e} \nu(s,e) \cdot s$ and $\varepsilon(e\bullet) = \sum_{e\bullet} \nu(e,s) \cdot s$. Each molecular event $e \in \mathcal{E}(M)$ occurs with a rate $r(e)$.

A molecular event e is called INTERACTION EVENT e_{IS} if it involves more than one component, $|\lambda^c(\bullet e \cup e\bullet)| > 1$. The total set of interaction events is given by $\mathcal{E}_{IA}(M) = \{e : |\lambda^c(\bullet e \cup e\bullet)| > 1\}$.

Example: The molecular events of ligand binding and dissociation in Figure 3.2, as well as the binding and dissociation of the adaptor protein to the phosphorylated tyrosine are interaction events, which either involve the receptor protein and ligand or the receptor protein and the adaptor protein.

Molecular Process

A MOLECULAR PROCESS $E(u)$ defines the set of molecular events changing the molecular state of a functional unit $u \in U(c)$, such that $E(u) = \bigcup_{S(u)}((\bullet s \cup s\bullet) \setminus (\bullet s \cap s\bullet))$.

Example: The molecular events of ligand binding and dissociation in Figure 3.2 define a molecular process.

Properties of Molecular Events

Depending on the type and reversibility of a molecular event, different properties can be defined. A molecular event $e \in \mathcal{E}(M)$ representing a degradation process might have an empty set of products $e\bullet \neq \emptyset$ if the products are not modelled. A set of molecular states $S(u)$ of a functional unit $u \in U(c)$ that is degraded by a molecular event $e \in \mathcal{E}(M)$ cannot share any molecular state $s \in S(u)$ with the product set of the molecular event e, such that $S(u) \cap e\bullet = \emptyset$.

Example: Assuming that the receptor protein in Figure 3.2 is degraded as a whole protein, all its functional units must be removed. After degradation, none of the functional units can be left. We neglect any by-products. The molecular states representing the functional units of the receptor protein can only be educts of the molecular event of degradation, and none of them can be a product.

In contrast, a molecular event $e \in \mathcal{E}(M)$ representing a synthesis process might have an empty set of educts $\bullet e \neq \emptyset$ if educts are not modelled. A set of molecular state $S(u)$ of a functional unit $u \in U(c)$ that is synthesised by a molecular event $e \in \mathcal{E}(M)$ cannot share any molecular state $s \in S(u)$ with the educt set of the molecular event e, such that $S(u) \cap \bullet e = \emptyset$.

Example: We neglect the amino acids necessary to compose a protein and assume that none of the functional units exists before the synthesis of the receptor protein. Thus, none of the molecular states of the receptor protein in Figure 3.2 can be an educt of the molecular event of receptor protein synthesis, they can only be products.

If the molecular event $e \in \mathcal{E}(M)$ does neither include degradation nor synthesis of a functional unit $u \in \mathcal{U}(M)$, it must have a non-empty set of products $e\bullet \neq \emptyset$ and educts $\bullet e \neq \emptyset$. In this case, each functional unit $u \in U(c)$ attending a molecular event $e \in \mathcal{E}(M)$ must share one molecular state $s \in S(u)$ with the set of products of the molecular event e, such that $|S(u) \cap e\bullet| = 1$, and one molecular state $s' \in S(u)$ with the set of educts of the molecular event e, such that $|S(u) \cap \bullet e| = 1$.

Example: The molecular event of activating the catalytic domain of the receptor protein in Figure 3.2 involves two functional units, the catalytic domain and the ligand binding domain. Both have one molecular state in the set of products (inactive catalytic domain, ligand-bound ligand binding domain) and one molecular state in the set of products (active catalytic domain, ligand-bound ligand binding domain) of the molecular event. If one of the molecular states were missing, the corresponding functional unit would be either degraded or synthesised.

If a molecular event $e \in \mathcal{E}(M)$ is reversible, there must exist another molecular event $e' \in \mathcal{E}(M)$, such that $e' = \{(\varepsilon(e\bullet), \varepsilon(\bullet e)), \varepsilon(e\bullet) \rightarrow \varepsilon(\bullet e)\}$.

Example: The molecular event representing the ligand binding to the receptor protein in Figure 3.2 (forward reaction) has a corresponding back reaction representing the dissociation of the ligand from the

receptor. The educt and product sets are transposed.

Furthermore, at least one functional unit of the main component $u \in U(c_0)$ has to attend at each molecular event $e \in \mathcal{E}(M)$, such that $\bigcup_{u \in U(c_0)} S(u) \subseteq e\bullet \cup \bullet e$.

Example: The receptor protein in Figure 3.2 (corresponding functional units are given in braces) attends at the molecular events for:

- binding and dissociation of the ligand (ligand binding domain),
- activation and inactivation of its catalytic domain (catalytic domain, ligand binding domain),
- phosphorylation and dephosphorylation of its tyrosine residue (tyrosine residue, catalytic domain),
- binding and dissociation of the adaptor protein (tyrosine residue), and
- the phosphorylation of the tyrosine residue of the adaptor protein (tyrosine residue, catalytic domain).

Other molecular events are not part of the receptor protein module.

3.1.2 *Module Types*

This section requires:

- *Module definition, Section 3.1.1*

According to the different types of genetic components and involved processes, we define four types of modules [116]:

1. A Gene Module M_{g,c_0} represents the transcriptional activity of a gene, which is controlled by the formation of the regulatory landscape and the pre-initiation complex. These processes include complex non-covalent interactions with transcriptions factors or other regulatory proteins interacting with silencer or enhancer sequences of the gene.

   ↪ Minimal requirements:
   - The module M_{g,c_0} must contain at least two molecular states representing the transcriptional active $s^{act}_{g,c_0} \in \mathcal{S}(M_{g,c_0})$ and inactive state $s^{inact}_{g,c_0} \in \mathcal{S}(M_{g,c_0})$ of the gene.

2. An mRNA Module M_{m,c_0} represents the biosynthesis of a particular mRNA by transcription of the respective gene; the post-transcriptional modification of the mRNA including capping, (alternative) splicing and polyadenylation; the translation into the proteins encoded by the processed mRNA; and the degradation of the mRNA including its potential control through proteins or small interfering anti-sense RNA molecules. These processes include covalent modification and non-covalent interactions of the mRNA.

   ↪ Minimal requirements:
   - According to its definition, the module M_{m,c_0} must contain at least one molecular state s^{mRNA}_{m,c_0} representing the mRNA, which is translated into a protein and three molecular events representing the synthesis of the mRNA, the degradation of the mRNA and the synthesis of the respective protein given by M_{p,c'_0}.

- For a molecular event $e \in \mathcal{E}(M_{m,c_0})$ representing the transcription of a gene defined by module M_{g,c'_0} it must be true:
  * $\mathcal{S}_0(M_{m,c_0}) \subseteq e\bullet$ - a home state $s \in \mathcal{S}_0(M_{m,c_0})$ of the main component c_0 defined by module M_{m,c_0} must be a product of the molecular event e.
  * $\mathcal{S}(M_{m,c_0}) \cap \bullet e = \emptyset$ - no molecular state $s \in \mathcal{S}(M_{m,c_0})$ of the main component c_0 defined by module M_{m,c_0} is an educt of the molecular event e.
  * $s^{act}_{g,c_0*} \in \bullet e, e\bullet$ - a molecular state s^{act}_{g,c_0*} representing a transcriptionally active state of the gene defined by module M_{g,c_0*} must be an educt and a product of the molecular event e.
- For a molecular event $e \in \mathcal{E}(M_{m,c_0})$ representing the translation into a protein defined by module M_{p,c'_0} it must be true:
  * $\mathcal{S}_0(M_{p,c'_0}) \subseteq e\bullet$ - a home state $s \in \mathcal{S}_0(M_{p,c'_0})$ of the main component c'_0 defined by module M_{p,c'_0} must be a product of the molecular event e.
  * $\mathcal{S}(M_{p,c'_0}) \cap \bullet e = \emptyset$ - no molecular state $s \in \mathcal{S}(M_{p,c'_0})$ of the main component c'_0 defined by module M_{p,c'_0} is an educt of the molecular event e.
  * $s^{mRNA}_{m,c_0} \in \bullet e, e\bullet$ - a molecular state s^{mRNA}_{m,c_0} representing the mRNA defined by module M_{m,c_0} must be an educt and a product of the molecular event e.

3. A Protein Module M_{p,c_0} describes the functionality of a particular protein (single poly-peptide chain), including changes of the protein conformation, non-covalent interactions with other components and covalent modifications that regulate the functionality. Thus, a protein module represents the formation and cleavage of covalent and non-covalent bonds, as well as conformational changes of the protein structure. Covalent modifications include:
   - Post translational modification, the addition of functional groups (most often using enzymes) during or after protein biosynthesis, e.g.:
     - hydrophobic groups for membrane localisation;
     - cofactors for enhanced enzymatic activity;
     - modifications of translation factors;
     - smaller chemical groups;
     - etc.
   - Covalent linkage of peptides, e.g.:
     - ISGylation, the covalent linkage to the ISG15 protein (Interferon-Stimulated Gene 15);
     - SUMOylation, the covalent linkage to the SUMO protein (Small Ubiquitin-related Modifier);
     - ubiquitination, the covalent linkage to the protein ubiquitin;
     - neddylation, the covalent linkage to Nedd;
     - pupylation, the covalent linkage to the Prokaryotic ubiquitin-like protein

- Chemical modification of amino acids, e.g.:
  - citrullination, or diminution, the conversion of arginine to citrulline;
  - deamination, the conversion of glutamine to glutamic acid or asparagine to aspartic acid;
  - eliminylation, the conversion to an alkene by beta-elimination of phosphothreonine and phosphoserine, or dehydration of threonine and serine, as well as by decarboxylation of cysteine;
  - carbamylation, the transfer of the carbamoyl from a carbamoyl-containing molecule (carbamoyl phosphate) to an acceptor moiety such as an amino group
- Structural changes, e.g.:
  - proteolytic cleavage, cleavage of a protein at a peptide bond;
  - racemization, conversion of an enantiomerically pure mixture (one where only one enantiomer is present) into a mixture where more than one of the enantiomers are present;
  - protein splicing, self-catalytic removal of intense analogous to mRNA processing

The protein module also represents non-covalent functional interactions, irrespective of the life-time of complexes formed with other components including:

- electrostatic interactions, the attraction of ions or molecules with full permanent charges of opposite signs;
- van der Waals forces, the attraction and repulsions between atoms, molecules, and surfaces, as well as other intermolecular forces, caused by correlations in the fluctuating polarisations of nearby particles in contrast to covalent and ionic bonding;
- π-effects, the non-covalent interaction of electron-rich π-system with a metal (cationic or neutral), an anion, another molecule and even another π-system (pivotal to protein-ligand recognition; and
- hydrophobic effects, the desire for non-polar molecules to aggregate in aqueous solutions by separation of water.

↪ Minimal requirements:

- A molecular event $e \in \mathcal{E}(M_{g,c_0'})$ representing the switch between the transcriptionally active $s^{act}_{g,c_0'} \in \mathcal{S}(M_{g,c_0'})$ and inactive state $s^{inact}_{g,c_0'} \in \mathcal{S}(M_{g,c_0'})$ of a gene, defined by module $M_{g,c_0'}$, and triggered by a molecular state $s \in \mathcal{S}(M_{p,c_0})$, where $s \in {\bullet e}$ and $s \in e^{\bullet}$, is not represented in the protein module M_{p,c_0}, $e \notin \mathcal{E}(M_{p,c_0})$. Meaning, the transition between the active and inactive state of a gene due to a common interaction state of the gene and the protein is not modelled in the protein module M_{p,c_0}.

4. A Protein Degradation Module M_{d,c_0} represents the degradation of a protein by proteolysis in lysosomes, ubiquitin-dependent

degradation by the proteasome, or degradation by digestive enzymes or by any other possible mechanism that may lead to the degradation or inactivation of the protein. The post-translational proteolytic processing is described in the corresponding protein module. Complex processes like the proteolytic processing can be summarised in a few steps if molecular events and states are defined accordingly.

↪ Minimal requirements:

- The module M_{d,c_0} must contain at least one molecular event e representing the degradation of the respective protein defined by M_{p,c_0}.
- The educts $\bullet e$ of a molecular event $e \in \mathcal{E}(M_{d,c_0})$ representing the degradation of the respective protein (or parts of the protein) given by M_{p,c_0} are limited to those molecular states of the respective main component c_0, which are no interaction states $\bullet e \cap \mathcal{S}_{IS}(M_{p,c_0}) = \emptyset$. Otherwise, functional units of interacting components would be degraded as well and disappear in the composed model, which might result in inconsistencies in particular if such an effect is unintended.

The introduction of gene modules and mRNA modules allows modeling regulated gene expression and protein biosynthesis. The gene expression pattern of a cell is not constant and can drastically change dependent on the cell type, physiological state, or experimental conditions. As a result, cells are equipped with specific sets of proteins of variable relative abundance. By introducing gene modules into the model, differentially regulated gene activity and the resulting gene expression patterns translate into the available quantities of mRNA and protein molecules. The rates of biochemical reactions always depend on both, the kinetic rate constants and the concentrations of the reactants. Thus, a change in the gene expression may effect the rates of biochemical reactions, which in turn may drastically alter the dynamic behaviour of a regulatory network. Moreover, altered concentrations of regulatory proteins (e.g. transcription factors) may feedback in a complex manner onto the gene level by changing the gene expression profiles. This circuitry of interwoven regulatory control becomes systematically accessible through the composed model [116].

The module types introduced above exclusively rely on known molecular mechanisms, even though mechanistic details may not necessarily be considered. To model possible influences on a molecular network through yet unknown molecular mechanisms or complex processes that cannot be represented in detail, we introduced two more module types [116]:

5. An Allelic Influence Module M_{ai,c_0} represents the effect of alleles (mutated versions of a gene) on molecular processes. In contrast to gene modules, the described effects are causal influences, which might be directly or indirectly mediated by unknown processes.

6. A Causal Influence Module M_{ci,c_0} describes the influence of arbitrary entities, others than alleles, on molecular processes.

The introduction of causal and allelic influence modules extends the modelling power of the BMKfr by disclaiming, in a formally correct manner, the need of known molecular mechanisms. Causal and allelic influence modules allow to include levels of abstraction required to cope with complex entities, mechanisms, or phenomena, while the corresponding model parts are restricted to accordingly defined modules. The formal framework facilitates the reverse engineering of biomodels from complex phenotype data sets resulting from genotypic variation e.g. by employing Petri net compatible algorithms [88, 108, 109]. It is obvious that such types of models have a high potential for the application to various areas from basic research to synthetic biology or personalized medicine [116].

Running Example. The running example in Figure 3.2 consists of four genetic components (receptor protein, receptor-encoding mRNA, receptor-encoding gene, adaptor protein) and one non-genetic component (ligand). Accordingly, the modularized system consists of two protein modules to represent the receptor M_{pR} and adaptor protein M_{pA}, one mRNA module for the receptor mRNA M_{mR}, one gene module for the receptor gene M_{gR} and one protein degradation module for the receptor protein M_{dR}.

The receptor protein consists of three functional units *(1)* ligand binding domain, which exists in two molecular states: free or bound to its ligand; *(2)* catalytic domain, which exists in two molecular states: inactive or active; *(3)* tyrosine residue, which exists in three molecular states: unphosphorylated, phosphorylated or phosphorylated and bound to the SH2 domain of the adaptor protein. The sequence of the receptor mRNA is not resolved by explicitly representing its nucleotide sequence and is thus be defined by only one functional unit with one molecular state. The receptor gene consists of two functional units: *(1)* coding sequence, which exists in two molecular states: inactive or active regarding being transcribed; *(2)* adaptor protein binding sequence, which exists in two molecular states: free or bound to the DNA binding site of the adaptor protein. The adaptor protein consists of three functional units *(1)* SH2 domain, which exists in two molecular states: free or bound to the phosphotyrosine of the receptor protein; *(2)* DNA binding domain, which exists in three molecular states: inactive, active and free or active and bound to the adaptor protein binding sequence of the receptor gene; *(3)* tyrosine residue, which exists in two molecular states: unphosphorylated or phosphorylated. The ligand as a non-genetic component is defined by only one functional unit, which exists in two molecular states: free or bound to the ligand binding domain of the receptor. In Table 3.2, we summarised the molecular structure of each module with its associated components, their functional units and molecular states. We also indicated the main component of each module, as well as the home states of the functional units and interaction states among the

Component	Functional Unit		Molecular State	HS	IS
(A) Receptor protein module M_{pR}					
receptor protein (pR)*	ligand binding domain (LBD)	$u^{M_{pR}}_{pR,1}$	free bound to ligand	x	x^1
	catalytic domain (CD)	$u^{M_{pR}}_{pR,2}$	inactive active	x	
	tyrosine (Y)	$u^{M_{pR}}_{pR,3}$	unphosphorylated phosphorylated phosphorylated, bound to SH2 of adaptor protein	x	x^2
adaptor protein (pA)	SH2 domain (SH2)	$u^{M_{pR}}_{pA,1} = u^{M_{pA}}_{pA,1}$	free bound to phosphorylated Y of receptor protein		x^2
	tyrosine (Y)	$u^{M_{pR}}_{pA,2} = u^{M_{pA}}_{pA,2}$	unphosphorylated phosphorylated		
ligand (L)	ligand (L)	$u^{M_{pR}}_{L,1}$	free bound to LBD of the receptor protein		x^1
(B) Adaptor protein module M_{pA}					
adaptor protein (pA)*	SH2 domain (SH2)	$u^{M_{pA}}_{pA,1}$	free bound to phosphorylated Y of receptor protein	x	x^2
	tyrosine (Y)	$u^{M_{pA}}_{pA,2}$	unphosphorylated phosphorylated	x	
	DNA binding site (DNA BS)	$u^{M_{pA}}_{pA,3}$	inactive active active, bound to the receptor gene	x	x^3
receptor protein (pR)	catalytic domain (CD)	$u^{M_{pA}}_{pR,1} \subset u^{M_{pR}}_{pR,2}$	active		
	tyrosine (Y)	$u^{M_{pA}}_{pR,2} \subset u^{M_{pR}}_{pR,3}$	phosphorylated phosphorylated, bound to SH2 of adaptor protein		x^2
receptor gene (gR)	adaptor protein binding sequence (ABS)	$u^{M_{pA}}_{gR,1} = u^{M_{gR}}_{gR,1}$	free bound to the DNA BS of the adaptor protein		x^3
(C) Receptor gene module M_{gR}					
receptor gene (gR)*	adaptor protein binding sequence (ABS)	$u^{M_{gR}}_{gR,1}$	free bound to the DNA BS of the adaptor protein	x	x^3
	DNA coding sequence of receptor protein	$u^{M_{gR}}_{gR,2}$	inactive active	x	
adaptor protein (pA)	DNA binding site (DNA BS)	$u^{M_{gR}}_{pA,1} = u^{M_{pA}}_{pA,3}$	active active, bound to the receptor gene		x^3
(D) Receptor mRNA module M_{mR}					
receptor mRNA (mR)*	mRNA coding sequence of receptor protein	$u^{M_{mR}}_{mR,1}$	(not further specified)		
receptor gene (gR)	DNA coding sequence of receptor protein	$u^{M_{mR}}_{gR,1} \subset u^{M_{gR}}_{gR,1}$	active		
receptor protein (pR)	ligand binding domain (LBD)	$u^{M_{mR}}_{pR,1} \subset u^{M_{pR}}_{pR,1}$	free		
	catalytic domain (CD)	$u^{M_{mR}}_{pR,2} \subset u^{M_{pR}}_{pR,2}$	inactive		
	tyrosine (Y)	$u^{M_{mR}}_{pR,3} \subset u^{M_{pR}}_{pR,3}$	unphosphorylated		
(E) Receptor degradation module M_{dR}					
receptor protein (pR)*	ligand binding domain (LBD)	$u^{M_{dR}}_{pR,1} \subset u^{M_{pR}}_{pR,1}$	free		
	catalytic domain (CD)	$u^{M_{dR}}_{pR,2} \subset u^{M_{pR}}_{pR,2}$	inactive active		
	tyrosine (Y)	$u^{M_{dR}}_{pR,3} \subset u^{M_{pR}}_{pR,3}$	unphosphorylated phosphorylated		

Table 3.2: Molecular structure of the modules in the running example. Each module is characterized by its components, their functional units and molecular states. A component indicated with a "*" is the main component of the module. Home states are given in the column "HS" and interaction states in column "IS", identical superscripts indicate identical interaction states.

modules.

All molecular events are assumed to be reversible, except degradation processes, transcription, and translation. The binding of the ligand to the ligand-binding domain of the receptor results in the activation of the catalytic domain of the receptor protein. The active catalytic domain of the receptor protein can phosphorylate the ty-

Molecular Event	M_{pR}	M_{pA}	M_{gR}	M_{mR}	M_{dR}
Binding of the ligand and the ligand binding domain of the receptor protein	x				
Unbinding of the ligand and the ligand binding domain of the receptor protein	x				
Activation of the catalytic domain of the receptor protein	x				
Inactivation of the catalytic domain of the receptor protein	x				
Phosphorylation of the tyrosine residue of the receptor protein by its active catalytic domain	x				
Dephosphorylation of the phosphotyrosine residue of the receptor protein	x				
Binding of the SH2 domain of the adaptor protein to the phosphotyrosine residue of the receptor protein	x	x			
Unbinding of the SH2 domain of the adaptor protein to the phosphotyrosine residue of the receptor protein	x	x			
Phosphorylation of the tyrosine residue of the adaptor protein by the active catalytic domain of the receptor	x	x			
Dephosphorylation of the phosphotyrosine residue of the receptor protein		x			
Activation of the DNA binding site of the adaptor protein by its phosphotyrosine residue		x			
Deactivation of the DNA binding site of the adaptor protein		x			
Binding of the DNA binding site of the adaptor protein and the adaptor protein binding sequence of the receptor gene		x	x		
Unbinding of the DNA binding site of the adaptor protein and the adaptor protein binding sequence of the receptor gene		x	x		
Basal activation of the receptor gene			x		
Basal inactivation of the receptor gene			x		
Adaptor-Protein dependent activation of the receptor gene			x		
Adaptor-Protein dependent inactivation of the receptor gene			x		
Synthesis of the receptor mRNA induced by the active receptor gene				x	
Degradation of the receptor mRNA				x	
Synthesis of the receptor protein				x	
Degradation of the receptor protein					x

Table 3.3: Molecular events of the modules in the running example. The molecular events of the running example are listed in association with their appearance in the modules. (M_{pR} - receptor protein module; M_{pA} - adaptor protein module; M_{gR} - receptor gene module; M_{mR} - receptor mRNA module; M_{dA} - receptor protein degradation module)

rosine residue of the receptor protein. The phosphotyrosine of the receptor protein acts as a binding site for the SH2 domain of the adaptor protein. The active catalytic domain of the receptor protein phosphorylates the tyrosine of the bound adaptor protein. If the tyrosine of the adaptor protein is phosphorylated, its DNA binding domain gets activated and binds to the adaptor protein binding sequence of the receptor gene. The occupied adaptor protein binding sequence activates the coding sequence of the receptor gene. The spontaneous activation of the receptor gene yields in a low basal level of activity. If the receptor gene is active, the transcription of the mRNA starts, which is translated into the receptor protein. Here, we do not consider the mRNA processing and the regulation of its stability. Both the receptor protein and mRNA are degraded. For all molecular events, we assume mass-action kinetics. The molecular events and their assignments to the above-defined modules are summarised in Table 3.3.

3.1.3 *Module Transformation in Petri Nets*

The formal description of a module M can be translated into the structural components of a Petri net $\mathcal{N} = \{P, T, F, f, v, m_0\}$. Molec-

This section requires:

- *Petri nets [82], Section 2.1*
- *Module definition, Section 3.1.1*
- *Module types, Section 3.1.2*

ular states are represented by places. A place $p \in P$ is mapped to a molecular state $s \in \mathcal{S}(M)$ according to the relation $\varrho_{p\to s} : p \to s$. Molecular events are encoded by transitions. A transition $t \in T$ is mapped to a molecular event $e \in \mathcal{E}(M)$ according to the relation $\varrho_{t\to e} : t \to e$.

The Petri net graph $\mathcal{N}(M_{c_0}) = \{P, T, F, f, v, m_0\}$ of a module M_{c_0} is defined as:

- Set of places $P = \bigcup_{C(M_{c_0})} P^c$,
  - where $P^c = \bigcup_{U(c)} P^u$ is the set of places representing a component $c \in C(M_{c_0})$ and
  - $P^u = \{p : \varrho_{p\to s}(p) \in S(u)\}$ is the set of places representing a functional unit $u \in \mathcal{U}(M_{c_0})$
- Set of transitions $T = \bigcup_{C(M_{c_0})} T^c$, where
  - $T^c = \bigcup_{U(c)} T^u$ is the set of transitions related to a component $c \in C(M_{c_0})$ and
  - $T^u = \{t : \varrho_{t\to e}(t) \in E(u)\}$ is the set of transitions related to a functional unit $u \in \mathcal{U}(M_{c_0})$
- Set of arcs $F := F_{SA} \subseteq (P \times T) \cup (T \times P)$
- Arc-weights f: $F \to \mathbb{N}_0$, where
  - $\forall s \in \mathcal{S}(M_{c_0}), e \in \mathcal{E}(M_{c_0})$ with $s \in {}^\bullet e$: $f_{SA}(\varrho^{-1}_{p\to s}(s), \varrho^{-1}_{t\to e}(e)) = \nu(s, e)$ (input arc)
  - $\forall s \in \mathcal{S}(M_{c_0}), e \in \mathcal{E}(M_{c_0})$ with $s \in e^\bullet$: $f_{SA}(\varrho^{-1}_{t\to e}(e), \varrho^{-1}_{p\to s}(s)) = \nu(e, s)$ (output arc)
- Set of firing rates $v : T \to H$ with $H = \bigcup_{t\in T}\{h(t)$, where:
  - $\forall e \in \mathcal{E}(M_{c_0}) : h(\varrho^{-1}_{t\to e}(e)) = r(e)$
- Initial Marking m_0: $P \to \mathbb{N}_0$:
  1. Case: M_{c_0} is a gene, protein, causal or allelic influence module:
     - $\forall s \in S_0(M_{c_0}) : m_0(\varrho_{p\to s}(p)) = n_{c_0}$
     - $\forall s \in \mathcal{S}(M_{c_0}) \setminus S_0(M_{c_0}) : m_0(\varrho_{p\to s}(p)) = 0$
  2. Case: M_{c_0} is a protein degradation or mRNA module:
     - $\forall s \in \mathcal{S}(M_{c_0}) : m_0(\varrho_{p\to s}(p)) = 0$

A special case are protein degradation module M_{d,c_0} concerning the use of arcs. Assuming a transition $t \in T$ representing the molecular event of protein degradation $e = \varrho_{t\to e}(t)$ of the protein defined by M_{p,c'_0}. The set of places $P^{nonIS}_{M_{p,c'_0}} = \varrho^{-1}_{p\to s}(\bigcup_{U(c'_0)} S(u) \setminus \mathcal{S}_{IS}(M_{p,c'_0}) \subseteq)$ representing non-interaction states are connected with transition t using reset arcs $f_{XA}(P^{nonIS}_{M_{p,c'_0}}, t) = 1$, and marking-dependent standard arcs $f_{SA}(t, P^{nonIS}_{M_{p,c'_0}}) = m(P^{nonIS}_{M_{p,c'_0}})$. The set of places $P^{IS}_{M_{p,c'_0}} = \varrho^{-1}_{p\to s}(\bigcup_{U(c'_0)} S(u) \cap \mathcal{S}_{IS}(M_{p,c'_0}))$ representing interaction states are connected with transition t using inhibitory arcs $f_{IA}(P^{IS}_{M_{p,c'_0}}, t) = 1$. This transformation ensures that proteins are only degraded if they are not interacting with other components and that only one copy of the protein is degraded.

Interface Networks

A set of related modules $I = \{M_1, \ldots, M_n\}$ share an interface network, defined by $\mathcal{N}_I(M) = \{P^I, T^I, F^I, f^I, v^I, m_0^I(M)\}$, where $\mathcal{N}_I(M) \subseteq \mathcal{N}(M)$, $M \in I$. The interface networks are crucial for the modular composition of models, see Section 3.3.1. All nodes in an interface network $\mathcal{N}_I(M)$ are declared as logical (fusion) nodes. In the case of an interface network, the set of transitions T^I can be empty, but not the set of places P^I.

The sets of nodes P^I, T^I shared through the interface network $\mathcal{N}_I(M)$ by a set of modules I depend on the correlation among the modules in I. Here, we have to distinguish four cases:

1. Interface network describing the genetic correlation among components:
   - mRNA synthesis: To connect a gene module M_{g,c_0} with its related mRNA module $M_{m,c_0'}$, $I = \{M_{g,c_0}, M_{m,c_0'}\}$, the mRNA module reuses places representing molecular states that indicate the transcriptionally active state $s^{act}_{g,c_0'} \in \mathcal{S}(M_{g,c_0})$ with $s^{act}_{g,c_0'} = s^{act}_{g,c_0}$ of the main component c_0 of the gene module M_{g,c_0}. Thus, the set of transitions T^I is empty. The set of places is determined by $P^I = \{p \in P \mid \varrho_{p\to s}(p) = s^{act}_{g,c_0'}\}$. The resulting interface network $\mathcal{N}_I(M)$ must be an instance in the gene and the mRNA module, $\mathcal{N}_I(M) \subseteq \mathcal{N}(M_{g,c_0}), \mathcal{N}(M_{m,c_0})$.
   - Protein synthesis: To connect a mRNA module M_{m,c_0} with its related protein module $M_{p,c_0'}$, $I = \{M_{m,c_0}, M_{p,c_0'}\}$, the mRNA module reuses all places of the protein module representing home states of its main components $c_{0'}$. Thus, the set of transitions T^I is also empty, and the set of places determined by $P^I = \{p \in P_{M_{p,c_0'}} \mid \varrho_{p\to s}(p) \in \bigcup_{U(c_{0'})} s_{0,u}\}$. The resulting interface network $\mathcal{N}_I(M)$ must be an instance in the mRNA and the protein module, $\mathcal{N}_I(M) \subseteq \mathcal{N}(M_{m,c_0}), \mathcal{N}(M_{p,c_0'})$.
   - Protein degradation: To connect a protein module M_{p,c_0} with its related protein degradation module $M_{d,c_0'}$, $I = \{M_{p,c_0}, M_{d,c_0'}\}$ the protein degradation module reuses all places of the protein module. The set of places is determined by $P^I = P_{M_{p,c_0}}$. The set of transitions T^I is empty. The resulting interface network $\mathcal{N}_I(M)$ must be instances of the protein and protein degradation module, $\mathcal{N}_I(M) \subseteq \mathcal{N}(M_{p,c_0}), \mathcal{N}(M_{d,c_0'})$.
2. Interface networks describing (non-)covalent interaction among a particular set of genetic components, which might also involve non-genetic components:
   - Protein-protein interactions (protein to protein modules),
   - Protein-gene interactions (protein to gene modules),
   - Protein-mRNA interaction (protein to mRNA modules).

   The resulting interface network $\mathcal{N}_I(M)$ is defined by the set of interacting components $K = K_g \cup K_{ng}$, where the subset $K_g = \{c^g_{1,0}, \ldots, c^g_{n,0}\}$ holds the set of genetic components and the subset $K_{ng} = \{c^{ng}_1, \ldots, c^{ng}_{n'}\}$ holds the of non-genetic components,

$I = \{M_{c^g_{1,0}}, \ldots, M_{c^g_{n,0}}\}$. The set of transitions T^I has to contain all transitions that involve the components in K, $T^I = \{t \in T : \lambda^c(\bullet\varrho_{t\to e}(t) \cup \bullet\varrho_{t\to e}(t)) = K\}$ and the set of places P^I has to contain all places connected to those transitions in T^I, $P^I = \{p \in P : p \in \bullet t \cup t\bullet \mid t \in T^I\}$. The interface network $\mathcal{N}_I(M)$ must be an instance in all modules given by I, $\mathcal{N}_I(M) \subseteq \mathcal{N}(M_{c_1}), \ldots, \mathcal{N}(M_{c_n})$.

3. Interface networks describing interaction among modules through shared non-genetic components:
   The resulting interface network $\mathcal{N}_I(M)$ is defined by the set of indirectly interacting genetic components K_g with $K_g = \{c^g_{1,0}, \ldots, c^g_{n,0}\}$, $I = \{M_{c^g_{1,0}}, \ldots, M_{c^g_{n,0}}\}$. The main component $c^g_{i,0} \in K_g$ of each module $M \in I$ is interacting with a non-genetic component c^{ng} through distinct molecular events. Thus, the set of transitions T^I is empty, and the set of places is determined by $P^I = \{p \in P : \lambda^c(\varrho_{p\to s}(p)) = c^{ng}\}$.
4. Interface networks of causal/allelic influence modules:
   The interface network $\mathcal{N}_I(M)$ of a causal/allelic influence module M and any other module M', $I = \{M, M'\}$, is not strictly defined. To connect a causal/allelic influence module M to another module M', either the sets of places $P^M \cap P^{M'} := P^I$ and/or the sets of transitions $T^M \cap T^{M'} := T^I$ have to share a subset of nodes, such that $\mathcal{N}_I(M') \subseteq \mathcal{N}(M')$ and $\mathcal{N}_I(M'') \subseteq \mathcal{N}(M'')$.

So far, we only defined the set of places P^I and transitions T^I of an interface network $\mathcal{N}_I(M) = \{P^I, T^I, F^I, f^I, v^I, m^I_0(M)\}$ shared by the modules in $I = \{M_1 \ldots M_n\}$ in module $M \in I$. Below, we define arcs, firing rates and the markings of an interface network $\mathcal{N}_I(M)$:

- Set of firing rates $v^I : T^I \to H^I$ with $H^I = \bigcup_{t\in T^I}\{h^I(t)$, where $h^I(t) = h^{M_1}(t) = \ldots = h^{M_n}(t)$
- Set of arcs $F^I = ((P^I \times T^I) \cup (P^I \times T^I))$, where $F^I \subseteq F^{M_1}, \ldots, F^{M_n}$
- Arc-weights f: $F \to \mathbb{N}_0$, where $\forall x, y \in P^I \cup T^I : f^I(x, y) = f^{M_1}(x, y) = \ldots = f^{M_n}(x, y)$
- Initial marking m^I_0: $P \to \mathbb{N}_0$, where $\forall p \in P^I : m^I_0(p, M) = m^M_0(p)$

The redundancy introduced by the interface networks might appear unnecessarily complicated, but for modules with complex interaction sites, the approach offers tremendous benefits in securing the correct functioning of modules in a composed model, see Section 3.3.1. Even more, it ensures that in the composed model, a module can only execute a particular interaction if the modules of the involved genetic components are included in the model. Like in the real world system, components can only interact if they are available.

Running Example. Using the Tables 3.3 and 3.2 allows to construct the Petri net graphs, see Figure 3.3, for each module specified in Section 3.1.1. The relation between molecular states and places, respectively molecular events and transitions is given in Tables 3.4 and 3.5. Places representing home states of the respective main components are initially marked, see Table 3.2.

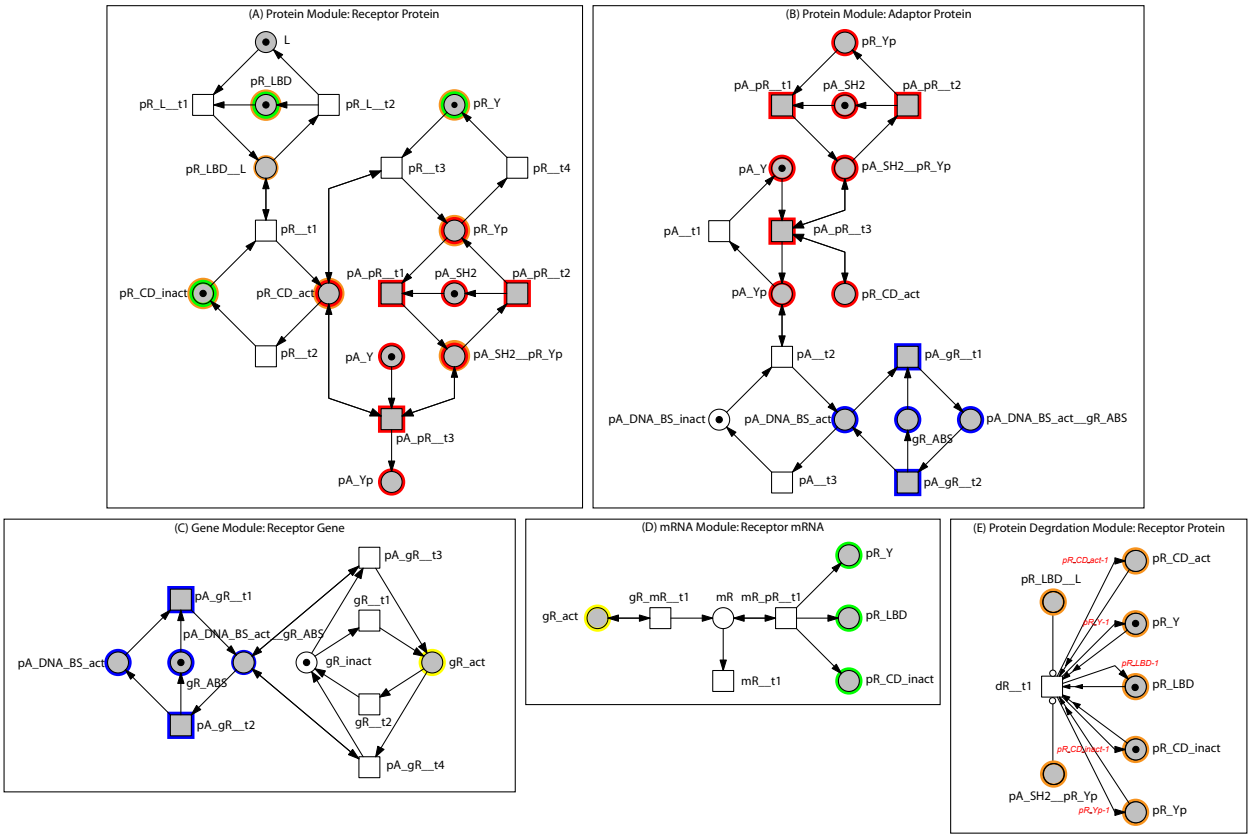


Figure 3.3: Petri net modules of the running example. According to the genetic components of the molecular network in Figure 3.2 two protein modules for the receptor and adaptor protein, one mRNA module for the receptor mRNA, one gene module for the receptor gene and one protein degradation module for the receptor protein have been constructed. The interface network shared by the modules of the involved components are indicated by logical nodes with coloured lines: receptor and adaptor protein (red), receptor gene and adaptor protein (blue), receptor gene and mRNA (yellow), receptor protein and mRNA (green), receptor protein and degradation (orange). (See text for more details.)

Each module depicted in Figure 3.3 is equipped with interface networks to connect to its related modules in the composed model. Nodes that are part of an interface network are defined as logical nodes and are visually distinguishable by their grey shading. The interface network $\mathcal{N}_I(M)$ of:

- Receptor gene and mRNA module, $I = \{M_{gR}, M_{mR}\}$, is given by (yellow nodes, see Figure 3.3):
  - $P^I = \{gR\_act\}$ (transcriptional active state of the receptor gene),
  - $T^I = \emptyset$
- Receptor mRNA and protein module, $I = \{M_{mR}, M_{pR}\}$, is given by (green nodes, see Figure 3.3):
  - $P^I = \{pR\_LBD, pR\_CD\_inact, pR\_Y\}$ (home states of the receptor protein)
  - $T^I = \emptyset$
- Receptor protein and protein degradation module, $I = \{M_{pR}, M_{dR}\}$, is given by (orange nodes, see Figure 3.3):
  - $P^I = \{pR\_LBD, pR\_CD\_act, pR\_CD\_inact, pR\_Y, pR\_Yp\}$ (non-interaction states of the receptor protein)
  - $T^I = \emptyset$
- Receptor protein and adaptor protein module, $I = \{M_{pR}, M_{pA}\}$, is given by (red nodes, see Figure 3.3):
  - $P^I = \{pR\_CD\_act, pR\_Yp, pA\_SH2, pA\_SH2\_\_pR\_Yp, pA\_Y, pA\_Yp\}$
  - $T^I = \{pA\_pR\_\_t1, pA\_pR\_\_t2, pA\_pR\_\_t3\}$

- Receptor gene and adaptor protein module, $I = \{M_{gR}, M_{pA}\}$, is given by (blue nodes, see Figure 3.3):
  - $P^I = \{gR\_ABS, pA\_DNA\_BS\_act\_\_gR\_ABS, pA\_DNA\_BS\_act\}$
  - $T^I = \{pA\_gR\_\_t1, pA\_gR\_\_t2, pA\_pR\_\_t3\}$

Table 3.4: Mapping of places and molecular states of the running example. Molecular states that have been marked as identical interaction states, see Table 3.2, are represented by logical places with identical names.

Functional Unit	Molecular State	Place Name
(A) Receptor protein		
ligand binding domain (LBD)	free	pR_LBD
	bound to ligand	$pR\_LBD\_\_L$
catalytic domain (CD)	inactive	pR_CD_inact
	active	pR_CD_act
tyrosine (Y)	unphosphorylated	pR_Y
	phosphorylated	pR_Yp
	phosphorylated, bound to SH2 of adaptor protein	$pA\_SH2\_\_pR\_Yp$
(B) Adaptor protein		
SH2 domain (SH2)	free	pA_SH2
	bound to phosphorylated Y of receptor protein	$pA\_SH2\_\_pR\_Yp$
tyrosine (Y)	unphosphorylated	pA_Y
	phosphorylated	pR_Yp
DNA binding domain (DNA BS)	inactive	$pA\_DNA\_BS\_inact$
	active	$pA\_DNA\_BS\_act$
	active, bound to the receptor gene	$pA\_DNA\_BS\_act\_\_gR\_ABS$
(C) Receptor gene		
adaptor protein binding sequence (ABS)	free	gR_ABS
	bound to the DNA BS of the adaptor protein	$pA\_DNA\_BS\_act\_\_gR\_ABS$
DNA coding sequence of receptor protein	inactive	gR_inact
	active	gR_act
(D) Receptor mRNA		
mRNA coding sequence of receptor protein	(not further specified)	mR

Table 3.5: Mapping of transitions and molecular events of the running example.

Molecular Event	Transition Name
Binding of the ligand and the ligand binding domain of the receptor protein	$pR\_L\_\_t1$
Unbinding of the ligand and the ligand binding domain of the receptor protein	$pR\_L\_\_t2$
Activation of the catalytic domain of the receptor protein	$pR\_L\_\_t1$
Inactivation of the catalytic domain of the receptor protein	$pR\_L\_\_t2$
Phosphorylation of the tyrosine residue of the receptor protein by its active catalytic domain	$pR\_\_t3$
Dephosphorylation of the phosphotyrosine residue of the receptor protein	$pR\_\_t4$
Binding of the SH2 domain of the adaptor protein to the phosphotyrosine residue of the receptor protein	$pA\_pR\_\_t1$
Unbinding of the SH2 domain of the adaptor protein to the phosphotyrosine residue of the receptor protein	$pA\_pR\_\_2t1$
Phosphorylation of the tyrosine residue of the adaptor protein by the active catalytic domain of the receptor	$pA\_pR\_\_t3$
Dephosphorylation of the phosphotyrosine residue of the receptor protein	$pA\_\_t1$
Activation of the DNA binding site of the adaptor protein by its phosphotyrosine residue	$pA\_\_t2$
Deactivation of the DNA binding site of the adaptor protein	$pA\_\_t3$
Binding of the DNA binding site of the adaptor protein and the adaptor protein binding sequence of the receptor gene	$pA\_gR\_\_t1$
Unbinding of the DNA binding site of the adaptor protein and the adaptor protein binding sequence of the receptor gene	$pA\_gR\_\_t2$
Basal activation of the receptor gene	$gR\_\_t1$
Basal inactivation of the receptor gene	$gR\_\_t2$
Adaptor-protein-dependent activation of the receptor gene	$pA\_gR\_\_t3$
Adaptor-protein-dependent inactivation of the receptor gene	$pA\_gR\_\_t4$
Synthesis of the receptor mRNA induced by the active receptor gene	mR_pR_t1
Degradation of the receptor mRNA	mR_t1
Synthesis of the receptor protein	mR_pR_t1
Degradation of the receptor protein	dR_t1

3.1.4 *Relation Between Graph Properties Emerging from the Module Definition*

In Section 2.1.1, we introduced graph properties that we are now using to characterise the Petri net graph of a module $\mathcal{N}(M)$. Due to the congruence between the module definition and the Petri net terminology, the properties of a molecular event and molecular state directly relate to graph properties. We assume conservation of mass for all molecular events, expect those representing the synthesis or degradation of a component.

The Petri net graph of a module $\mathcal{N}(M)$ must be *connected* and has:

- ...*input transitions* if there exits a molecular event $e \in \mathcal{E}(M)$ involving the synthesis of a component $c \in C(M)$, where the set of educts is empty $\bullet e = \emptyset$, such that $\bullet\varrho^{-1}_{t\to e}(e) = \emptyset$.
- ...*output transitions* if there exits a molecular event $e \in \mathcal{E}(M)$ including the degradation of a component $c \in C(M)$, where the set of products is empty $e\bullet = \emptyset$, such that $\varrho^{-1}_{t\to e}(e)\bullet = \emptyset$.
- ...*input places* if there exists a molecular state $s \in \mathcal{S}(M)$ that is not a product of any molecular event $\bullet s = \emptyset$, such that $\bullet\varrho^{-1}_{p\to s}(s) = \emptyset$.
- ...*output places* if there exists a molecular state $s \in \mathcal{S}(M)$ that is not an educt of any molecular event $s\bullet = \emptyset$, such that $\varrho^{-1}_{p\to s}(s)\bullet = \emptyset$.

The Petri net graph of a module $\mathcal{N}(M)$ cannot be:

- ...*pure*, if there exists a molecular state $s \in \mathcal{S}(M)$ that is product and educt of a molecular event $e \in \mathcal{E}(M)$, $s \in e\bullet$ and $s \in \bullet e$, such that $f(\varrho^{-1}_{t\to e}(e), \varrho^{-1}_{p\to s}(s)) \neq 0$ and $f(\varrho^{-1}_{p\to s}(s), \varrho^{-1}_{t\to e}(e)) \neq 0$.
- ...*ordinary* if there exists a molecular event $e \in \mathcal{E}(M)$ and a molecular state $s \in \mathcal{S}(M)$ with a stoichiometric coefficient greater one, $\nu(e,s) > 1$ (respectively $\nu(s,e) > 1$), such that $f(\varrho^{-1}_{p\to s}(s), \varrho^{-1}_{t\to e}(e)) > 1$ and/or $f(\varrho^{-1}_{t\to e}(e), \varrho^{-1}_{p\to s}(s)) > 1$.
- ...*homogeneous* if there exists a molecular state $s \in \mathcal{S}(M)$ that is educt of at least two molecular events $e, e' \in \mathcal{E}(M)$, $s \in \bullet e$ and $s \in \bullet e'$, with different stoichiometric coefficients $\nu(s,e) \neq \nu_{in}(s,e')$, such that:
  $$0 < max\{f(\varrho^{-1}_{t\to e}(e), \varrho^{-1}_{p\to s}(s)) \mid \varrho^{-1}_{t\to e}(e) \in (\varrho^{-1}_{p\to s}(s)\bullet\}.$$
- ...*non-blocking multiplicities free* if there exists a molecular state $s \in \mathcal{S}(M)$ that is not an educt of any molecular event $e \in \mathcal{E}(M)$, $\bullet s = \emptyset$, such that $\bullet\varrho^{-1}_{p\to s}(s) = \emptyset$. The property can also not be fulfilled if there exist a molecular state $s \in \mathcal{S}(M)$, where the minimum of the stoichiometric coefficients with the molecular state s being a product $min\{\nu(e,s) \mid \forall e \in \bullet s\}$ is smaller than the maximum of the stoichiometric coefficients with the molecular state s being an educt $max\{\nu(s,e) \mid \forall e \in s\bullet\}$, such that:
  $$min\{f(\varrho^{-1}_{t\to e}(e), \varrho^{-1}_{p\to s}(s)) \mid \forall\varrho^{-1}_{t\to e}(e) \in \bullet(\varrho^{-1}_{p\to s}(s)\} < max\{f(\varrho^{-1}_{p\to s}(s), \varrho^{-1}_{t\to e}(e)) \mid \forall\varrho^{-1}_{t\to e}(e) \in (\varrho^{-1}_{p\to s}(s)\bullet\}$$

... *conservative*, if there exists a molecular event $e \in \mathcal{E}(M)$ representing a covalent or non-covalent binding or unbinding, where the number of molecular states on the product and educt site multiplied by their stoichiometric coefficients are not equal, $\varepsilon(\bullet e) \neq \varepsilon(e\bullet)$, such that:

$$\sum_{\varrho_{p\to s}^{-1}(s)\in\varrho_{t\to e}^{-1}(e)\bullet} f(\varrho_{t\to e}^{-1}(e), \varrho_{p\to s}^{-1}(s)) = \sum_{\varrho_{p\to s}^{-1}(s)\in\bullet\varrho_{t\to e}^{-1}(e)} f(\varrho_{p\to s}^{-1}(s), \varrho_{t\to e}^{-1}(e))$$

... *static conflict free* if there exists a molecular state $s \in \mathcal{S}(M)$ that is an educt of at least two molecular events $e, e' \in \mathcal{E}(M)$, $s \in \bullet e$ and $s \in \bullet e'$, such that $\bullet\varrho_{t\to e}^{-1}(e) \cup \bullet\varrho_{t\to e}^{-1}(e') \neq \emptyset$.

... *strongly connected* and *reversible* if there exists an irreversible molecular event $e \in \mathcal{E}(M)$, such that no molecular event $e' \notin \mathcal{E}(M)$ exits, where $e' = \{(\varepsilon(e\bullet), \varepsilon(\bullet e)), \varepsilon(e\bullet) \to \varepsilon(\bullet e)\}$.

... *k-bounded* and *structurally bounded* if there exits a functional unit $u \in \mathcal{U}(M)$ attending at a molecular event $e \in \mathcal{E}(M)$, where the intersection of the set of molecular states $S(u)$ of the functional unit u with the product set of the molecular event $e\bullet$ results into a molecular state s, $S(u) \cap e\bullet = s$, but the intersection with the educt set is empty, $S(u) \cap \bullet e = \emptyset$, which happens in the case of the synthesis.

... *live* if there exists a molecular event $e \in \mathcal{E}(M)$ that represents an interaction of the main component c_0 and at least one other component $c \in C(M)$, which is the case for each molecular event $e \in \mathcal{E}_{IA}(M)$. A molecular event $e \in \mathcal{E}_{IA}(M)$ causes a dead transition, such that $\nexists m' \in [m\rangle : m'[\varrho_{t\to e}^{-1}(e)\rangle$.

... *reversible* if there exists a molecular state $s \in \mathcal{S}(M)$ that is not a product (educt) of any molecular event $e \in \mathcal{E}(M)$, such that $\exists m \in [m_0\rangle : m_0 \notin [m\rangle$.

Due to the assumed mass conservation, each functional unit $u \in U(M)$ represents a minimal P-invariant, $P^u \cdot \mathbb{C}(\mathcal{N}(M)) = 0$, if no molecular event $e \in \bigcup_{S(u)}(\bullet s \cup s\bullet)$ involves degradation $S(u) \cup e\bullet \neq \emptyset$ or synthesis $S(u) \cup \bullet e \neq \emptyset$. The Petri net graph of the module $\mathcal{N}(M)$ is covered by P-invariants, if this condition holds for all functional units in $U(M)$

The molecular process $E(u)$ of a functional unit $u \in U(M)$ corresponds to a T-invariant, not necessarily minimal, $\mathbb{C}(\mathcal{N}(M)) \cdot E(u) = 0$, if for each molecular state $s \in S(u)$ it is true that it is a product of at least one molecular action $e \in E(u)$, $s \in e\bullet$, and an educt of at least one molecular action $e' \in E(u)$, $s \in \bullet e'$. In other words, if the subnetwork induced by the molecular process $E(u)$ is strongly connected (closed), $E(u)$ defines a T-invariant. Otherwise, if the subnetwork induced by the molecular process $E(u)$ is not strongly connected (open), $E(u)$ defines no T-invariant. The Petri net graph of the module $\mathcal{N}(M)$ is covered by T-invariants, if this condition holds for all molecular processes in $E(u)$.

The defined relations between the module definition and Petri

net graph can be used for the validation of the constructed modules. Thus, modules have to adhere to a particular set of properties depending on their module type and involved molecular events. The structural validation of modules is necessary to exclude modelling faults and inconsistencies. Only successfully validated modules can be integrated into modularly composed models, see Section 3.3, to obtain a comprehensive and consistent model. False validated modules might introduce incorrect molecular mechanisms into the model and thus, interfere with the model behaviour.

Running Example. The Petri net graphs of all modules, see Figure 3.2, expect the receptor degradation module M_{dR} [8], have been subjected to the qualitative graph analysis. The results have been summarised in Table 3.6.

[8] Qualitative graph analysis of Petri nets with reset arcs and self-referencing standard arcs is not established in Charlie [141] and Marcie [129]

Table 3.6: Graph properties of the running example. The computed graph properties of the modules confirm the observations described in this section.

Property	M_{pR}	M_{pA}	M_{gR}	M_{mR}
Pure	N	N	N	N
Ordinary	Y	Y	Y	Y
Homogeneous	Y	Y	Y	Y
Non-blocking Multiplicity	N	Y	Y	Y
Conservative	N	N	N	N
Static Conflict Free	N	N	N	N
Input Transition	N	N	N	N
Output Transition	N	N	N	Y
Input Places	Y	N	N	N
Output Places	Y	N	N	Y
Connected	Y	Y	Y	Y
Strongly Connected	N	Y	Y	N
Covered by P-invariants	Y	Y	Y	N
Covered by T-invariants	N	Y	Y	N
Structurally Bounded	Y	Y	Y	N
k-Bounded	Y (k=1)	Y (k=1)	Y (k=1)	Y (k=0)
Dead States	1	1	0	1
Dead Transitions	Y	Y	Y	Y
Liveness	N	N	N	N
Reversibility	N	N	Y	N

Since all of the modules contain molecular states functioning as a product and educt at the same time, none of the modules is pure. In the modelled example, the stoichiometric coefficients of all molecular events are assumed to be one. Thus, all modules are ordinary and homogeneous. The tyrosine residue of the adaptor protein is an input place in the receptor protein module M_{pR}, which causes the module to have blocking multiplicities in contrast to the remaining modules. Binding and unbinding processes due to interaction among the components cause the modules to be not conservative. Also, all modules contain molecular states that are educts of more than one molecular event. Thus, all modules contain static conflicts. In the receptor protein module M_{pR}, the molecular state of the phosphotyrosine residue of the adaptor protein causes an output place. Places representing the home states of the receptor protein in the receptor mRNA module M_{mR} are also output places. None of the modules has input or output transitions. The output places of the receptor protein module M_{pR} and mRNA module M_{mR}, as well as the input place of

Module	P-invariants	T-invariants
M_{pR}	$P_{INV,1}(M_{pR}) = \{pR\_LBD, pR\_LBD\_\_L\}$ $\hookrightarrow = u_{pR,1}^{M_{pR}}$ $P_{INV,2}(M_{pR}) = \{pR\_CD\_act, pR\_CD\_inact\}$ $\hookrightarrow = u_{pR,2}^{M_{pR}}$ $P_{INV,3}(M_{pR}) = \{pR\_Y, pR\_Yp, pA\_SH2\_\_pR\_Yp\}$ $\hookrightarrow = u_{pR,3}^{M_{pR}}$ $P_{INV,4}(M_{pR}) = \{L, pR\_LBD\_\_L\}$ $\hookrightarrow = u_{L,1}^{M_{pR}}$ $P_{INV,5}(M_{pR}) = \{pA\_SH2, pA\_SH2\_\_pR\_Yp\}$ $\hookrightarrow = u_{pA,1}^{M_{pR}} = u_{pA,1}^{M_{pA}}$ $P_{INV,6}(M_{pR}) = \{pA\_Y, pA\_Yp\}$ $\hookrightarrow = u_{pA,2}^{M_{pR}} = u_{pA,1}^{M_{pA}}$	$T_{INV,1}(M_{pR}) = \{pR\_L\_\_t1, pR\_L\_\_t2\}$ $\hookrightarrow = E(u_{pR,1}^{M_{pR}}) = E(u_{L,1}^{M_{pR}})$ $T_{INV,2}(M_{pR}) = \{pR\_\_t1, pR\_\_t2\}$ $\hookrightarrow = E(u_{pR,2}^{M_{pR}})$ $T_{INV,3}(M_{pR}) = \{pR\_\_t3, pR\_\_t4\}$ $\hookrightarrow \subset E(u_{pR,3}^{M_{pR}})$ $T_{INV,4}(M_{pR}) = \{pA\_pR\_\_t1, pA\_pR\_\_t2\}$ $\hookrightarrow \subset E(u_{pR,3}^{M_{pR}})$, $\hookrightarrow = E(u_{pA,1}^{M_{pR}})$
M_{pA}	$P_{INV,1}(M_{pA}) = \{pA\_SH2, pA\_SH2\_\_pR\_Yp\}$ $\hookrightarrow = u_{pA,1}^{M_{pA}}$ $P_{INV,2}(M_{pA}) = \{pA\_Y, pA\_Yp\}$ $\hookrightarrow = u_{pA,2}^{M_{pA}}$ $P_{INV,3}(M_{pA}) = \{pA\_DNA\_BS\_inact, pA\_DNA\_BS\_act,$ $pA\_DNA\_BS\_act\_\_gR\_ABS\}$ $\hookrightarrow = u_{pA,3}^{M_{pA}}$ $P_{INV,4}(M_{pA}) = \{pR\_CD\_act\}$ $\hookrightarrow = u_{pR,1}^{M_{pA}} \subset u_{pR,2}^{M_{pR}}$ $P_{INV,5}(M_{pA}) = \{pR\_Yp, pA\_SH2\_\_pR\_Yp\}$ $\hookrightarrow = u_{pR,2}^{M_{pA}} \subset u_{pR,3}^{M_{pR}}$ $P_{INV,6}(M_{pA}) = \{gR\_ABS, pA\_DNA\_BS\_act\_\_gR\_ABS\}$ $\hookrightarrow = u_{gR,1}^{M_{pA}} = u_{gR,2}^{M_{gR}}$	$T_{INV,1}(M_{pA}) = \{pA\_pR\_\_t1, pA\_pR\_\_t2\}$ $\hookrightarrow = E(u_{pA,1}^{M_{pA}}) = E(u_{pR,2}^{M_{pA}})$ $T_{INV,2}(M_{pA}) = \{pA\_pR\_\_t3, pA\_\_t1\}$ $\hookrightarrow = E(u_{pA,2}^{M_{pA}})$ $T_{INV,3}(M_{pA}) = \{pA\_\_t2, pA\_\_t3\}$ $\hookrightarrow \subset E(u_{pA,3}^{M_{pA}})$ $T_{INV,4}(M_{pA}) = \{pA\_gR\_\_t1, pA\_gR\_\_t2\}$ $\hookrightarrow \subset E(u_{pA,3}^{M_{pA}})$, $\hookrightarrow = E(u_{gR,1}^{M_{pA}})$
M_{gR}	$P_{INV,1}(M_{gR}) = \{gR\_ABS, pA\_DNA\_BS\_act\_\_gR\_ABS\}$ $\hookrightarrow = u_{gR,1}^{M_{gR}}$ $P_{INV,2}(M_{gR}) = \{gR\_act, gR\_inact\}$ $\hookrightarrow = u_{gR,2}^{M_{gR}}$ $P_{INV,3}(M_{gR}) = \{pA\_DNA\_BS\_act\_\_gR\_ABS, pA\_DNA\_BS\_act\}$ $\hookrightarrow = u_{pA,1}^{M_{gR}} \subset u_{pA,3}^{M_{pA}}$	$T_{INV,1}(M_{gR}) = \{pA\_gR\_\_t1, pA\_gR\_\_t2\}$ $\hookrightarrow = E(u_{gR,1}^{M_{gR}}) = E(u_{pA,1}^{M_{gR}})$ $T_{INV,2}(M_{gR}) = \{gR\_\_t1, gR\_\_t2\}$ $\hookrightarrow \subset E(u_{gR,2}^{M_{gR}})$ $T_{INV,3}(M_{gR}) = \{pA\_gR\_\_t3, pA\_gR\_\_t4\}$ $\hookrightarrow \subset E(u_{gR,2}^{M_{gR}})$ $T_{INV,4}(M_{gR}) = \{pA\_gR\_\_t4, gR\_\_t1\}$ $\hookrightarrow \subset E(u_{gR,2}^{M_{gR}})$ $T_{INV,5}(M_{gR}) = \{pA\_gR\_\_t3, gR\_\_t2\}$ $\hookrightarrow \subset E(u_{gR,2}^{M_{gR}})$
M_{mR}	$P_{INV,1}(M_{mR}) = \{gR\_act\}$ $\hookrightarrow = u_{gR,1}^{M_{mR}} \subset u_{gR,1}^{M_{gR}}$	$T_{INV,1}(M_{mR}) = \{mR\_\_t1, gR\_mR\_\_t1\}$ $\hookrightarrow = E(u_{mR,1}^{M_{mR}})$

Table 3.7: P- and T-invariants of the running example.

the receptor protein module M_{pR}, cause these two modules to be not strongly connected. Since the receptor mRNA module M_{mR} contains molecular events representing the synthesis of the receptor mRNA and protein it is not bounded. According to the assumption on mass conservation, all other modules are structurally and k-bounded. In the uncoupled modules, transitions representing molecular events of interacting components are dead. Thus, none of the modules is live. In the receptor gene module M_{gR}, the two transitions representing the reversible basal activation process of the gene can be enabled, which causes the module to be reversible in contrast to all other modules. In consequence, the receptor gene module M_{gR} is the only module with no dead states.

Table 3.7 lists all P-invariants and T-invariants of the modules. Each functional unit of a component in the receptor gene module M_{gR}, the receptor protein module M_{pR}, and the adaptor protein module M_{pA} is represented by a minimal P-invariant. In the receptor mRNA module M_{mR} only the involved functional unit of the receptor gene is given by a minimal P-invariant, since all other functional units of the receptor mRNA and protein are involved in synthesis and degradation processes. Thus, expect the receptor mRNA module M_{mR}, all modules are covered by P-invariants. Each molecular process with a closed functional unit subnetwork is represented by a minimal T-invariant or a linear combination of minimal T-invariant. The linear combination of minimal T-invariants is necessary for the molecular processes given by:

- $E(u_{pR,3}^{M_{pR}}) = \{pR\_\_t3, pR\_\_t4, pA\_pR\_\_t1, pA\_pR\_\_t2\}$
  $\hookrightarrow T_{INV,3}(M_{pR}) + T_{INV,4}(M_{pR})$
- $E(u_{pA,3}^{M_{pA}}) = \{pA\_\_t2, pA\_\_t3, pA\_gR\_\_t1, pA\_gR\_\_t2\}$
  $\hookrightarrow T_{INV,3}(M_{pA}) + T_{INV,4}(M_{pA})$
- $E(u_{gR,1}^{M_{gR}}) = \{gR\_\_t1, gR\_\_t2, pA\_gR\_\_t3, pA\_gR\_\_t4\}$
  $\hookrightarrow T_{INV,2}(M_{gR}) + T_{INV,3}(M_{gR})$, or $T_{INV,4}(M_{gR}) + T_{INV,5}(M_{gR})$

All remaining molecular processes with an open functional unit subnetwork cannot be represented by a T-invariant, which includes the molecular process $E(u_{pA,2}^{M_{pR}})$ in the receptor protein module M_{pR} and $E(u_{pR,1}^{M_{mR}})$, $E(u_{pR,2}^{M_{mR}})$, $E(u_{pR,3}^{M_{mR}})$ in the receptor mRNA module M_{mR}. As a result, the receptor protein module M_{pR} and the receptor mRNA module M_{mR} are not covered by T-invariants, where the receptor gene module M_{gR} and the adaptor protein module M_{pA} are covered by T-invariants.

3.2 BMK - Module Annotation

This section requires:

- *XML [22]*
- *Module definition, Section 3.1.1*
- *Module types, Section 3.1.2*

As discussed by Le Novère et al. in [54] most published biological models are lost due missing access or insufficient characterisation. Le Novère et al. proposed standards for reference correspondence concerning the encoding of a model, its structure, and behaviour obtained from an instantiated simulation. Furthermore, Le Novère et al. suggested information to provide with the encoded model to be able to trace its origin and the people who were involved in its creation. These two requirements define a standard for curating and encoding models called MIRIAM (Minimum Information Requested In the Annotation of Models) [54]. We integrate the ideas proposed in MIRIAM [54] into the BMKfr by defining the module annotation format, called BioModelKit mark-up language (BMKml). The module annotation provides a sufficient characterisation of each module to allow its reuse and the assessment of its visibility by other users. The BMKml is a machine-readable Extensible Markup Language (XML) [22], defined by an XML Schema Definitions (XSD) and is MIRIAM compliant [54]. All valid BMKml documents must begin with an XML declaration <?xml . . . />, which specifies the attribute of the XML version (1.0) and the attribute of the document character encoding (UFT-8). Below, we explain the structure and elements defined in the BMKml.

The root element module, see XSD code snippet in Figure 3.4, of the XML annotation file has seven attributes:

Figure 3.4: XSD code snippet for the XML element module.

```
<xs:element name="module">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="infoList" type="bmk:infoListType"></xs:element>
            <xs:element name="placeList" type="bmk:placeListType"></xs:element>
            <xs:element name="transitionList" type="bmk:transitionListType"></xs:element>
            <xs:element name="parameterList" type="bmk:parameterListType"></xs:element>
        </xs:sequence>
        <xs:attribute name="dbName" use="required" type="xs:string" fixed="Ensembl"></
            xs:attribute>
        <xs:attribute name="id" use="required" type="xs:string"></xs:attribute>
        <xs:attribute name="name" use="required" type="xs:string"></xs:attribute>
        <xs:attribute name="type" use="required">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="protein module"></xs:enumeration>
                    <xs:enumeration value="protein degradation module"></xs:enumeration>
                    <xs:enumeration value="gene module"></xs:enumeration>
                    <xs:enumeration value="mRNA module"></xs:enumeration>
                    <xs:enumeration value="causal interaction module"></xs:enumeration>
                    <xs:enumeration value="allelic influence module"></xs:enumeration>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        </xs:complexType>
</xs:element>
```

- **dbName** identifies the name of the reference database in which the modelled genetic component is specified and referenced. The value is fixed to "Ensembl" [78], see also Table 3.8.
- **id** specifies a unique identifier for the genetic component represented by the module in respect to the Ensembl database [78] (required). Since each modularized genetic component can be traced back to its gene, we use the Ensembl identifier for genes (ENSG. . .) [78] independent of the module type.
- **name** specifies the name of the model, which can for example be the HGNC symbol [140] of the related gene (required).

- **type** specifies the module type according to the choice (required):
  - gene module
  - mRNA module
  - protein module
  - protein degradation module
  - causal interaction module
  - allelic influence module
- **xmlns** declares the XML namespaces applied within the MODULE element:
  - **xmlns:xs** with URI `http://www.w3.org/2001/XMLSchema-instance` (required)
  - **xmlns:bmk** with URI `http://www.biomodelkit.org` (required)
- **xs:schemaLocation** with URI `http://www.biomodelkit.org~module_metadata.xsd` (required)

Figure 3.5 (B) illustrates an example of the corresponding BMKML code. In addition, the root element MODULE holds the four child-elements INFOLIST, PLACELIST, TRANSITIONLIST, and PARAMETERLIST (all required). They provide information on different scopes of the model annotation, which will be explained in detail in the paragraphs below.

The element INFOLIST provides general information using four more child-elements to define the terms of use, the authors, the date of creation and last modification, and a description of the module, as well as a list of incorporated publication references, see XSD code snippet in Figure 3.5(A) and (B) for an example:

- TERMS element specifies the terms of use.
- AUTHORLIST element specifies a separate child element AUTHOR for each person involved in the modelling process.
  - AUTHOR element includes three attributes to hold the most relevant contact information of each person:
    - **firstName** specifies the first name of the author (required).
    - **lastName** specifies the last name of the author (required).
    - **email** specifies a valid email address of the author (required).
- DATE element specifies the date of creation and last modification using two attributes:
  - **creationDate** specifies the date of creation in the format "CCYY-MM-DD" (required)
  - **modificationDate** specifies the date of last modification in the format "CCYY-MM-DD" (required).
- DESCRIPTION element provides a free text description of the component given by the module (required).
- PUBREFLIST element provides a list of incorporated publication references (required), see paragraphs below and XSD code snippet in Figure 3.9.

Figure 3.5: (A) XSD code snippet for the XML element INFOLIST, (B) corresponding BMKML code snippet of the IL6 module including the MODULE element.

(A)

```
<xs:complexType name="infoListType">
    <xs:sequence>
        <xs:element name="terms" type="string"></xs:element>
        <xs:element name="authorList" type="bmk:authorListType"></xs:element>
        <xs:element name="date" type="bmk:dateType"></xs:element>
        <xs:element name="description" type="xs:string"></xs:element>
        <xs:element name="pubRefList" type="bmk:pubRefListType"></xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="dateType">
    <xs:attribute name="creationDate" use="required" type="xs:date"></xs:attribute>
    <xs:attribute name="modificationDate" use="required" type="xs:date"></xs:attribute>
</xs:complexType>
<xs:complexType name="authorListType">
    <xs:sequence>
        <xs:element name="author" maxOccurs="unbounded">
            <xs:complexType>
                <xs:attribute name="firstName" use="required" type="xs:string"></xs:attribute>
                <xs:attribute name="lastName" use="required" type="xs:string"></xs:attribute>
                <xs:attribute name="email" use="required">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:pattern value="[^@]+@[^\.]+\..+"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

(B)

```
<bmk:module dbName="Ensembl" id="ENSG00000136244" name="IL6" type="protein module" xmlns:xs="
        http://www.w3.org/2001/XMLSchema-instance" xs:schemaLocation="http://www.biomodelkit.org
        module_metadata.xsd" xmlns:bmk="http://www.biomodelkit.org">
    <bmk:infoList>
        <bmk:terms>copyright, freely distributable</bmk:terms>
        <bmk:authorList>
            <bmk:author firstName="Mary-Ann" lastName="Blaetke" email="mary-ann.blaetke@ovgu.
                de"></bmk:author>
        </bmk:authorList>
        <bmk:date creationDate="2012-02-03" modificationDate="2014-11-06" ></bmk:date>
        <bmk:description>Protein module of Interleukin 6</bmk:description>
        <bmk:pubRefList>
            <bmk:pubRef dbName="PubMed" id="9712900"></bmk:pubRef>
            <bmk:pubRef dbName="PubMed" id="9973404"></bmk:pubRef>
            <bmk:pubRef dbName="PubMed" id="7744001"></bmk:pubRef>
            ...
        </bmk:pubRefList>
    </bmk:infoList>
    ...
</bmk:module>
```

The element PLACELIST contains a number of PLACE child elements according to the number of places in the module given by $\mathcal{N}(M)$, see XSD code snippet and example in Figure 3.6. Each PLACE element has a single attribute:

- **name** specifies the name of the place (required)

and comprises three child elements to provide a description of the place, a list of components specified by the place, and a list of reference publications, see XSD code snippet in Figure 3.6:

- DESCRIPTION element provides a descriptive interpretation of the place.
- COMPONENTLIST element provides a list of components which are represented by the place. The list of components is separated into two list for genetic and non-genetic components using the child elements:
  - GCOMPONENTLIST, which contains a number of GCOMPONENT child elements. If the place does not represent any genetic component, the number of child elements is zero. Each GCOMPONENT element has the following attributes:
    - **dbName** specifies the name of the reference database, which is fixed to "Ensembl" [78], see also Table 3.8.

(A)

```
<xs:complexType name="placeListType">
    <xs:sequence>
        <xs:element name="place"  maxOccurs="unbounded">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="description" type="xs:string"></xs:element>
                    <xs:element name="componentList" type="bmk:componentListType"></xs:element
                        >
                    <xs:element name="pubRefList" type="bmk:pubRefListType"></xs:element>
                </xs:sequence>
                <xs:attribute name="name" use="required" type="xs:string"></xs:attribute>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="componentListType">
    <xs:sequence>
        <xs:element name="gComponentList" type="bmk:gComponentListType"></xs:element>
        <xs:element name="ngComponentList" type="bmk:ngComponentListType"></xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="gComponentListType">
    <xs:sequence>
        <xs:element name="gComponent"  minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="dbRefList" type="bmk:dbRefListType"></xs:element>
                </xs:sequence>
                <xs:attribute name="dbName" use="required" fixed="Ensembl"></xs:attribute>
                <xs:attribute name="id" use="required"></xs:attribute>
                <xs:attribute name="name" use="required"></xs:attribute>
                <xs:attribute name="type" use="required">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="protein"></xs:enumeration>
                            <xs:enumeration value="gene"></xs:enumeration>
                            <xs:enumeration value="mRNA"></xs:enumeration>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="ngComponentListType">
    <xs:sequence>
        <xs:element name="ngComponent" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="dbRefList" type="bmk:dbRefListType"></xs:element>
                </xs:sequence>
                <xs:attribute name="dbName" use="required" fixed="PubChem"></xs:attribute>
                <xs:attribute name="id" use="required" ></xs:attribute>
                <xs:attribute name="name" use="required"></xs:attribute>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

(B)

```
<bmk:place name="IL6_siteI:IL6R_CBM">
    <bmk:description>
            cytokine binding module (CBM) of IL6R bound to binding site I (siteI) of IL6
    </bmk:description>
    <bmk:componentList>
        <bmk:gComponentList>
            <bmk:gComponent dbName="Ensembl" id="ENSG00000160712" type="protein" name="
                    IL6R_CBM">
                <bmk:dbRefList>
                    <bmk:dbRef dbName="Uniprot" id="P08887"></bmk:dbRef>
                    <bmk:dbRef dbName="InterPro" id="IPR003961"></bmk:dbRef>
                    <bmk:dbRef dbName="Pfam" id="PF00041"></bmk:dbRef>
                            ...
                </bmk:dbRefList>
            </bmk:gComponent>
            <bmk:gComponent dbName="Ensembl" id="ENSG00000136244" type="protein" name="
                    IL6_siteI">
                <bmk:dbRefList>
                    <bmk:dbRef dbName="Uniprot" id="P05231"></bmk:dbRef>
                    <bmk:dbRef dbName="InterPro" id="IPR009079"></bmk:dbRef>
                    <bmk:dbRef dbName="InterPro" id="IPR012351"></bmk:dbRef>
                            ...
                </bmk:dbRefList>
            </bmk:gComponent>
        </bmk:gComponentList>
        <bmk:ngComponentList></bmk:ngComponentList>
    </bmk:componentList>
    <bmk:pubRefList>
        <bmk:pubRef dbName="PubMed" id="12829785"></bmk:pubRef>
        <bmk:pubRef dbName="PubMed" id="2261637"></bmk:pubRef>
        <bmk:pubRef dbName="PubMed" id="8706672"></bmk:pubRef>
        ...
    </bmk:pubRefList>
</bmk:place>
```

Figure 3.6: (A) XSD code snippet for the XML element placeList, (B) corresponding BMKml code for place *IL6_siteI__IL6R_CBM*.

* **id** specifies a unique identifier for the corresponding gene of the genetic component (Ensembl identifier for genes (ENSG...) [78], required).
* **name** specifies the sub-string of the place name representing

the genetic component (required).
    - **type** specifies the genetic type according to the choice (required): gene, mRNA and protein.
  - ngComponentList, which contains a number of ngComponent child elements. If the place does not represent any non-genetic component, the number of child elements is zero. Each ngComponent element has the following attributes:
    - **dbName** specifies the name of the reference database, which is fixed to "PubChem" [75], see also Table 3.8.
    - **id** specifies a unique identifier for the corresponding gene of the non-genetic component (PubChem identifier (CID) [75], required).
    - **name** specifies the sub-string of the place name representing the non-genetic component (required).

    Both elements gComponent, ngComponent have a single child element:
    - dbRefList element provides a list of references to other relevant biomolecular databases, see paragraphs below and XSD code snippet in Figure 3.10.
- pubRefList element provides a list of reference publications, see paragraphs below and XSD code snippet in Figure 3.9.

Figure 3.7: (A) XSD code snippet for the XML element transitionList, (B) corresponding BMKml code for transition *IL6_IL6R_IL6ST_t2*.

(A)

```
<xs:complexType name="transitionListType">
    <xs:sequence>
        <xs:element name="transition" maxOccurs="unbounded">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="description" type="xs:string"></xs:element>
                    <xs:element name="dbRefList" type="bmk:dbRefListType"></xs:element>
                    <xs:element name="pubRefList" type="bmk:pubRefListType"></xs:element>
                </xs:sequence>
                <xs:attribute name="name" use="required" type="xs:string"></xs:attribute>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

(B)

```
<bmk:transition name="IL6_IL6R_IL6ST_t2">
    <bmk:description>
            dissociation of the complex between the CBM of IL6ST, binding site IIa of IL6 and binding site IIb of IL6R
    </bmk:description>
    <bmk:dbRefList>
        <bmk:dbRef dbName="GO" id="GO:0005515"></bmk:dbRef>
    </bmk:dbRefList>
    <bmk:pubRefList>
        <bmk:pubRef dbName="PubMed" id="9712900"></bmk:pubRef>
        <bmk:pubRef dbName="PubMed" id="9973404"></bmk:pubRef>
        <bmk:pubRef dbName="PubMed" id="7744001"></bmk:pubRef>
        ...
    </bmk:pubRefList>
</bmk:transition>
```

The element transitionList contains a number of transition child elements according to the number of transitions in the module given by $\mathcal{N}(M)$, see XSD code snippet and example in Figure 3.7. Each transition element has a single attribute:

- **name** specifies the name of the transition (required)

and comprises three child elements to provide a description of the transition, a list of reference publications and other used database entries, see XSD code snippet in Figure 3.7:

(A)

```
<xs:complexType name="parameterListType">
    <xs:sequence>
        <xs:element name="parameter"  minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="description" type="xs:string"></xs:element>
                    <xs:element name="dbRefList" type="bmk:dbRefListType"></xs:element>
                    <xs:element name="pubRefList" type="bmk:pubRefListType"></xs:element>
                </xs:sequence>
                <xs:attribute name="name" use="required" type="xs:string"></xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

(B)

```
<bmk:parameter name="k">
    <bmk:description>constant</bmk:description>
    <bmk:dbRefList></bmk:dbRefList>
    <bmk:pubRefList></bmk:pubRefList>
</bmk:parameter>
```

Figure 3.8: (A) XSD code snippet for the XML element PARAMETERLIST, (B) corresponding BMKML code for parameter k.

- DESCRIPTION element provides a descriptive interpretation of the transition.
- DBREFLIST element provides a list of references to other relevant biomolecular databases, see paragraphs below and XSD code snippet in Figure 3.10.
- PUBREFLIST element provides a list of reference publications, see paragraphs below and XSD code snippet in Figure 3.9.

The element PARAMETERLIST contains a number of PARAMETER child elements according to the number of parameters defined in the module given by $\mathcal{N}(M)$, see XSD code snippet in Figure 3.8. Each PARAMETER element has a single attribute:

- **name** specifies the name of the parameter (required)

and comprises three child elements to provide a description of the parameter, a list of publication references and other cross-references:

- DESCRIPTION element provides a descriptive interpretation of the parameter.
- DBREFLIST element provides a list of reference to other relevant biomolecular databases, see paragraphs below and XSD code snippet in Figure 3.10.
- PUBREFLIST element provides a list of reference publications, see paragraphs below and XSD code snippet in Figure 3.9.

```
<xs:complexType name="pubRefListType">
    <xs:sequence>
        <xs:element name="pubRef"  maxOccurs="unbounded">
            <xs:complexType>
                <xs:attribute name="dbName" use="required" fixed="PubMed"></xs:attribute>
                <xs:attribute name="id" use="required"></xs:attribute>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

Figure 3.9: XSD code snippet for the XML element PUBREFLIST.

The element PUBREFLIST contains a number of PUBREF child elements, see XSD code snippet and example in Figure 3.9 (examples can be found in Figures 3.5, 3.6 and 3.7). Each PUBREF element has two attributes:

Figure 3.10: XSD code snippet for the XML element dbRefList.

```
<xs:complexType name="dbRefListType">
    <xs:sequence>
        <xs:element name="dbRef"  maxOccurs="unbounded">
            <xs:complexType>
                <xs:attribute name="dbName" use="required">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="Gene3D"></xs:enumeration>
                            <xs:enumeration value="HAMAP"></xs:enumeration>
                            <xs:enumeration value="InterPro"></xs:enumeration>
                            <xs:enumeration value="PANTHER"></xs:enumeration>
                            <xs:enumeration value="Pfam"></xs:enumeration>
                            <xs:enumeration value="PIRSF"></xs:enumeration>
                            <xs:enumeration value="PRINTS"></xs:enumeration>
                            <xs:enumeration value="ProDom"></xs:enumeration>
                            <xs:enumeration value="PROSITE"></xs:enumeration>
                            <xs:enumeration value="SMART"></xs:enumeration>
                            <xs:enumeration value="SUPFAM"></xs:enumeration>
                            <xs:enumeration value="TIGRFAMs"></xs:enumeration>
                            <xs:enumeration value="GlycoSuiteDB"></xs:enumeration>
                            <xs:enumeration value="PhosphoSite"></xs:enumeration>
                            <xs:enumeration value="PhosSite"></xs:enumeration>
                            <xs:enumeration value="GO"></xs:enumeration>
                            <xs:enumeration value="Ensembl"></xs:enumeration>
                            <xs:enumeration value="Uniprot"></xs:enumeration>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="id" use="required"></xs:attribute>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

- **dbName** specifies the name of the reference database, which is fixed to "PubMed" [32](required), see also Table 3.8.
- **id** specifies a unique identifier of publication in the PubMed database "PMID" [32] (required).

The element dbRefList contains a number of dbRef child elements, see XSD code snippet in Figure 3.10 (examples can be found in Figures 3.5, 3.6 and 3.7). Each dbRef element has two attributes:

- **dbName** specifies the name of the reference database according to the choice of databases specified in Table 3.8, excluding the PubMed database [32](required). The value is set to the abbreviation of the chosen database, compare Figure 3.10 and Table 3.8.
- **id** specifies a unique identifier of the reference according to the chosen database (required).

MIRIAM compliance

As already mentioned at the beginning of this section, MIRIAM is a standard format for the *Minimum Information Requested In the Annotation of biochemical Models* to improve the quality of a unevocal model documentation and reusability, which has been proposed by Le Novère et al. [54]. First of all, models must be encoded in a public, machine-readable format, which is given for the modules defined in the BMKfr, since we are using Snoopy [120] to define the underlying Petri net model. Snoopy [120] is publicly available and employs a machine-readable XML format to store models, but also supports the export and import of SBML. Thus, each module inherently obeys the standards and syntax of Petri nets. Since modules represent a particular genetic component, we decided to provide the corresponding Ensembl id [78] as the main reference. The module reflects the biological processes listed for the corresponding Ensembl id, respectively given in cross-references that can be approached from the web-site of the corresponding Ensembl id. With the module, we

Name (Abbreviation)	Reference
Gene3D Structural and Functional Annotation of Protein Families (Gene2D)	[65]
HAMAP database of protein families (HAMAP)	[38]
Integrated resource of protein families, domains and functional sites (InterPro)	[72]
The PANTHER Classification System (PANTHER)	[71]
Pfam protein domain database (Pfam)	[59]
PIRSF; a whole-protein classification database	[50]
Protein Motif fingerprint database; a protein domain database (PRINTS)	[34]
ProDom; a protein domain database (ProDom)	[52]
PROSITE; a protein domain and family database (PROSITE)	[61]
Simple Modular Architecture Research Tool; a protein domain database (SMART)	[63]
Superfamily database of structural and functional annotation (SUPFAM)	[30]
TIGRFAMs; a protein family database (TIGRFAMs)	[73]
GlycoSuiteDB; an annotated and curated relational database of glycan structures (GlycoSuiteDB)	[36]
Phosphorylation site database (PhosphoSite)	[47]
Phosphorylation site database for Archaea and Bacteria (PhosSite)	[51]
Gene Ontology (GO)	[79]
Genome annotation databases (Ensembl)	[78]
Protein Knowledgebase UniprotKB (Uniprot)	[143]
PubChem; database of chemical molecules and their activities against biological assays (PubChem)	[75]
US National Library of Medicine National Institutes of Health (PubMed)	[32]

Table 3.8: External reference databases.

do also provide all relevant information to instantiate its simulation, including all initial conditions, marking, parameter values, and kinetic expressions. Thus, the behaviour of the genetic component represented by the module, which is given by the Ensembl reference description can be reproduced. The annotation file of a module, which is also written in the machine-readable BMKML-format (see above), specifies a preferred module name and lists reference publications using PubMed identifiers [32]. The listed reference publications give a complete mechanistic description of the components represented by the module. Even more, each place and each transition are linked to a set of reference publications and entries in other relevant biomolecular databases, see Table 3.8. Also, the annotation file includes all persons involved in the model creation, the date of creation and last modification, as well as a statement about the terms of distribution. In summary, modules defined in the BMKFR comply with the standards postulated by MIRIAM [54].

3.3 BMK - Modular Model Composition

As introduced in Section 3.1.1, modules of interacting components are inherently equipped with redundant interface networks, which are required to modulary compose models. During the modular composition redundant interface networks are automatically matched due to the use of logical (fusion) nodes, making manual modifications and manual matching of nodes dispensable, see Section 3.3.1 and 3.3.2. Based on the modular model composition, we introduce the algorithmic model mutation inin Section 3.3.3 and the spatial transformation of modularly composed models in Section 3.3.4. All of the algorithms to compose, mutate, and spatially transform modularly composed models can be fully automatized by organising modules in the BMKDB, see Section 4.

3.3.1 Modular Model Composition

This section requires:

- *Petri nets [82], Section 2.1*
- *Module definition, Section 3.1.1*
- *Module types, Section 3.1.2*
- *Petri net representation of modules, Section 3.1.3*

A set of modules $G = \{M_1, \ldots, M_{n_M}\}$, $n_M \geq 1$, defines a modularly composed model defined by the Petri net graph $\mathcal{N}(G) = (P^G, T^G, F^G, f^G, v^G, m_0^G)$. The set of modules G can also be written as $G = G_g \cup G_m \cup G_p \cup G_d \cup G_{ai} \cup G_{ci}$, where:

- $G_g = \{M_{g,c_{0,1}}, \ldots, M_{g,c_{0,n_g}}\}$ - subset of gene modules
- $G_m = \{M_{m,c_{0,1}}, \ldots, M_{m,c_{0,n_m}}\}$ - subset of mRNA modules
- $G_p = \{M_{p,c_{0,1}}, \ldots, M_{p,c_{0,n_p}}\}$ - subset of protein modules
- $G_d = \{M_{d,c_{0,1}}, \ldots, M_{d,c_{0,n_d}}\}$ - subset of protein degradation modules
- $G_{ai} = \{M_{ai,c_{0,1}}, \ldots, M_{ai,c_{0,n_{ai}}}\}$ - subset of allelic influence modules
- $G_{ci} = \{M_{ci,c_{0,1}}, \ldots, M_{ci,c_{0,n_{ci}}}\}$ - subset of causal influence modules

All module subsets G_g, G_m, G_p, G_d, G_{ai}, and G_{ci} are distinct.

The Petri net graph of the composed model $\mathcal{N}(G) = (P^G, T^G, F^G, f^G, v^G, m_0^G)$ is given by:

- Set of places $P^G = \bigcup_{M \in G} P$,
- Set of transitions $T^G = \bigcup_{M \in G} T$,
- Set of arcs $F^G = \bigcup_{M \in G} F$,
- Arc-weights f^G: $F^G \to \mathbb{N}_0$
  - $f^G \in F^G$, where $f^G \in F^{M_1}, \ldots, F^{M_m}$ with $\{M_1, \ldots, M_m\} \subseteq G$: $f^G = f^{M_1} = \ldots = f^{M_m}$
- Firing rates v^G: $T^G \to H^G$, $H^G = \bigcup_{M \in G} H$
- Initial marking: $m_0^G : P^G \to \mathbb{N}_0$, where
  - $\forall e \in P^G$ with $\varrho_{p \to s}(p) \in \bigcup_{M \in G} \mathcal{S}_0(M)$: $m_0^G(p) = n_{c_0}$ with $c_0 = \lambda^c(\varrho_{p \to s}(p))$
  - $\forall e \in P^G$ with $\varrho_{p \to s}(p) \notin \bigcup_{M \in G} \mathcal{S}_0(M)$: $m_0^G(p) = 0$

Furthermore, we introduce the following notations for the composed model $\mathcal{N}(G)$:

- $C^G = \bigcup_{M \in G} C(M)$ - the total set of components,
- $\mathcal{S}^G = \bigcup_{M \in G} \mathcal{S}(M)$ - the total set of molecular states,
- $\mathcal{S}_0^G = \bigcup_{M \in G} \mathcal{S}_0(M)$ - the total set of home states,

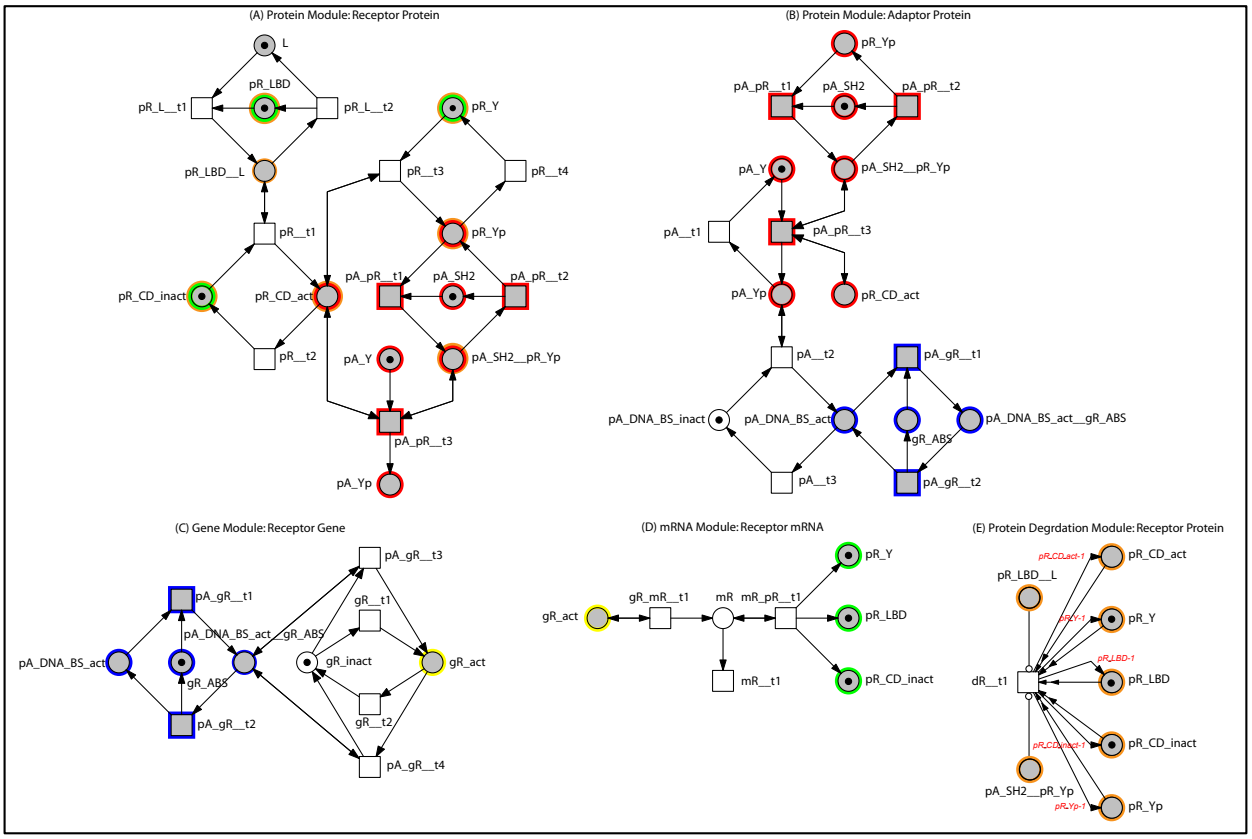


Figure 3.11: A modularly composed model of the running example. The individual modules of the running example, see Figure 3.3, are now connected in a comprehensive modular model. Nodes in the interface networks among the modules are automatically matched. The marking has been set according to the defined rules. The interface networks shared by the modules of the involved components are indicated by logical nodes with coloured borders: receptor and adaptor protein (red), receptor gene and adaptor protein (blue), receptor gene and mRNA (yellow), receptor protein and mRNA (green), receptor protein and degradation (orange).

- $\mathcal{S}_{IS}^{G} = \bigcup_{M \in G} \mathcal{S}_{IS}(M)$ - the total set of complexes,
- $\mathcal{E}_{IA}^{G} = \bigcup_{M \in G} \mathcal{E}_{IA}(M)$ - the total set of all interactions,
- $k = \lambda^{c}(s)$, $s \in \mathcal{S}_{IS}^{G}$ - a complex of mapped components, and
- $K^{G} = \bigcup_{s \in \mathcal{S}_{IS}^{G}} k$ - total set of different complexes.

The modular model composition builds the foundation of the algorithmic model mutation and spatial transformation of modularly composed models, see Section 3.3.3 and 3.3.4.

Running Example. From the set of modules defined by $G = \{M_{gR}, M_{mR}, M_{pR}, M_{dR}, M_{pA}\}$ in Figure 3.3, we generated the composed model $\mathcal{N}(G)$, compare Figure 3.11. Nodes part of an interface network are automatically matched in order to connect the modules in G. Places representing home states of a module shared by interface networks are consistently marked in all modules in the composed model:

- $N_I(M)$ with $I = \{M_{gR}, M_{mR}\}$:
  - does not include places representing home states
- $N_I(M)$ with $I = \{M_{mR}, M_{pR}\}$:
  - $m^{G}(pR\_LBD) = 1$
  - $m^{G}(pR\_CD\_inact) = 1$
  - $m^{G}(pR\_Y) = 1$
- $N_I(M)$ with $I = \{M_{pR}, M_{dR}\}$:
  - $m^{G}(pR\_LBD) = 1$

  - $m^G(pR\_CD\_inact) = 1$
  - $m^G(pR\_Y) = 1$
- $N_I(M)$ with $I = \{M_{pR}, M_{pA}\}$:
  - $m^G(pA\_Y) = 1$
  - $m^G(pA\_Yp) = 1$
  - $m^G(pA\_SH2) = 1$

3.3.2 *Instantiation of Modularly Composed Models*

This section requires:

- *Petri nets [82], Section 2.1*
- *Coloured Petri nets [122], Section 2.3*
- *Module definition, Section 3.1.1*
- *Module types, Section 3.1.2*
- *Petri net representation of modules, Section 3.1.3*
- *Modular model composition, Section 3.3.1*

Hitherto, we assumed only a single instance for each main component c_0 represented by a module M_{c_0} by setting $n_{c_0} = 1$. For a biomolecular system, this might not always be true. In a single cell, usually, multiple copies of proteins and mRNAs exist, which is even more true for non-genetic components. Considering a multi-cellular or a polynuclear organism, multiple copies of a gene must be considered. Thus, the instantiation of a modularly composed model $\mathcal{N}(G)$ needs to be changed, which can be accomplished by two approaches. The approach to choose depends on whether the copies of components should be indistinguishable from each other in the modularly composed model $\mathcal{N}(G)$ or not. It might be necessary to distinguish copies of a component in the composed model, if the structural aspects of the cell like pools, compartments and membranes are considered, if single cells should be differentiated from each other, or if the movement or behaviour of single components is of interest rather than the overall behaviour of the represented molecular network. If the copies of components are assumed to be not distinguishable in the modularly composed model $\mathcal{N}(G)$, the instantiation can be adjusted solely by the marking. Otherwise, if the copies of components are distinguishable from each other, it is not sufficient to solely change the marking. The components represented in the modularly composed model $\mathcal{N}(G)$ need to be equipped with an additional attribute to differentiate each copy of a component from each other. The instantiation is accomplished by transforming the modularly composed model into a coloured Petri net, where the copies of a component are handled by component specific colour sets. Thus, each copy of a component is defined by a colour of the component specific colour set, which allows distinguishing copies of components.

Instantiation by Adjusting the Marking

The most convenient way to instantiate the modularly composed model $\mathcal{N}(G)$ is to individually adjust the marking of components if instances should be indistinguishable. In the case of genetic components, the parameter n_{c_0} determining number of copies of a main component $c_0 \in C^G$ can be set to any other value greater one. The marking of places representing home states of component c_0 is automatically adjusted. Otherwise, in the case of non-genetic components, we first need to define a parameter $n_{c_{ng}} \to \mathbb{N}$ for each non-genetic component $c_{ng} \in C^G$. The set of places representing a non-genetic component c_{ng}, $P^{c_{ng}} = \{p \in P : \lambda^c(\varrho_{p \to s}(p)) = c_{ng}\}$, should be marked according to the rule $\sum_{p \in P^{c_{ng}}} m_0(p) = n_{c_{ng}}$.

Instantiation by Applying Coloured Petri Nets

The transformation of the modularly composed model $\mathcal{N}(G) = (P^G, T^G, F^G, f^G, v^G, m_0^G)$ into a coloured Petri net $\mathcal{N}(G_{col}) = \{P^{G_{col}}, T^{G_{col}}, F^{G_{col}}, W^{G_{col}}, \psi^{G_{col}}, g^{G_{col}}, f^{G_{col}}, v^{G_{col}}, m_0^{G_{col}}\}$ is necessary, if the identity of components instances should be distinguishable. Algorithm 3.1 summarizes the steps of the coloured Petri net transformation. The model instantiation by applying coloured Petri nets is also a part of the spatial transformation algorithm, see Section 3.3.4.

For each component $c \in C^G$ we need to specify an upper bound for the number of possible instances, $n_{max,c} \to \mathbb{N}$, as well as an initial value $n_{ini,c} \to \mathbb{N}$, $n_{ini,c} \leq n_{max,c}$. The coloured Petri net $\mathcal{N}(G_{col}) = \{P^{G_{col}}, T^{G_{col}}, F^{G_{col}}, W^{G_{col}}, \psi^{G_{col}}, g^{G_{col}}, f^{G_{col}}, v^{G_{col}}, m_0^{G_{col}}\}$ is defined as follows:

- $P^{G_{col}} = P^G \cup P^C$ - set of coloured places, where $P^C = \{p_{c_1}, \ldots, p_{c_{|C^G|}}\}$. Each component of $c \in C^G$ is represented by an additional place p_c, called component place.
- $T^{G_{col}} = T^G$ - set of coloured transitions
- $F^{G_{col}} = F^G$ - set of coloured arcs
- $W^{G_{col}} = W_{simple} \cup W_{compound}$ - set of colour sets
  - $W_{simple} = \{w'_{c_1}, \ldots, w'_{c_{|C^G|}}\}$ - set of simple colour sets, where each component of $c \in C^G$ is represented by a simple colour set $w'_c = 1 - n_{max,c}$ with $n_{max,c}$ colours.
  - $W_{compound} = \{w''_{s_{is},1}, \ldots, w''_{s_{is},|K^G|}\}$ - set of compound colour sets, where each complex $s_{is} \in \mathcal{S}_{IS}^G$ is represented by a product colour set based on the set of components $\lambda^c(s_{is})$ defining the complex, such that $w''_k = \prod_{c \in \lambda^c(s_{is})} w'_c$.
- $\psi^{G_{col}} : P^{G_{col}} \to W^{G_{col}}$ - colour function:
  - $\psi(p) = w'_c$ if $c = \lambda^c(\varrho_{p \to s}(p))$,
  - $\psi(p_c) = w'_c$, and
  - $\psi(p) = w''_k$ if $k = \lambda^c(\varrho_{p \to s}(p))$
- $g^{G_{col}} : T^{G_{col}} \to \emptyset$ - guard function
- $f^{G_{col}} : F^{G_{col}} \to EXP$ - arc function:
  - All outgoing and ingoing arcs, $f(p,t)$ and $f(t,p)$, of a place $p \in P^{G_{col}}$ with $\psi(p) = w'_c$ are extended to the multiset expression $f(p,t)^{G_{col}} = f(p,t)`a_c$, or respectively $f(t,p)^{G_{col}} = f(t,p)`a_c$, where $t \in T^{G_{col}}$, where a_c is defined as a variable of the simple colour set w'_c.
  - All outgoing and ingoing arcs, $f(p,t)$ and $f(t,p)$, of a place $p \in P^{G_{col}}$ with $\psi(p) = w''_k$ are extended to the multiset expression $f(p,t)^{G_{col}} = f(p,t)`\bigcup_{c \in k} a_c$, or respectively $f(p,t)^{G_{col}} = f(t,p)`\bigcup_{c \in k} a_c$. In case for inhibitory arcs $f_{IA}(p,t)$, the multiset expression must be composed as follows:

$$f_{IA}^{G_{col}}(p,t) = f_{IA}(p,t)`\bigcup_{c \in \lambda^c(\varrho_{p \to s}(p))} \begin{cases} a_c, & c = c' \\ all(), c \neq c' \end{cases}$$

The expression $all()$ is a function filtering all colours of a colour set.

- $v^{G_{col}} : T^{G_{col}} \to H^G$ - firing rate function:

- $h^{G_{col}}(t) = h(t)`all()$

- $m_0^{G_{col}} : P^{G_{col}} \rightarrow EXP$ - marking initialization function:
  - $\forall p_c \in P^C$: $m_0^{G_{col}}(p_c) = 1`1 + \ldots + 1`n_{ini,c} \hat{=} 1`all()$„
  - $\forall p \in \bigcup_{s \in \mathcal{S}_0^G} \varrho_{p \rightarrow s}^{-1}(s)$ with $c = \lambda^c(\varrho_{p \rightarrow s}(p))$ and c is a genetic component: $m_0^{G_{col}}(p) = 1`1 + \ldots + 1`n_{ini,c} \hat{=} 1`all()$
    - $\forall p \in P^{G_{col}}$ with $\lambda^c(\varrho_{p \rightarrow s}(p)) = c$ and c is a non-genetic component: $m_0^{G_{col}}(p) = 1`1 + \ldots + 1`n_{ini,c} \hat{=} 1`all()$
  - $\forall p \in \bigcup_{s \in \mathcal{S}_0^G} \varrho_{p \rightarrow s}^{-1}(s)$ and $c = \lambda^c(\varrho_{p \rightarrow s}(p))$ if c is a genetic component, and
  - $m_0^{G_{col}}(p) = 1`1 + \ldots + 1`n_{ini,c} \hat{=} 1`all()$, $c = \lambda^c(\varrho_{p \rightarrow s}(p))$ if c is non-genetic component.

Modifications are necessary if the composed model contains transitions describing the synthesis or degradation of a component. Indeed, this is the case for mRNA modules and protein degradation modules. A transition $t \in T^{G_{col}}$ representing the synthesis of a component $c \in C^G$, such that $c \notin \bigcup_{p \in {}^\bullet t} \lambda^c(\varrho_{p \rightarrow s}(p))$ and $c \in \bigcup_{p \in t^\bullet} \lambda^c(\varrho_{p \rightarrow s}(p))$, is connected to the component place $p_c \in P^C$ via an inhibitory arc $f_{IA}^{G_{col}}(p_c, t) = 1`a_c$ and a standard arc $f_{SA}^{G_{col}}(t, p_c) = 1`a_c$. The additional arcs ensure, that if an instance of a component already exists, it cannot be synthesised a second time. In reverse, a transition $t \in T^{G_{col}}$ representing the degradation of a component $c \in C^G$, such that $c \notin \bigcup_{p \in t^\bullet} \lambda^c(\varrho_{p \rightarrow s}(p))$ and $c \in \bigcup_{p \in {}^\bullet t} \lambda^c(\varrho_{p \rightarrow s}(p))$, is connected to the component place $p_c \in P^C$ via an additional standard arc $f_{SA}^{G_{col}}(p_c, t) = 1`a_c$.

There might exist other situations that have not been covered so far. Below, we introduce two more cases, which need further intention:

Special Case 1. Another case that implies further modifications are molecular events that are involved in homodimerization or -multimerisation processes, where the respective transitions represent interaction among several instances of the same component or complex, see Figure 3.13(A) for illustration. Such transitions can be identified through arcs connecting a place $p \in P^G$ with a transition $t \in T^G$, that has an arc-weight greater than one, $f(x, y) \geq 1$, $x, y \in P^G \cup T^G$. Meaning, two or more instances of a component or complex interact with each other through a particular molecular state s of the same functional unit. If the molecular state s represented by the place p can only be mapped to a single component $c = \lambda^c(s)$ with $|\lambda^c(s)| = 1$, $f(x, y)$ additional variables $a_c^1, \ldots, a_c^{f(x,y)}$ need to be defined for the simple colour set w_c' of component c. In the coloured model the corresponding arc-expression is changed to $f_{col}^G(x, y) = 1`a_c^1 + \ldots + 1`a_c^{f(x,y)}$. If the molecular state s represented by the place p defines a complex (set of components) $k = \lambda^c(s)$, $|\lambda^c(s)| > 1$, $f(x, y)$ additional variables $a_c^1, \ldots, a_c^{f(x,y)}$ need to be defined for the simple colour set w_c' of each component $c \in k$. In the coloured model the corresponding arc-expression is $f_{col}^G(x, y) = 1` \bigcup_{c \in k} a_c^1 + \ldots + 1` \bigcup_{c \in k} a_c^{f(x,y)}$.

Algorithm 3.1: Coloured Petri net transformation of modularly composed models.

Require: modularly composed model $\mathcal{N}(G) = \{P^G, T^G, F^G, f^G, v^G, m_0^G\}$

$P^{G_{col}} = P^G$
$T^{G_{col}} = T^G$
$F^{G_{col}} = F^G$
$W^{G_{col}} = \emptyset$
$g^{G_{col}} : T \to \emptyset$
▷ initialization of the coloured Petri net

$P^C = \emptyset$
$A^C = \emptyset$
for all $c \in C^G$ **do**
  Define $n_{max,c}$ ▷ define maximum number of component instances
  Define $n_{ini,c}$ ▷ define initial number of component instances
  $P^C = P^C \cup p_c$ ▷ create component place
  $A^C = A^C \cup a_c$ ▷ create component variable
  $w'_c = \{1, \ldots, n_{max,c}\}$ ▷ create component colour set
  $W^{G_{col}} = W^{G_{col}} \cup w'_c$
  $\psi(p_c) = w'_c$ ▷ assign colour set to component place
  $m_0^{G_{col}}(p_c) = m_0^G(p)`1 + \ldots + m_0^G(p)`n_{ini,c}$ ▷ set m_0 of component place
  for all $p \in P^{G_{col}}$ **do**
    if $c = \lambda^c(\varrho_{p \to s}(p))$ **then**
      $\psi(p) = w'_c$ ▷ assign colour set to respective places
      for all $t \in T^{G_{col}}$ **do**
        $f_{SA}^{G_{col}}(p,t) = f_{SA}^G(p,t)`a_c$
        $f_{SA}^{G_{col}}(t,p) = f_{SA}^G(t,p)`a_c$
        $f_{RA}^{G_{col}}(p,t) = f_{RA}^G(p,t)`a_c$
        $f_{EA}^{G_{col}}(p,t) = f_{EA}^G(p,t)`a_c$
        $f_{IA}^{G_{col}}(p,t) = f_{IA}^G(p,t)`a_c$
        ▷ adjust arc-expression according to colour set of the place
      end for
      $m_0^{G_{col}}(p) = m_0^G(p)`1 + \ldots + m_0^G(p)`n_{ini,c}$ ▷ set m_0
    end if
  end for
end for
$P^{G_{col}} = P^{G_{col}} \cup P^C$

for all $k \in K^G$ **do**
  $A_k = \emptyset$
  $A'_k = \emptyset$
  $w''_k = \emptyset$
  for all $c \in k$ **do**
    $w''_k = w''_k \times w'_c$
    ▷ create product colour set based on simple colour set of involved components
    $A_k = A_k \cup a_c$
    ▷ create variable set based on variables of involved components
    if $c = c_0$ **then**
      $A'_k = A'_k \cup a_c$
    else
      $A'_k = A'_k \cup all()$
    end if
  end for
  $W^{G_{col}} = W^{G_{col}} \cup w''_k$
  for all $p \in P^{G_{col}}$ **do**
    if $k = \lambda^c(\varrho_{p \to s}(p))$ **then**
      $\psi(p) = w''_k$
      for all $t \in T^{G_{col}}$ **do**
        $f_{SA}^{G_{col}}(p,t) = f_{SA}^G(p,t)`A_k$
        $f_{SA}^{G_{col}}(t,p) = f_{SA}^G(t,p)`A_k$
        $f_{RA}^{G_{col}}(p,t) = f_{RA}^G(p,t)`A_k$
        $f_{EA}^{G_{col}}(p,t) = f_{EA}^G(p,t)`A_k$
        $f_{IA}^{G_{col}}(p,t) = f_{IA}^G(p,t)`A'_k$
        ▷ adjust arc-expression according to product colour set of the place
      end for
      $m_0^{G_{col}}(p) = m_0^G(p)`1 + \ldots + m_0^G(p)`n_{ini,c}$ ▷ set m_0
    end if
  end for
end for

Algorithm 3.1: Coloured Petri net transformation of modularly composed models (cont.).

60: **for all** $t \in T^{G_{col}}$ **do**
61: $\quad h^{G_{col}}(t) = h^{G}(t)`all()$ ▷ set firing-rates of transitions
62: **end for**

63: **for all** $c \in C^{G}$ **do**
64: $\quad$ **for all** $t \in T^{G_{col}}$ **do**
65: $\quad\quad$ **if** $c \notin \bigcup_{p \in {}^\bullet t} \lambda^{c}(\varrho_{p \to s}(p))$ & $c \in \bigcup_{p \in t^\bullet} \lambda^{c}(\varrho_{p \to s}(p))$ **then**
▷ find transition representing the synthesis of a component
66: $\quad\quad\quad f_{IA}^{G_{col}}(p_c, t) = 1`a_c$
67: $\quad\quad\quad f_{SA}^{G_{col}}(t, p_c) = 1`a_c$
68: $\quad\quad$ **end if**
69: $\quad\quad$ **if** $c \notin \bigcup_{p \in t^\bullet} \lambda^{c}(\varrho_{p \to s}(p))$ & $c \in \bigcup_{p \in {}^\bullet t} \lambda^{c}(\varrho_{p \to s}(p))$ **then**
▷ find transition representing the degradation of a component
70: $\quad\quad\quad f_{SA}^{G_{col}}(p_c, t) = 1`a_c$
71: $\quad\quad$ **end if**
72: $\quad$ **end for**
73: **end for**

Result: A set of variables A^c and the coloured modularly composed model $\mathcal{N}(G_{col}) = \{P^{G_{col}}, T^{G_{col}}, F^{G_{col}}, W^{G_{col}}, \psi^{G_{col}}, g^{G_{col}}, f^{G_{col}}, v^{G_{col}}, m_0^{G_{col}}\}$

Special Case 2. The model instantiation using coloured Petri nets allows to differentiate between the intermolecular and intramolecular interaction of one particular component. In general, we define an intermolecular interaction as an interaction among different components or different copies of a particular component, see Figure 3.13(A) and (B) for illustration. If the interacting functional units are identical as well, the scenario described in the last paragraph can be applied. In contrast, we define an intramolecular interaction, as an interaction among the functional units of one and the same copy of a particular component. Here, we want to discuss intermolecular interaction among copies of one particular component and intramolecular interaction of the same copy of one particular component. For both types of interaction, we can further distinguish whether the involved functional units form a complex, and thus have a common interaction state, or not. Since, we did not distinguish component instances so far, all intermolecular interactions among copies of a particular component are unintentionally considered as intramolecular interactions. A potential transition $t \in T^{G}$ involves a set of places ${}^\bullet t \cup t^\bullet$ that represents molecular states of different functional units $U'(c) = \bigcup_{p \in {}^\bullet t \cup t^\bullet} \bigcup_{(u \in \lambda^{u}(\varrho_{p \to s}(p))) \wedge (c \in \lambda^{c}(\varrho_{p \to s}(p)))} u$ of the same component $U'(c) \subseteq U(c)$. The following steps have to be performed to transform an intramolecular interaction into an intermolecular interaction: For each functional unit $u \in U'(c)$, an additional variable a_c^u needs to be defined for the simple colour set w_c' of component c. The arc-expression of each arc connecting the transition t with a place $p \in {}^\bullet t \cup t^\bullet$ representing a single component, where $\varrho_{p \to s}(p) \in S(u)$ and $|\lambda^{c}(s)| = 1$, is set to $f(p,t) = 1`a_c^u$, respectively $f(t,p) = 1`a_c^u$. The arc-expression of the arc connecting the transition t with a place $p \in {}^\bullet t \cup t^\bullet$ representing a complex, where $\varrho_{p \to s}(p) \in S(u)$ and $|\lambda^{c}(s)| >= 1$, is set to $f(p,t) = 1`\bigcup_{c \in k} a_c^u$, respectively $f(t,p) = 1`\bigcup_{c \in k} a_c^u$.

The scenarios explained in the last two paragraphs cannot be automated so far. Additional detailed biological knowledge about the respective molecular events and involved components is necessary, that might not be included in the module annotation. It might be even necessary to modify the arc-expression of neighbouring transitions as well, see Figure 3.13(A) and (B) again for illustration. The observations above only show, how to detect scenarios that need to be treated separately and give some recommended instructions. Of course it might be necessary to make individual adaptations.

Running Example. The result of the model instantiation using coloured Petri nets applied to the composed model of the running example, see Figure 3.11, is shown in Figure 3.12. According to Algorithm 3.1, we defined a maximum and initial number of instances for each component:

- $n_{max,pR} = 10$, $n_{ini,pR} = 1$
- $n_{max,pA} = 10$, $n_{ini,pA} = 5$
- $n_{max,gR} = 1$, $n_{ini,gR} = n_{max,gR}$
- $n_{max,mR} = 3$, $n_{ini,mR} = 0$
- $n_{max,L} = 10$, $n_{ini,L} = 5$

First, we added the component places named $comp\_pR$, $comp\_gR$, $comp\_mR$, $comp\_pA$, and $comp\_L$, see Figure 3.12(2). For each component, we defined a simple colour set

- $cs\_pR := 1 - n_{max,pR}$
  $\hookrightarrow$ assigned to all places representing molecular states of functional units of the receptor protein, except places $pA\_SH2\_\_pR\_Yp$ and $pR\_LBD\_\_L$ representing interaction states, see Table 3.4(A).
- $cs\_pA := 1 - n_{max,pA}$
  $\hookrightarrow$ assigned to all places representing molecular states of functional units of the receptor protein, except places $pA\_SH2\_\_pR\_Yp$ and $pA\_DNA\_BS\_act\_\_gR\_ABS$ representing interaction states, see Table 3.4(B).
- $cs\_gR := 1 - n_{max,gR}$
  $\hookrightarrow$ assigned to all places representing molecular states of functional units of the receptor gene, except place $pA\_DNA\_BS\_act\_\_gR\_ABS$ representing an interaction state, see Table 3.4(C).
- $cs\_mR := 1 - n_{max,mR}$
  $\hookrightarrow$ assigned to all places representing molecular states of functional units of the receptor mRNA, see Table 3.4(D).
- $cs\_L := 1 - n_{max,L}$
  $\hookrightarrow$ assigned to place L.

  as well as a variable bound to the colour set:

- $pR := cs\_pR$
- $pA := cs\_pA$
- $gR := cs\_gR$
- $mR := cs\_mR$
- $L := cs\_L$

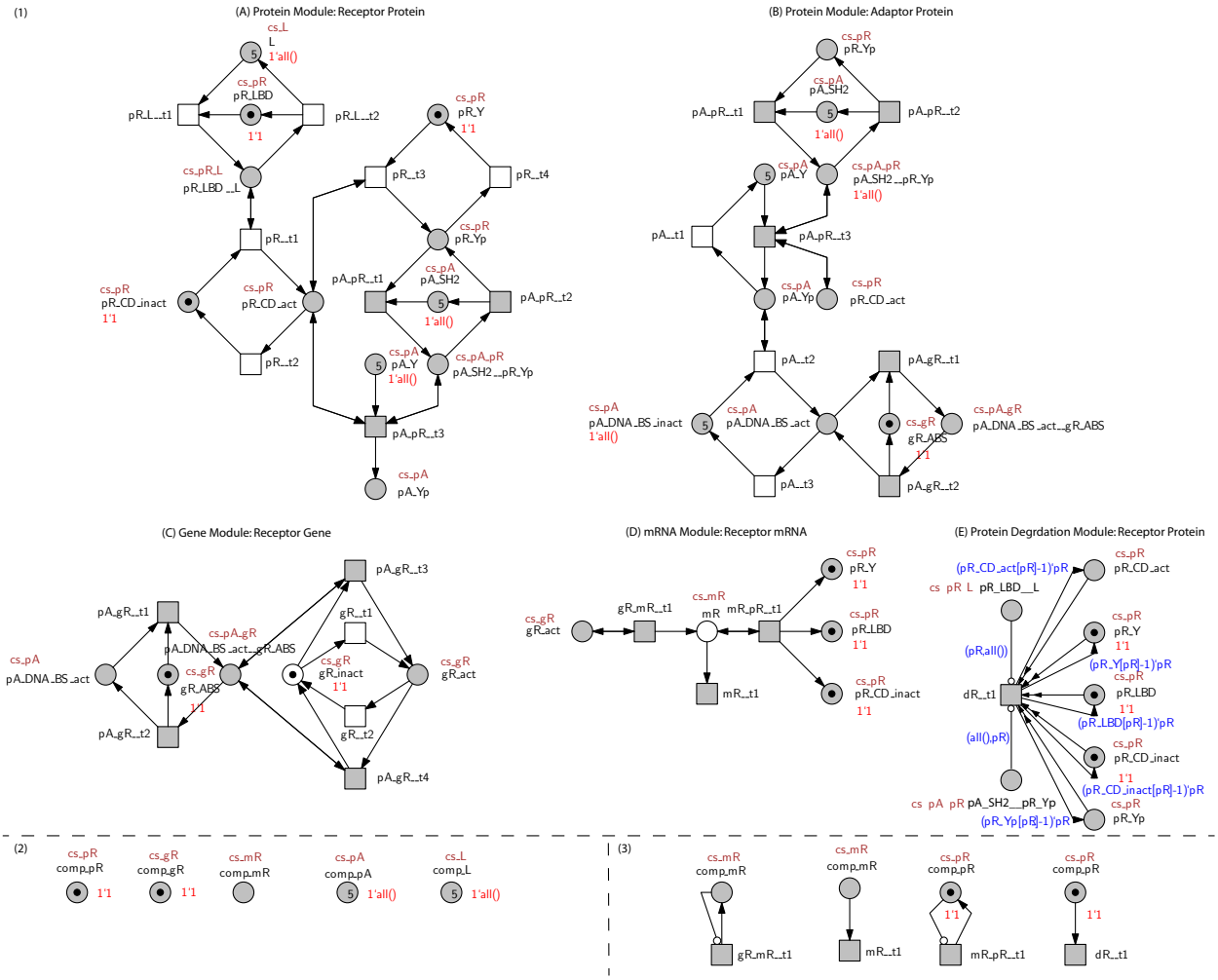


Figure 3.12: Model instantiation of the running example using coloured Petri nets. (1) Coloured Petri net representation of the modules, (2) component place, (3) additional modifications for the synthesis and degradation of the receptor mRNA and protein. (colour set of a place is given maroon letters; marking of a place is given in red letters; arc-expressions are given in blue letters, only non-trivial arc-expression are shown for clearness)

Since we can determine three different complexes (interaction states) given by the places *pR_LBD__L*, *pA_SH2__pR_Yp* and *pA_DNA_BS_act__gR_ABS*, we need to define three compound colour sets of type product:

- $cs\_pR\_L := cs\_pR \times cs\_L$
  $\hookrightarrow$ assigned to place *pR_LBD__L*
- $cs\_pA\_pR := cs\_pA \times cs\_pR$
  $\hookrightarrow$ assigned to place *pA_SH2__pR_Yp*
- $cs\_pA\_gR := cs\_pA \times cs\_gR$
  $\hookrightarrow$ assigned to place *pA_DNA_BS_act__gR_ABS*

The arc-expression of an in-going or out-going arc of a place is set to the variable bound to the colour set of the place.

Transitions describing the synthesis and degradation of the receptor protein and mRNA have to be connected with their corresponding component places, as explained in the previous section, see Figure 3.12(3).

To illustrate the handling of dimerization processes and intramolecular versus intermolecular interaction, we exemplarily introduce the

following assumptions on the molecular mechanism depicted in Figure 3.2:

1. Assumption: The ligand-binding domain of the receptor has to dimerize with a second copy of the receptor protein before binding the ligand, compare Figure 3.13(A) (modifications are shaded in green):
   - add two places *pR_LBD__pR_LBD* and *pR_LBD__pR_LBD__L* to represent the molecular states of the complex between the ligand binding domains of the two receptor protein copies and the complex of the ligand binding domains of two receptor protein copies with the ligand
   - delete place *pR_LBD__L* (the molecular state does not exist anymore)
   - add two transitions representing the binding of the ligand binding domains of two receptor protein copies *pR__t5* and their dissociation *pR__t6*.
   - reconnect the places and transitions in the receptor protein module as shown in Figure 3.13(A) to assign the correct educts and products to the added molecular events
   - define two additional compound colour sets of type product to provide an appropriate colour set for the added molecular states (places):
     - $cs\_pR\_pR := cs\_pR \times cs\_pR$
       $\hookrightarrow$ assigned to place *pR_LBD__pR_LBD*
     - $cs\_pR\_pR\_L := cs\_pR \times cs\_pR \times cs\_L$
       $\hookrightarrow$ assigned to place *pR_LBD__pR_LBD__L*
   - define two additional variables to distinguish between receptor protein copies:
     - $pR1 := cs\_pR$
     - $pR2 := cs\_pR$
   - set arc-expression as shown in Figure 3.13(A) to be consistent with the added colour sets and assumptions
   - set $n_{ini,pR} = 2$ to initially define two copies of the receptor protein (at least two receptor protein copies are needed for the interaction assumed above)
2. Assumption: The phosphorylation of the receptor protein tyrosine happens in *trans* by a second bound receptor protein copy, compare Figure 3.13(A) (modifications are shaded in orange):
   - define two additional variables according to the involved functional units to distinguish between receptor protein copies:
     - $pR\_CD := cs\_pR$
     - $pR\_Y := cs\_pR$
   - set arc-expression as shown in Figure 3.13(A) to be consistent with the added colour sets and assumptions
3. Assumption: The phosphotyrosine of the adaptor protein has to be bound by an SH2 domain of another adaptor protein copy before activating the DNA binding site, compare Figure 3.13(A) (modifications are shaded in yellow):

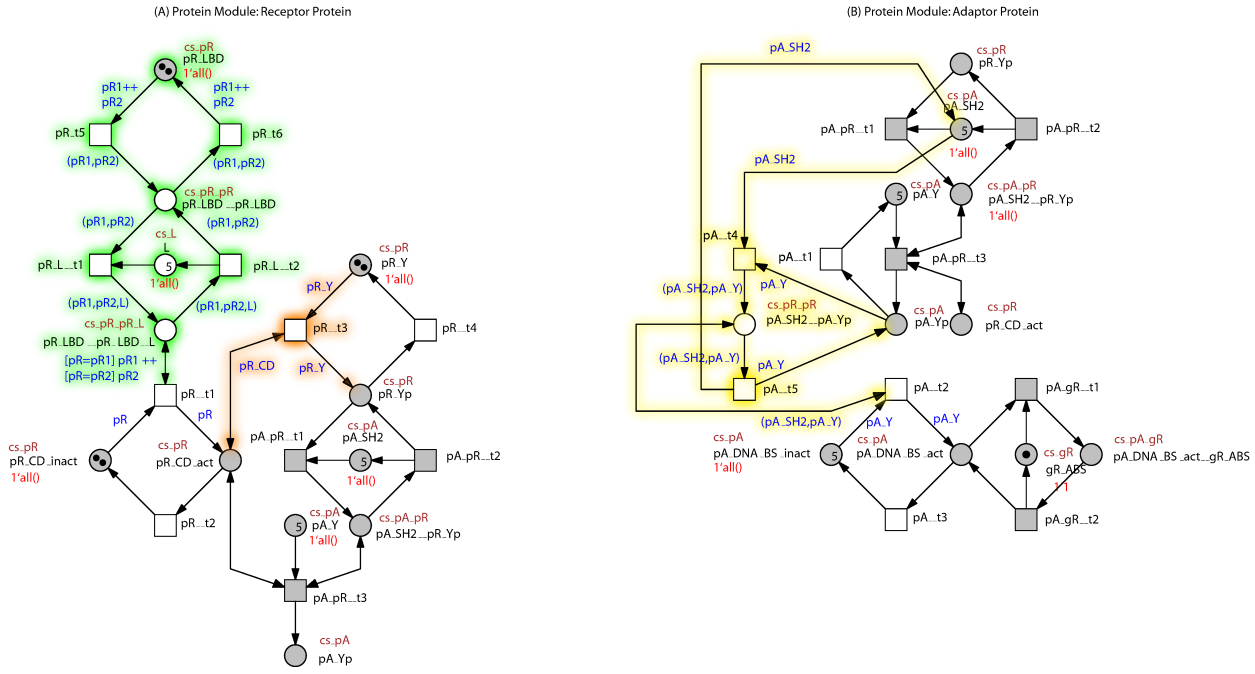


Figure 3.13: Discrimination of component instances involved in homodimerization and intermolecular interactions. The instantiation of the composed model using coloured Petri net transformation, allows distinguishing component instances involved in dimerization processes and intermolecular interaction between instances of the same component. Therefore, in the receptor protein module (A), the dimerization of the ligand binding domain (shaded in green) and the trans-phosphorylation (shaded in orange) of the receptor protein tyrosine have been introduced. In the adaptor protein module (B), the binding of the phosphotyrosine of the adaptor protein to an SH2 domain of another adaptor protein copy has been added to activate the DNA binding site (shaded in yellow).

- add a place $pA\_SH2\_\_pA\_Yp$ representing the molecular state of the complex between the SH2 domain and the phosphotyrosine of the two different adaptor protein copies
- add two transitions representing the binding of the SH2 domain and the phosphotyrosine of the adaptor protein $pA\_\_t4$ and their dissociation $pA\_\_t5$
- disconnect place $pA\_\_Yp$ and transition $pA\_\_t2$, $pA\_\_Yp$ does not trigger the activation of the DNA binding site according to the new assumption above
- reconnect the places and transitions in the receptor protein module as shown in Figure 3.13(B) to assign the correct educts and products to the added molecular events
- define an additional compound colour set of type product to represent the complex of two different copies of the adaptor protein:
  - $cs\_pA\_pA := cs\_pA \times cs\_pA$
    $\hookrightarrow$ assigned to place $pA\_SH2\_\_pA\_Yp$
- define two additional variables according to the involved functional units to distinguish between adaptor protein copies:
  - $pA\_SH2 := cs\_pA$
  - $pA\_Y := cs\_pA$
- set arc-expression as shown in Figure 3.13(B) to allow the interaction of the two different adaptor protein copies

3.3.3 *Algorithmic Model Mutation*

This section requires:

- *Petri nets [82], Section 2.1*
- *Module definition, Section 3.1.1*
- *Module types, Section 3.1.2*
- *Petri net representation of modules, Section 3.1.3*
- *Modular model composition, Section 3.3.1*

Life sciences showed that it is necessary to not only investigate the native state of a system but also alterations due to e.g. genetic differences, which are naturally occurring or have been experimentally introduced. Therefore, the mutation of the nucleotide sequence of genomes is a fundamental method. Experimentally induced mutations may abolish the gene function. Thus, the encoded protein is either missing or completely inactive (gene knock-outs). But mutations might also affect the function of a protein qualitatively or quantitatively by altering the protein structure. Structural alterations might cause a gain of function, a loss of function or potentially affect the substrate specificity. We integrated the idea of mutation studies into the BMKFR by defining algorithms to conveniently mimic different mutation types *in silico* based on the modular modelling concept and model composition. In the BMKFR, gene knock-outs either caused by deletion of the gene or by a loss of function mutation are modelled by the deletion of entire modules, while structural alterations of a gene, mRNA, or protein are mimicked by modifying the Petri net structure of the respective module, see Table 3.9. Using the suggested algorithmic model mutation allows generating sets of alternative models for an extensive analysis of the effects of structural alterations on the performance and behaviour of the model.

Mutation Type	**Biological Effect**	***In Silico* Realisation**
gene deletion	no protein synthesised	delete module
protein structure mutation	loss of function, gain of function, indifferent	modify Petri net structure of the module

Table 3.9: Mutation types.

The large number of alternative models generated by the algorithmic model mutation demands powerful and sophisticated high-throughput tools for the efficient analysis of the models' behaviour. A promising approach to explore a complex and high-dimensional solution space relies on machine learning [123]. It yields temporal logic classifications of the behaviours generated by a huge number of models in high-throughput simulations. Applying this machine learning approach to models obtained by algorithmic module mutation produces clusters and classifies them according to their behaviour by temporal logic properties, and thus reveals characteristic phenotypes of mutations that had been introduced *in silico*. The systematic mutation of modules together with high-throughput analysis frameworks might reveal fundamental insights by investigating and comparing alternative models to identify components that are crucial determinants of the often complicated and unintuitive behaviour of a system. We will now provide algorithms for *in silico* introducing different mutation types into modularly composed models.

Knock-out of Genes: Deletion of Proteins

In our modular approach, the *in silico* equivalent of a gene knock-out is to eliminate modules from the unmodified modularly composed

model $\mathcal{N}(G)$. In the case of a single knock-out, all modules with the same main component c_0 have to be deleted. In some cases, single gene knock-outs might only insufficiently block a specific functionality due to redundancy in the gene function. Thus, double or even multiple gene knock-outs have to be performed to address genes complementing each other for a single cellular function. Algorithm 3.2 encodes the systematic single and double knock-out of modules in the unmodified modularly composed model $\mathcal{N}(G)$ based on their main components.

Algorithm 3.2: Module knock-out.

Require: Set of modules G.
1: $visitedSKO = \emptyset$
2: $ComposeModel(\mathcal{N}(G))$; ▷ unmodified model
3: **for** $i := 1$ **to** $|G|$ **do**
4: **if** $M_i \notin visitedModulesSKO$ **then**
5: $G_{SKO} = G \setminus \{M_{g,c_{0,i}}, M_{m,c_{0,i}}, M_{p,c_{0,i}}, M_{d,c_{0,i}}\}$;
6: $ComposeModel(\mathcal{N}(G_{SKO}))$; ▷ single module knock-out
7: $visitedSKO = visitedSKO \cup \{M_{g,c_{0,i}}, M_{m,c_{0,i}}, M_{p,c_{0,i}}, M_{d,c_{0,i}}\}$;
8: $visitedDKO = \emptyset$,
9: **for** $j := i + 1$ **to** $|G_{SKO}|$ **do**
10: **if** $M_i \notin visitedDKO$ **then**
11: $G_{DKO} = G_{SKO} \setminus \{M_{g,c_{0,j}}, M_{m,c_{0,j}}, M_{p,c_{0,j}}, M_{d,c_{0,j}}\}$;
12: $ComposeModel(\mathcal{N}(G_{DKO}))$; ▷ double module knock-out
13: $visitedDKO = visitedDKO \cup \{M_{g,c_{0,j}}, M_{m,c_{0,j}}, M_{p,c_{0,j}}, M_{d,c_{0,j}}\}$;
14: **end if**
15: **end for**
16: **end if**
17: **end for**

Modification of Protein Structure and Function by Mutation

Point mutations (single nucleotide exchanges), insertions or deletions, can affect the protein structure and hence alter function and activity of a genetic component qualitatively or quantitatively. The effect of point mutations can be a gradual gain or loss of function, which might be evoked for instance by the deletion of structural units of the protein, exchange of amino acids, freezing of a specific conformational state of a protein, or the loss of interactions with other proteins or small molecules.

Since a protein module represents the functional units of the structure of a protein and hence determines its functionality within the model, we can mimic structural alterations of a protein *in silico* by modifying the Petri net structure of the corresponding module. This approach can be applied to all other module types as well. We propose two algorithms to alter modules of any type systematically in the unmodified modularly composed model $\mathcal{N}(G)$ to cover the most prevalent and functionally relevant mutations, see Algorithm 3.3 and 3.4.

Algorithm 3.3 systematically deletes one or two transitions at a time from the unmodified modularly composed model $\mathcal{N}(G)$. By deleting transitions, the affected functional units can only adopt a subset of molecular states, which might influence the functionality of the respective component. The functionality might be increased, decreased or remain unchanged by the mutation.

In addition, Algorithm 3.4 restricts the set of transitions by the

Algorithm 3.3: Transition knock-out.

Require: Set of modules G

1: $ComposeModel(\mathcal{N}(G))$ ▷ unmodified model
2: **for all** $t \in T^{G_{pr}}$ **do**
3: $\mathcal{N}(G_{SM}) = \mathcal{N}(G)$;
4: $T^{G_{SM}} = T^{G} \setminus t$;
5: $H^{G_{SM}} = H^{G} \setminus h(t)$;
6: $f^{G_{SM}}(P^{G_{SM}}, t) = 0$;
7: $f^{G_{SM}}(t, P^{G_{SM}}) = 0$;
8: $ComposeModel(\mathcal{N}(G_{SM}))$ ▷ single transition knock-out
9: **for all** $t' \in T^{G_{pr}} \setminus t$ **do**
10: $\mathcal{N}(G_{DM}) = \mathcal{N}(G_{SM})$;
11: $T^{G_{DM}} = T^{G_{SM}} \setminus t'$;
12: $H^{G_{DM}} = H^{G_{SM}} \setminus h'$;
13: $f^{G_{DM}}(P^{G_{DM}}, t') = 0$;
14: $f^{G_{DM}}(t', P^{G_{DM}}) = 0$;
15: $ComposeModel(\mathcal{N}(G_{DM}))$ ▷ double transition knock-out
16: **end for**
17: **end for**

Algorithm 3.4: Annotated transition knock-out.

Require: Set of modules G, $A(t) = \{a_1, \dots a_{n_{A_t}}\}$ for each transition $t \in T^G$, set of annotations $A_{MUT} = \{a_1^{MUT}, \dots a_{n_{A_{MUT}}}^{MUT}\}$ to filter transitions

1: $ComposeModel(\mathcal{N}(G))$ ▷ unmodified model
2: $\mathcal{N}(G_{AM}) = \mathcal{N}(G)$;
3: **for all** $t \in T^{G_{pr}}$ **do**
4: **if** $A(t) \cap A_{MUT} \neq \emptyset$ **then**
5: $\mathcal{N}(G_{SM}) = \mathcal{N}(G)$;
6: $T^{G_{SM}} = T^{G} \setminus t$;
7: $H^{G_{SM}} = H^{G} \setminus h(t)$;
8: $f^{G_{SM}}(P^{G_{SM}}, t) = 0$;
9: $f^{G_{SM}}(t, P^{G_{SM}}) = 0$;
10: $ComposeModel(\mathcal{N}(G_{SM}))$ ▷ single transition knock-out
11: **for all** $t' \in T^{G_{pr}} \setminus t$ **do**
12: **if** $A(t') \cap A_{MUT} \neq \emptyset$ **then**
13: $\mathcal{N}(G_{DM}) = \mathcal{N}(G_{SM})$;
14: $T^{G_{DM}} = T^{G_{SM}} \setminus t'$;
15: $H^{G_{DM}} = H^{G_{SM}} \setminus h'$;
16: $f^{G_{DM}}(P^{G_{DM}}, t') = 0$;
17: $f^{G_{DM}}(t', P^{G_{DM}}) = 0$;
18: $ComposeModel(\mathcal{N}(G_{DM}))$ ▷ double transition knock-out
19: **end if**
20: **end for**
21: $T^{G_{AM}} = T^{G} \setminus t$;
22: $H^{G_{AM}} = H^{G} \setminus h(t)$;
23: $f^{G_{AM}}(P^{G_{AM}}, t) = 0$;
24: $f^{G_{AM}}(t, P^{G_{AM}}) = 0$;
25: **end if**
26: **end for**
27: $ComposeModel(\mathcal{N}(G_{AM}))$ ▷ all transition knock-out

use of cross-references mapped to the transitions by BMKML model annotation. Each transition $t \in T^G$ is annotated with a set of cross-references $A(t) = \{a_1, \dots a_{n_{A_t}}\}$, where $n_{A_t} \geq 1$. The set $A_{MUT} = \{a_1^{MUT}, \dots a_{n_{A_{MUT}}}^{MUT}\}$ defines the cross-references, which will be applied to filter transitions from T^G. A transition $t \in T^G$ is deleted if $A(t) \cap A_{MUT} \neq \emptyset$. One, two or all affected transitions can be deleted from the unmodified modularly composed model at a time.

Running Example. Here, we apply the model mutation algorithms to the running example, see Figure 3.11 for the unmodified modularly

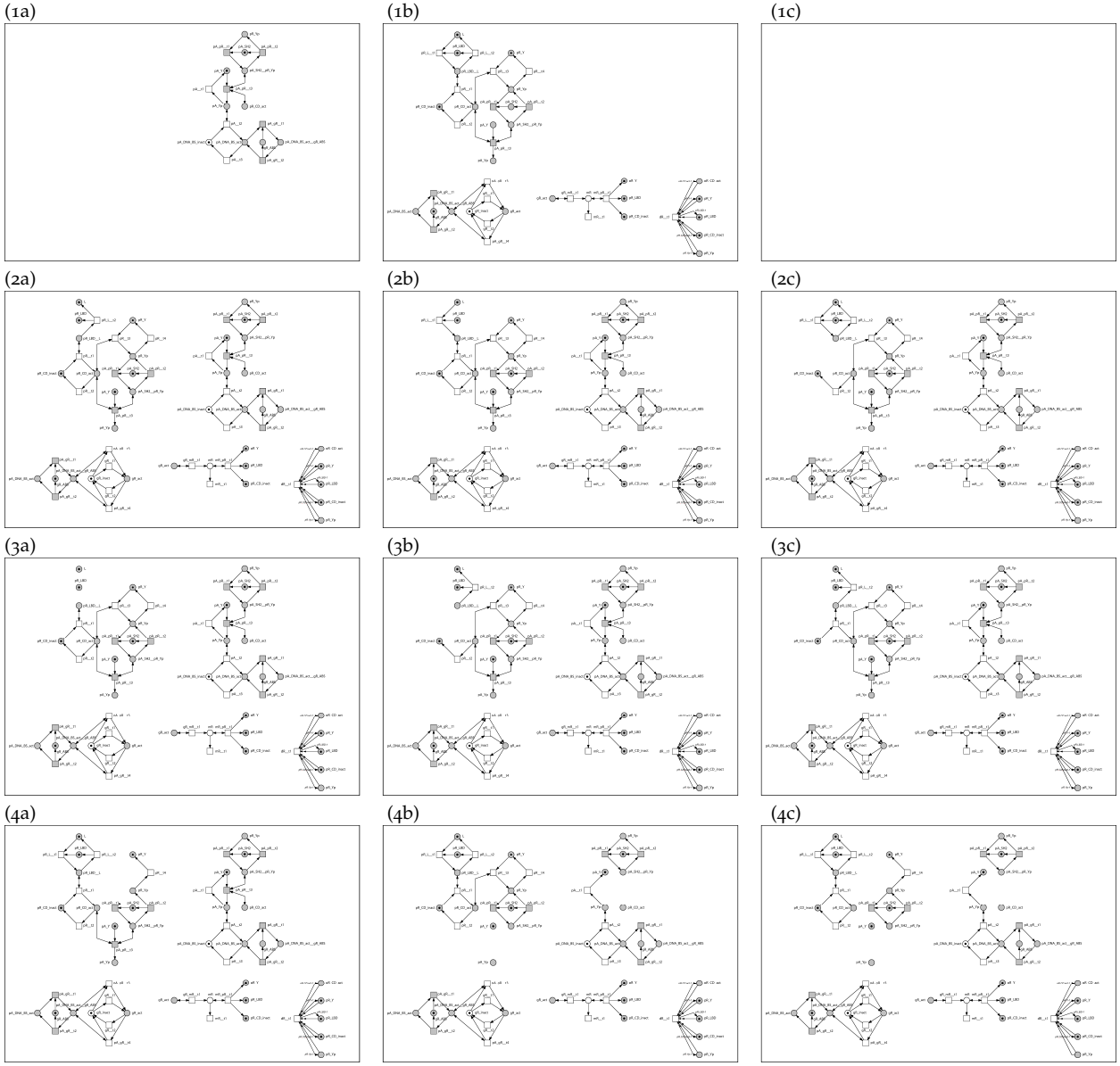


Figure 3.14: Model mutation applied to the running example. (1) Module knock-out: (1a) single knock-out of the receptor protein, (1b) single kock-out of the adaptor protein, (1c) double knock-out of receptor and adaptor protein; (2) Single transition knock-out (exemplified, not complete); (3) Double transition knock-out (exemplified, not complete); (4) Annotated transition knock-out of transitions representing the phosphorylation of tyrosine residues.

composed model. The set of alternative models produced by the model mutation algorithms can be submitted to a high-throughput analysis workflow, to cluster and identify models with desired behaviour.

Applying the module knock-out, Algorithm 3.2, results into two alternative models for the single module knock-out with $G_{SKO,1} = \{M_{pA}\}$ and $G_{SKO,2} = \{M_{gR}, M_{mR}, M_{pR}, M_{dR}\}$ and an empty model for the double module knock-out $G_{DKO,1} = \emptyset$, see Figure 3.14(1a)-(1c).

According to the number of transitions in the unmodified modularly composed model $\mathcal{N}(G)$, the single knock-out of transitions, Algorithm 3.4, results in 25 alternative models, Figure 3.14(2a)-(2c) provides some examples. The double knock-out of transitions yields

300 additional alternative models, see Figure 3.14(3a)-(3c) for some examples.

In the case of annotated transition knock-out, Algorithm 3.3, we assume that all transitions are mapped to specific GO identifiers [9] [79] according to their corresponding molecular events. For example transition pR_t3 and pA_pR_t4 represent the phosphorylation of a tyrosine residue, which corresponds to the identifier $GO : 0018108$. In this example only transitions with the annotation $GO : 0018108$ should be knocked out. Accordingly, the algorithm generates two alternative models for the single transition knock-out. Only one of the annotated transitions pR_t3 and pA_pR_t4 is knocked out in each model. Another alternative model is generated for the double transition knock-out, where both annotated transitions pR_t3 and pA_pR_t4 are knocked out, see Figure 3.14(4a)-(4c). In this case, the double transition knock-out of the annotated transitions is equivalent to the knock-out of all transitions annotated with $GO : 0018108$.

[9] GO - Gene Ontology [79]

3.3.4 *Spatial Model Transformation*

This section requires:

- *Petri nets [82], Section 2.1*
- *Coloured Petri nets [122], Section 2.3*
- *Module definition, Section 3.1.1*
- *Module types, Section 3.1.2*
- *Petri net representation of modules, Section 3.1.3*
- *Modular model composition, Section 3.3.1*

Next to the mechanistic and temporal aspects of a biomolecular system, spatial aspects are also essential, like the distribution of the involved components, the position of cells and components, compartmentalization, cell size, and shape. Those aspects determine the correct functioning of a biomolecular system.

Integrating spatial information into models and allowing biomolecular components to move in a defined grid, in particular in the case of existing models, is a challenging task. Regarding biochemical systems represented as Petri nets, coloured Petri nets have been applied to incorporate spatial aspects and the movement in space as shown in [128]. Therefore, space is discretized into a grid of one, two or three dimensions. A single place encodes the position on the grid. This approach works well if the components in the model do not need to be distinguished on each grid position, e.g. as in the case of the diffusion of a component, as presented in [128]. This concept can even be applied to more complex reaction-diffusion systems, as shown in [135]. More examples on how to use coloured Petri nets to incorporate spatial aspects of biomolecular systems can be found in [127, 132].

All models of the examples mentioned above have in common that they model space by discretization into a grid and having one subnet (ranging from a single place to a complex network) per grid position. This approach is handy for models with components having no internal behaviour or state, and components that do not need to be distinguished as well. Otherwise, if components have an internal network that needs to move on the grid, this approach is not suitable due to several arising modelling and simulation issues. Using this method would drastically increase the size of the unfolded model, which impacts the analysis and simulation of the model and may lead to inconvenient run times.

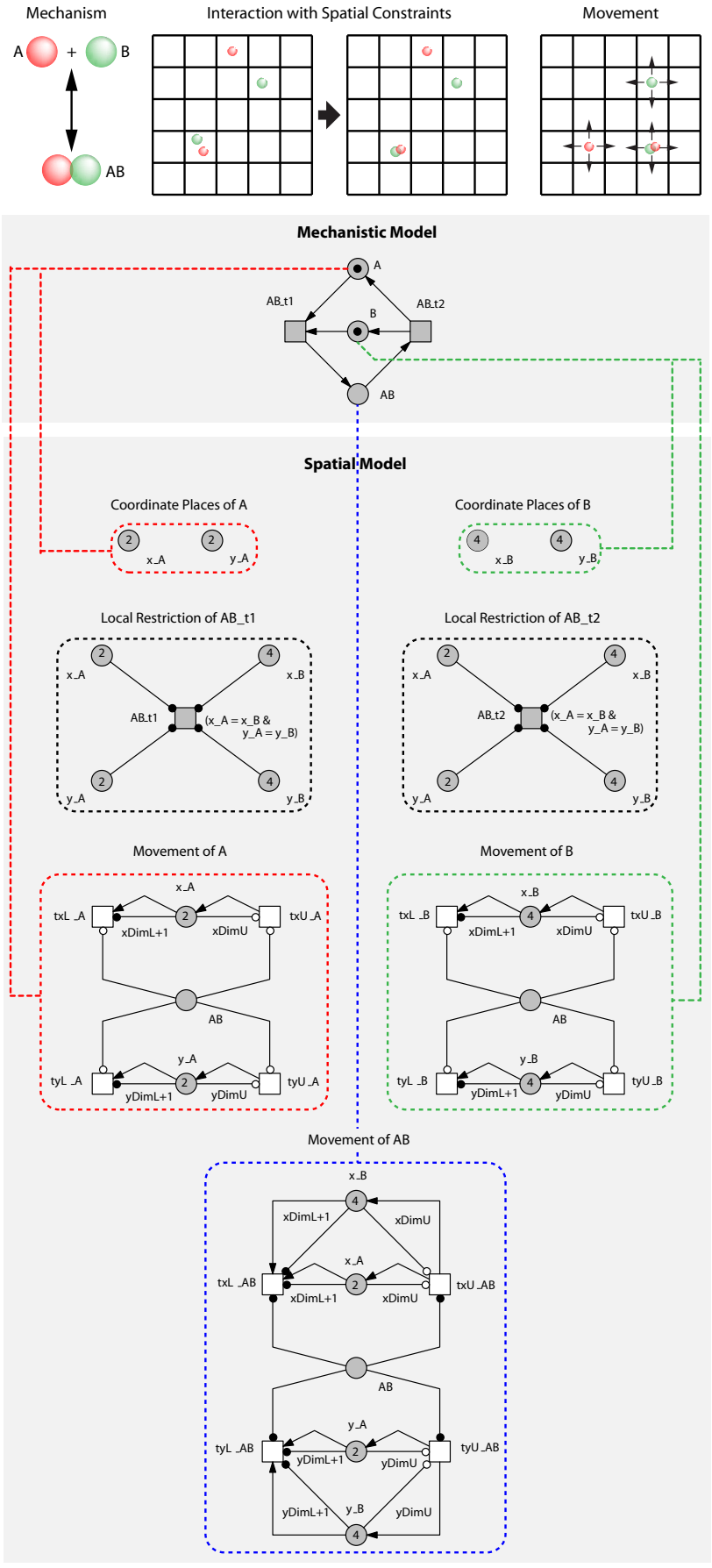


Figure 3.15: Representing biomolecular mechanisms and movement in one coherent model is a challenging task. The use of Petri nets allows describing biomolecular mechanisms next to the movement on a defined grid. The resulting model can be divided into a mechanistic model and a spatial model. The mechanistic model represents the biomolecular mechanisms involved in the modelled system, which can later be replaced by a modularly composed model. The spatial model encodes the positions of the involved component, the spatial constraints applied to the interactions, as well as the movement of components, respectively complexes formed due to interactions. Interactions can only occur, if components fulfil a defined neighbourhood condition, e.g. stay at the same position. Encoding multiple copies of components and their individual movement requires the use of coloured Petri nets to instantiate the model. In the Example, the mechanistic model describes the binding of A and B to form complex AB, and its dissociation. The spatial model provides places encoding the (x,y)-coordinates of A and B, the marking of those places reflects the position on the grid, which is defined by the parameters $(xDimL, xDimU)$ and $(yDimL, yDimU)$ (lower/upper bound at the x/y-axis). Also, the spatial model encodes spatial constraints to locally restrict interactions among A and B. Therefore, the transitions AB_t1 and AB_t2 are connected with the coordinates places by read arcs. The firing rates are multiplied by a Boolean expression evaluating whether A and B are at the same position. Only if A and B stay at the same position, the interaction can be executed in the model. In the spatial model, A and B can either move as single entities or as complex AB. Thus, they can either move towards the lower bound of an axis $(t\{x,y\}L\_\{A,B,AB\})$ or the upper bound of an axis $(t\{x,y\}U\_\{A,B,AB\})$. The marking of the coordinate place must always be greater than $xDimL/yDimL$, to move towards the lower bound of an axis. This is checked by the read arcs with the arc-weight $xDimL+1/yDimL+1$. Also, the marking of the coordinate place must always be less equal than $xDimU/yDimU$, to move towards the upper bound of an axis. This is checked by the inhibitory arcs with the arc-weight $xDimU/yDimU$. In addition, A and B are only allowed to move as a single entity if the place AB is empty. This is checked by the inhibitory arcs connecting the place AB with transitions $t\{x,y\}L\_\{A,B\}$ and $t\{x,y\}U\_\{A,B,\}$. In contrast, to move A and B as a complex, place AB must not empty. This is checked by the read arcs connecting place AB with transitions $t\{x,y\}U\_AB/\ tyU\_AB$.

Since in our modularization approach components are characterised by complex subnets with an internal behaviour and single copies of a component need to be distinguished, the approach above is not practical. The key of our concept is to represent space by adding coordinate places for each component in the modular composed model. The marking of the coordinate places of a component indicates its position on a defined grid. The size of the grid is encoded by the minimal and maximal number of tokens, which are allowed to stay on a coordinate place. Depending on the dimensionality of the grid, one, two, or three coordinate places have to be added per component. In contrast, in the previous approach each position of each component on a grid was defined by one place. Assuming a model with 3 components and a 2-dimensional grid of size 5×5, in our approach we need 6 places to encode the grid of the components (2 coordinate places per component), where in the previous approach we need 75 places ($5 \times 5 = 25$ places per component). Based on our idea, the modular approach defined in the BMKFR allows to extend modularly composed models with spatial aspects, to represent the local positioning and movement of components, and to constrain interactions based on the local positioning of components [139]. In Figure 3.15, we illustrate our approach using a very simple example. The following steps transform a modularly composed model into a spatial model:

1. Components have to be equipped with individual spatial attributes by adding coordinate places representing their current position on a defined grid.
2. The interaction among the components have to be constrained. Therefore, the firing rate of a transition representing an interaction event is multiplied by a Boolean function to check, whether the interacting components fulfil a pre-defined neighbourhood conditions, e.g. components must have the same coordinate values.
3. The movement of a component has to be realised by adding additional transitions, which remove and add tokens to the coordinate places. Components are only allowed to move as single entities if they are not interacting with any other component, e.g. all places representing interaction states must be empty. To move components forming a complex, all places representing corresponding interaction states involved in the complex formation must be marked, where places representing other interaction states must be empty. The localisation of components in a complex needs to be changed simultaneously.
4. The modified model has to be transformed into a coloured Petri net according to Algorithm 3.1 in Section 3.3.2 to represent multiple instances of each component.
5. Additional compartments according to the compartmentalization and structural composition of the cell (e.g. pools, membrane, etc.) can be introduced into the grid. For each component, it must be specified, in which compartments it is allowed to stay. The firing-

rate of a transition describing the movement of a component or complex has to be multiplied by a Boolean expression; that evaluates whether the respective component stays within its assigned compartments or not. Only if the Boolean expression is evaluated to true, the movement can be executed.

Below, we formally describe the five steps above of the spatial model transformation in more detail. Each step is followed by the running example introducing the previous formalization into the model.

Explicit Encoding of Local Positions.

The explicit encoding of local positions of components in the composed model $\mathcal{N}(G)$ is given in detail in Algorithm 3.5.

For each component $c \in C^G$ its local position is encoded by d coordinate places given by $P^c_{ax} = \{p^c_{ax,1}, \dots p^c_{ax,d}\}$, where $d \geq 1$ defines the number of axes of the assumed grid. For each coordinate place $p^c_{ax,i} \in P^c_{ax}$, a lower $m_L(p^c_{ax,i})$ and upper bound $m_U(p^c_{ax,i})$ define the size of the component specific grid, such that $m_{L/U}(p^c_{ax,i}) > 0$ and $m_L(p^c_{ax,i}) < m_U(p^c_{ax,i})$. The initial position of the component c is defined by a tuple of coordinates $o_c = (x^c_1, \dots, x^c_3)$, where $m_L(p^c_{ax,i}) \leq x^c_i \leq m_U(p^c_{ax,i})$. The initial marking of a coordinate place $p^c_{ax,i}$ of component c is set to $m_0(p^c_{ax,i}) = x^c_i$.

The entire set of coordinate place for all component in C^G is defined by $P_{ax} = \bigcup_{c \in C^G} P^c_{ax}$. Therefore, the set of places P^G is now defined as $P^G = P^G \cup P_{ax}$.

Algorithm 3.5: Explicit encoding of local positions.

Require: $\mathcal{N}(G) = \{P^G, T^G, F^G, f^G, v^G, m^G_0\}$, dimensionality d, set of initial coordinates for each component $O^G = \bigcup_{c \in C^G} o_c$ with $o_c = (x^c_1, \dots, x^c_d)$

1: $Pax = \emptyset$
2: $P^G_{org} = P^G$

3: **for all** $c \in C^G$ **do**
4: **for** $i = 1$ **to** d **do**
5: $P^c_{ax} = P^c_{ax} \cup p^c_{ax,i}$ ▷ add coordinate places
6: $m^G_0(p^c_{ax,i}) = x^c_i$ ▷ initialize marking with coordinate value
7: **end for**
8: $P_{ax} = P_{ax} \cup P^c_{ax}$
9: **end for**
10: $P^G = P^G \cup P_{ax}$

Result: $\mathcal{N}(G_{step1}) = \{P^G, T^G, F^G, f^G, v^G, m^G_0\}$

Running Example. To transform the modularly composed model of the running example, see Figure 3.11, we assume a two-dimensional grid of the same size for each all component:

- Lower Bound:
  - $xDimL\_pR = xDimL\_gR = xDimL\_mR = xDimL\_pA = xDimL\_L = 1$
  - $yDimL\_pR = yDimL\_gR = yDimL\_mR = yDimL\_pA = yDimL\_L = 1$

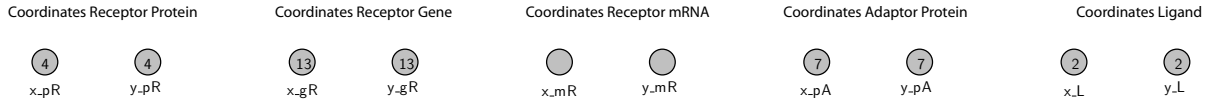


Figure 3.16: Definition of coordinate places. For each component coordinate places have to be defined, e.g. places x_pR and y_pR representing the x- and y-coordinates of the receptor protein. The initial marking of the coordinate places is set according to the initial position assumed for each component, see text. *Note:* logical nodes have been used for a decluttered representation.

- Upper Bound:
  - $xDimU\_pR = xDimU\_gR = xDimU\_mR = xDimU\_pA = xDimU\_L = 20$
  - $yDimU\_pR = yDimU\_gR = yDimU\_mR = yDimU\_pA = yDimU\_L = 20$

For each component in the composed model, two coordinate places are added, representing the x- and y-coordinates in a Cartesian coordinate system, see Figure 3.16(1):

- receptor protein $\rightarrow x\_pR, y\_pR$ with $m_0(x\_pR) = m_0(y\_pR) = 4$
- receptor gene $\rightarrow x\_gR, y\_gR$ with $m_0(x\_gR) = m_0(y\_gR) = 13$
- receptor mRNA $\rightarrow x\_mR, y\_mR$ with $m_0(x\_mR) = m_0(y\_mR) = 0$ (receptor mRNA does initially not exist)
- adaptor protein $\rightarrow x\_pA, y\_pA$ with $m_0(x\_pA) = m_0(y\_pA) = 7$
- ligand $\rightarrow x\_L, y\_L$ with $m_0(x\_L) = m_0(y\_L) = 2$

Local Constraint of Interactions.

The local constrains of interactions in the composed model $\mathcal{N}(G)$ is given in detail in Algorithm 3.6.

To constrain the executability of each transition $t \in T^G$ representing an interaction among components, $\varrho_{t\rightarrow e}(t) \in \mathcal{E}^G_{IA}$, the firing rate $h(t)$ must be multiplied by a Boolean function $b(t)$ describing a defined neighbourhood relation: $h'(t) = b(t) * h(t)$. If the neighbourhood relation claims that the distance between interacting components must be zero, $b(t)$ is defined as:

$$b(t) = \begin{cases} 1, & l(t) = 0 \\ 0, & l(t) \neq 0 \end{cases} \quad \text{with}$$

$$l(t) = \sum_{i=1}^{|\lambda^c(\varrho_{p\rightarrow s}(\bullet t \cup t\bullet))|-1} \sum_{j=i+1}^{|\lambda^c(\varrho_{p\rightarrow s}(\bullet t \cup t\bullet))|} \sum_{k=1}^{d} (m(p^{c_i}_{ax,k}) - m(p^{c_j}_{ax,k}))^2.$$

In addition, read arcs, connecting transition t and the coordinate places of each participating component $c \in \lambda^c(\varrho_{p\rightarrow s}(\bullet t \cup t\bullet)$, have to be added, such that $f_{RA}(\bigcup_{c\in\lambda^c(\varrho_{p\rightarrow s}(\bullet t\cup t\bullet))} P^c_{ax}, t) = 1$.

Algorithm 3.6: Local restriction of interactions.

Require: $\mathcal{N}(G_{step1}) = \{P^G, T^G, F^G, f^G, v^G, m^G_0\}$ with $P^G = P^G_{org} \cup P_{ax}$, a set of neighbourhood functions $B = \{b_1, \ldots, b_{|\mathcal{E}^G_{IA}|}\}$ (Boolean expressions)

1: $T^G_{IA} = \varrho^{-1}_{t\rightarrow e}(\mathcal{E}^G_{IA})$
2: **for all** $t \in T^G_{IA}$ **do**
3: **for all** $c \in \lambda^c(\varrho_{p\rightarrow s}(\bullet t \cup t\bullet))$ **do**
4: **for** $i = 1$ **to** d **do**
5: $f^G_{RA}(t, p^c_{ax,i}) = 1$ ▷ connect transitions with coordinate places
6: **end for**
7: **end for**
8: $h^G(t) = h^G(t) * b_i$ ▷ multiply firing rate with neighbourhood function
9: **end for**

Result: $\mathcal{N}(G_{step3}) = \{P^G, T^G, F^G, f^G, v^G, m^G_0\}$

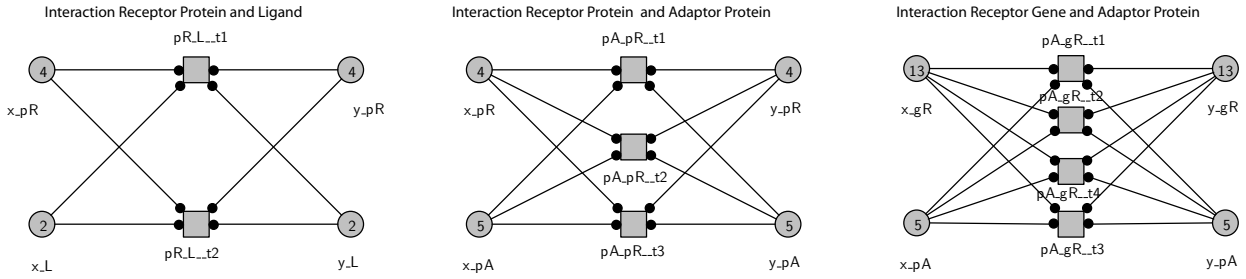


Figure 3.17: Local constraint of interaction in the running example. Coordinate places are linked to transitions representing interactions via read arcs, e.g. coordinate places of the receptor protein (x_pR and y_pR) and ligand (x_L and y_L) are linked to transitions $pR\_L\_\_t1$ and $pR\_L\_\_t2$ representing the interaction of the receptor protein and the ligand. The firing rates of those transitions are multiplied by a Boolean function to evaluate whether the involved components stay one the same position. Thus, interactions can only be executed if the coordinate places of the involved components have identical markings. *Note:* logical nodes have been used for a decluttered representation.

Running Example. The coordinate places of all components involved in the interaction are added via read arcs to each transition representing an interaction in Figure 3.11, see Figure 3.17:

- $pR\_L\_\_t1$, $pR\_L\_\_t2 \rightarrow x\_pR$, y_pR, x_L, y_L
- $pA\_pR\_\_t1$, $pA\_pR\_\_t2$, $pA\_pR\_\_t3 \rightarrow x\_pR$, y_pR, x_pA, y_pA
- $pA\_gR\_\_t1$, $pA\_gR\_\_t2$, $pA\_pg\_\_t3$, $pA\_gR\_\_t4$
  $\rightarrow x\_pA$, y_pA, x_gR, y_gR

The firing rates of these transitions are multiplied by a Boolean function evaluating the distance between the interacting components:

- $b(pR\_L\_\_t1) = b(pR\_L\_\_t2) = \{x\_pR = x\_L \;\&\; y\_pR = y\_L\}$
- $b(pA\_pR\_\_t1) = b(pA\_pR\_\_t2) = b(pA\_pR\_\_t3) =$
  $\{x\_pA = x\_pR \;\&\; y\_pA = y\_pR\}$
- $b(pA\_gR\_\_t1) = b(pA\_gR\_\_t2) = b(pA\_gR\_\_t3) = b(pA\_gR\_\_t3) =$
  $\{x\_pA = x\_gR \;\&\; y\_pA = y\_gR\}$

Explicit Encoding of Local Position Changes.

To encode the position changes (movement) of a component $c \in C^G$, three scenarios have to be considered dependent on the state of interaction:

1. components with no active interaction, which move as single entities, e.g. all places representing an interaction state of component c have to be empty;
2. components with a single active interaction site, e.g. all places representing one particular complex (might involve one or more interaction states) of component c have to be marked, places representing other complexes of component c have to be empty; and
3. components with multiple active interaction sites, e.g. all places representing a subset of possible complexes of component c have to be marked, places representing other complexes of component c have to be empty.

For each case, we need to ensure that the corresponding molecular conditions are fulfilled. Thus, components can only move as a single entity if they have no active interaction state, where components forming a complex are only allowed to move simultaneously if the corresponding interaction states are active. Below, we describe how to proceed in each case.

1. Local position change of individual components, see Algorithm 3.7: For each component $c \in C^G$ and each coordinate place $p^c_{ax,i} \in P^c_{ax}$ two transitions $t^c_{i,L}$ and $t^c_{i,U}$ are needed to incrementally decrease or increase the amount of tokens. Thus, the movement of component c is described by set of transitions $T^c_{move} = T^c_{move,L} \cup T^c_{move,U}$, where $T^c_{move,L} = \{t^c_{1,L}, \ldots, t^c_{d,L}\}$ and $T^c_{move,U} = \{t^c_{1,U}, \ldots, t^c_{d,U}\}$. The transition $t^c_{j,L}$ with the firing rate $h(t^c_{j,L})$ subtracts tokens from the coordinate place $p^c_{ax,i}$ till the lower bound of the grid is reached, $m(p^c_{ax,i}) = m_L(p^c_{ax,i})$. Accordingly, the following arcs have to be introduced $f^G_{SA}(p^c_{ax,i}, t^c_{i,L}) = 1$ and $f^G_{RA}(p^c_{ax,i}, t^c_{i,L}) = m_L(p^c_{ax,i}) + 1$. The transition $t^c_{i,U}$ with the firing rate $h(t^c_{j,U})$ adds tokens to the coordinate place $p^c_{ax,i}$ till the upper bound of the grid is reached, $m(p^c_{ax,i}) = m_U(p^c_{ax,i})$. Consequently, the following arcs have to be introduced $f^G_{SA}(t^c_{i,U}, p^c_{ax,i}) = 1$ and $f^G_{IA}(p^c_{ax,i}, t^c_{i,U}) = m_U(p^c_{ax,i})$. The total set of transition describing the movement of components is given by $T_{move1} = \bigcup_{c \in C^G} T^c_{move}$. Therefore, the set of places T^G is now defined as $T^G = T^G \cup T_{move1}$.
   Each component $c \in C^G$ can only move as a single entity if all places representing an interaction state of component c are empty, $m_0(P^c_{IS}) = 0$ with $P^c_{IS} = \{p \mid c \in \lambda^c(\varrho_{p \to s}(p)) \wedge |\lambda^c(\varrho_{p \to s}(p))| > 1\}$, $P^c_{IS} \subseteq P^G$. Therefore, an additional inhibitory arc for each transition $t \in T^c_{move}$ and each place $p \in P^c_{IS}$ has to be introduced $f^G_{IA}(p, t) = 1$ have to introduced.
   In addition, we have to introduce component places, as in Algorithm 3.1, to check whether a component exist before moving it. For each component $c \in C^G$, we introduce a component place p_c, the set of all component places is given by $P_{comp} = \bigcup_{c \in C^G} p_c$. There-

Algorithm 3.7: Local position change of components as single entities.

Require: $\mathcal{N}(G_{step2}) = \{P^G, T^G, F^G, f^G, v^G, m^G_0\}$ with $P^G = P^G_{org} \cup P_{ax}$

$T^c_{move} = \emptyset$
$T_{move1} = \emptyset$
$P_{comp} = \emptyset$
for all $c \in C^G$ **do**
  $P_c = P_c \cup p_c$
  for $i = 1$ **to** d **do**
    $T^c_{move} = T^c_{move} \cup \{t^c_{i,L}, t^c_{i,U}\}$ ▷ define transitions for movement
    $f^G_{SA}(p^c_{ax,i}, t^c_{i,L}) = 1$
    $f^G_{RA}(p^c_{ax,i}, t^c_{i,L}) = m_L(p^c_{ax,i}) + 1$
    $f^G_{SA}(t^c_{i,U}, p^c_{ax,i}) = 1$
    $f^G_{IA}(p^c_{ax,i}, t^c_{i,L}) = m_U(p^c_{ax,i})$
    ▷ connect coordinate places and test for lower/upper bound
    for all $p \in P^G_{org}$ **do**
      if $c \in \lambda^c(\varrho_{p \to s}(p))$ **&** $|\lambda^c(\varrho_{p \to s}(p))| > 1$ **then**
        $f^G_{IA}(p, t^c_{i,L}) = 1$
        $f^G_{IA}(p, t^c_{i,U}) = 1$
        ▷ test if places representing interactions of the component are empty
      end if
    end for
  end for
  $T_{move1} = T_{move1} \cup T^c_{move}$
end for
$P^G = P^G \cup P_{comp}$
$T^G = T^G \cup T_{move1}$

Result: $\mathcal{N}(G_{step3.1}) = \{P^G, T^G, F^G, f^G, v^G, m^G_0\}$

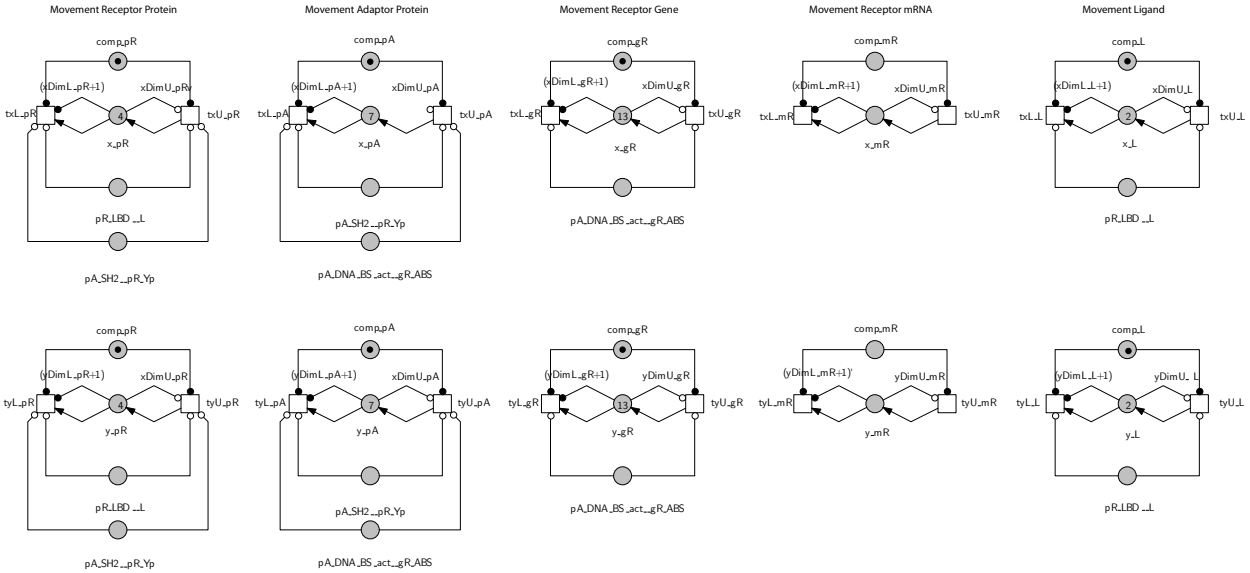


Figure 3.18: Movement of components of the running example as single entities (top - x-axis, bottom - y-axis). For each axis and component, two transitions are added to increase and decrease the marking of the coordinate place. Places representing an interaction of a component must be empty, which is tested by inhibitory arcs. For example, in the case of the movement of the receptor protein along the x-axis, the transition txL_pR removes tokens from the coordinate place x_pR as long as the marking is greater than the lower bound of the grid defined for the receptor protein given by the arc-weight $xDim\_L_pR + 1$ of the read arc. The transition txU_pR can add tokens to the coordinate place x_pR as long as the marking is less or equal than the upper bound of the grid defined for the receptor protein given by the arc-weight $xDim\_U$ of the inhibitory arc. The receptor protein can only move as a single entity if it is not bound to the ligand and the adaptor protein. Therefore, the places $pR\_LBD\_\_L$, $pA\_SH2\_\_pR\_Yp$ need to be empty. This is checked by linking those places via inhibitory arcs to the transitions txL_pR and txU_pR, which can now only remove or add tokens if those places $pR\_LBD\_\_L$, $pA\_SH2\_\_pR\_Yp$ are empty. To ensure that the receptor protein exists before moving it along the x-axis, the component place of the receptor protein $comp\_pR$ is connected with txL_pR and txU_pR via a read arc. *Note:* logical nodes have been used for a decluttered representation.

fore, the set of places P^G is now defined as $P^G = P^G \cup P_{comp}$. The component place p_c of component c is connected with each transition $t \in T^c_{move}$ via a read arc, such that $f^G_{RA}(p_c, t) = 1$. The initial number of instances of component c is defined by the parameter $n_{ini,c}$. The marking of the component place p_c is set $m^G_0(p_c) = n_{ini,c}$.

Running Example. To represent the movement of components as single entities, see Figure 3.18, we added two transitions for each coordinate place of each component in Figure 3.11 to increase and decrease the coordinate values:

- $x\_pR \rightarrow txL\_pR, txU\_pR$;
  $y\_pR \rightarrow tyL\_pR, tyU\_pR$
- $x\_gR \rightarrow txL\_gR, txU\_gR$;
  $y\_gR \rightarrow tyL\_gR, tyU\_gR$
- $x\_mR \rightarrow txL\_mR, txU\_mR$;
  $y\_mR \rightarrow tyL\_mR, tyU\_mR$
- $x\_pA \rightarrow txL\_pA, txU\_pA$;
  $y\_pA \rightarrow tyL\_pA, tyU\_pA$
- $x\_L \rightarrow txL\_L, txU\_L$;
  $y\_L \rightarrow tyL\_L, tyU\_L$

The marking of the coordinate places can only be changed if the respective component stays in its defined grid. Also, components can only move as single entities if they are not interacting with other components. Places representing an interaction state of a component are added to the respective transitions above via inhibitory arcs:

- receptor protein $\rightarrow pR\_LBD\_\_L, pA\_SH2\_\_pR\_Yp$
- receptor gene $\rightarrow pA\_DNA\_BS\_act\_\_gR\_ABS$
- adaptor protein $\rightarrow pA\_SH2\_\_pR\_Yp, pA\_DNA\_BS\_act\_\_gR\_ABS$
- ligand $\rightarrow pR\_LBD\_\_L$

To move a component as a single entity, we also must ensure that it actually exists. Therefore, the component places ($comp\_pR$, $comp\_gR$, $comp\_mR$, $comp\_pA$, $comp\_L$) are connected via read arcs with the corresponding transitions from above:

- $comp\_pR \rightarrow txL\_pR, txU\_pR, tyL\_pR, tyU\_pR$
- $comp\_gR \rightarrow txL\_gR, txU\_gR, tyL\_gR, tyU\_gR$
- $comp\_mR \rightarrow txL\_mR, txU\_mR, tyL\_mR, tyU\_mR$
- $comp\_pA \rightarrow txL\_pA, txU\_pA, tyL\_pA, tyU\_pA$
- $comp\_L \rightarrow txL\_L, txU\_L, tyL\_L, tyU\_L$

2. Local position change of single interaction site complexes, see Algorithm 3.8:
   For each single interaction site complex $k \in K^G$ and each dimension i, $1 \leq i \leq d$, two transitions $t^k_{i,L}$ and $t^k_{i,U}$ are needed to incrementally decrease or increase the amount of tokens on the respective coordinate places given by $P^k_{ax,i} = \bigcup_{c \in k} p^c_{ax,i}$. Thus, the movement of the single interaction site complex k is described by set of transitions $T^k_{move} = T^k_{move,L} \cup T^k_{move,U}$, where $T^k_{move,L} = \{t^k_{1,L}, \ldots, t^k_{d,L}\}$ and $T^k_{move,U} = \{t^k_{1,U}, \ldots, t^k_{d,U}\}$. The transition $t^k_{i,L}$ with the firing rate $h(t^k_{i,L})$ removes tokens from the set of coordinate places $P^k_{ax,i}$ till at least one component $c \in k$ reached the lower bound of its grid, $m(p^c_{ax,i}) = m_L(p^c_{ax,i})$. Accordingly, for each place $p^c_{ax,i} \in P^k_{ax,i}$ the arcs $f^G_{SA}(p^c_{ax,i}, t^k_{i,L}) = 1$ and $f^G_{RA}(p^c_{ax,i}, t^k_{i,L}) = m_L(p^c_{ax,i}) + 1$ have to be introduced for each component $c \in k$. The transition $t^k_{i,U}$ with the firing rate $h(t^k_{j,U})$ adds tokens to the set of coordinate places $P^k_{ax,i}$ till at least one component $c \in k$ reached the lower bound of its grid $m(p^c_{ax,i}) = m_U(p^c_{ax,i})$. Accordingly, for each place $p^c_{ax,i} \in P^k_{ax,i}$ the arcs $f^G_{SA}(t^k_{i,U}, p^c_{ax,i}) = 1$ and $f^G_{IA}(p^c_{ax,i}, t^k_{i,U}) = m_U(p^c_{ax,i})$ have to be introduced for each component $c \in k$. The total set of transition describing the movement of components is given by $T_{move2} = \bigcup_{k \in K^G} T^k_{move}$. Therefore, the set of places T^G is now defined as $T^G = T^G \cup T_{move2}$.
   A single interaction site complex $k \in K^G$ can only move if all places representing the respective interaction are marked, $m(P^k_{IS}) \neq 0$ with $P^k_{IS} = \{p \mid k = \lambda^c(\varrho_{p \rightarrow s}(p))\}$, $P^k_{IS} \subseteq P^G$. This condition is tested for each transition $t \in T^k_{move}$ and each place $p \in P^k_{IS}$ through a read arc $f_{RA}(p, t) = 1$.
   Furthermore, a single interaction site complex $k \in K^G$ can only move if all places representing other interaction of a component $c \in k$ in the single interaction site complex k are empty, $m(P^k_{coexIS}) = 0$ with $P^k_{coexIS} = \{p : \lambda^c(\varrho_{p \rightarrow s}(p)) = k', k' \in K^k_{coex}\}$ and $K^k_{coex} = \{k' \in K^G : k \cap k' \neq \emptyset \mid k \neq k'\}$. This condition is tested for each transition $t \in T^k_{move}$ and each place $p \in P^k_{coexIS}$ through an inhibitory arc $f_{IA}(p, t) = 1$.

Running Example. To represent the movement of the single interaction site complexes, see Figure 3.19, we added two transitions for each axis of each single interaction site complex to increase and decrease the coordinate values:

- receptor protein and ligand:

Algorithm 3.8: Local position change of single interaction site complexes.

Require: $\mathcal{N}(G_{step3.1}) = \{P^G, T^G, F^G, f^G, v^G, m_0^G\}$ with $P^G = P_{org}^G \cup P_{ax} \cup P_{comp}$ and $T^G = T_{org}^G \cup T_{move1}$

$T_{move}^k = \emptyset$
$T_{move2} = \emptyset$
for all $k \in K^G$ **do**
  for $i = 1$ **to** d **do**
    $T_{move}^k = T_{move}^k \cup \{t_{i,L}^k, t_{i,U}^k\}$ ▷ define transitions for movement
    for all $c \in k$ **do**
      $f_{SA}^G(p_{ax,i}^c, t_{i,L}^k) = 1$
      $f_{RA}^G(p_{ax,i}^c, t_{i,L}^k) = m_L(p_{ax,i}^c) + 1$
      $f_{SA}^G(t_{i,U}^k, p_{ax,i}^c) = 1$
      $f_{IA}^G(p_{ax,i}^c, t_{i,L}^k) = m_U(p_{ax,i}^c)$
      ▷ connect coordinate places and test for lower/upper bound
      for all $k' \in K^G \setminus k$ **do**
        if $k \cap k' \neq \emptyset$ **then**
          for all $p \in P_{org}^G$ **do**
            if $k' = \lambda^c(\varrho_{p \to s}(p))$ **then**
              $f_{IA}^G(p, t_{i,L}^k) = 1$
              $f_{IA}^G(p, t_{i,U}^k) = 1$
              ▷ test if places representing the respective interaction are marked
            end if
          end for
        end if
      end for
    end for
    for all $p \in P_{org}^G$ **do**
      if $k = \lambda^c(\varrho_{p \to s}(p))$ **then**
        $f_{RA}^G(p, t_{i,L}^k) = 1$
        $f_{RA}^G(p, t_{i,U}^k) = 1$
        ▷ test if places representing other interactions of the involved components are empty
      end if
    end for
  end for
  $T_{move2} = T_{move2} \cup T_{move}^k$
end for
$T^G = T^G \cup T_{move2}$

Result: $\mathcal{N}(G_{step3.2}) = \{P^G, T^G, F^G, f^G, v^G, m_0^G\}$

x_pR, $x\_L \to txL\_pR\_L$, txU_pR_L;
y_pR, $y\_L \to tyL\_pR\_L$, tyU_pR_L

- receptor protein and adaptor protein:
  x_pR, $x\_pA \to txL\_pR\_pA$, txU_pR_pA;
  y_pR, $y\_pA \to tyL\_pR\_pA$, tyU_pR_pA
- adaptor protein and receptor gene:
  x_pA, $x\_gR \to txL\_pA\_gR$, txU_pA_gR;
  y_pA, $y\_gR \to tyL\_pA\_gR$, tyU_pA_gR

The marking of the coordinate places can only be changed if the components of the single interaction site complex stay in their defined grid. Single interaction site complexes can only move if all places representing the respective complex are marked. Those places are added to the respective transitions above via read arcs:

- receptor protein and ligand: $pR\_LBD\_\_L$
- receptor protein and adaptor protein: $pA\_SH2\_\_pR\_Yp$
- adaptor protein and receptor gene: $pA\_DNA\_BS\_act\_\_gR\_ABS$

Also, single interaction site complexes can only move if all places representing other complexes of the involved components in the single interaction site complex are empty. Those places are added to the respective transitions above via inhibitory arcs:

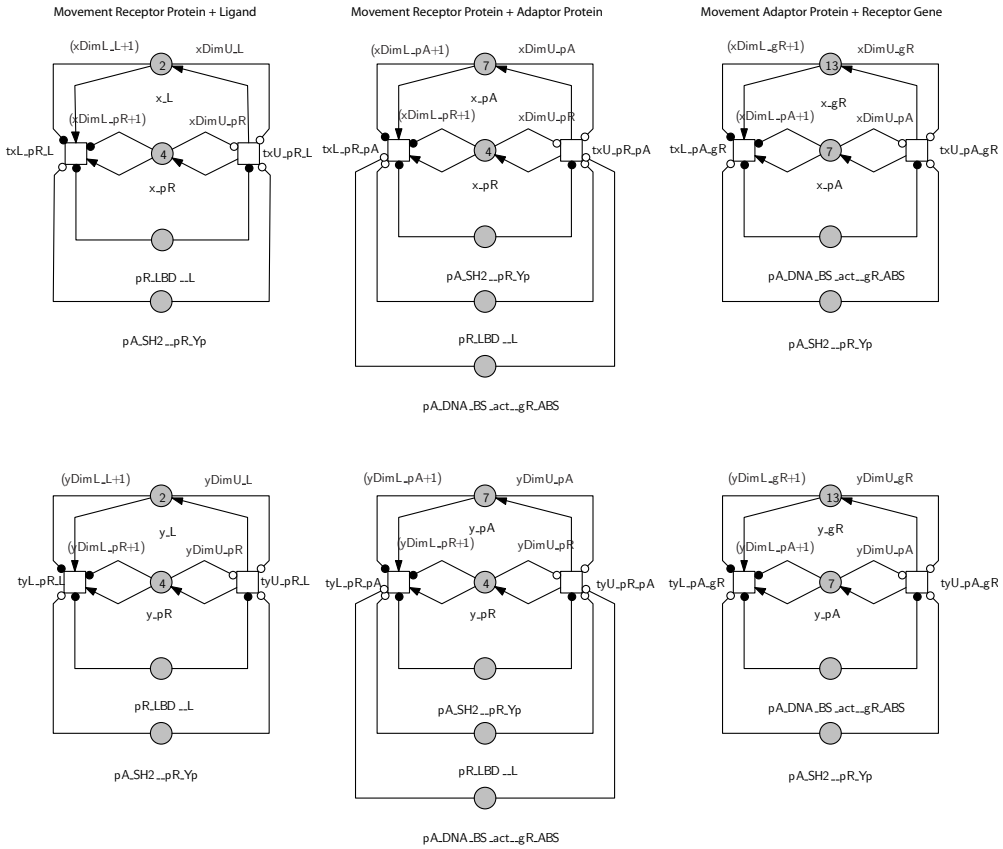


Figure 3.19: Movement of single interaction site complexes of the running example (top - x-axis, bottom - y-axis). For each axis and each single interaction site complex, two transitions are added to increase and decrease the marking of the coordinate place. Places representing the interaction of the single interaction site complex must be marked, which is tested by read arcs. Places representing other interactions of components not part of the single interaction site complex must be empty, which is tested by inhibitory arcs. Three single interaction site complexes can be determined in the running example: receptor protein and ligand; receptor protein and adaptor protein; adaptor protein and receptor gene. Compare also with Figure 3.18 on how to read the Petri net graphs. *Note:* logical nodes have been used for a decluttered representation.

- receptor protein and ligand: $pA\_SH2\_\_pR\_Yp$
- receptor protein and adaptor protein: $pR\_LBD\_\_L$, $pA\_DNA\_BS\_act\_\_gR\_ABS$
- adaptor protein and receptor gene: $pA\_SH2\_\_pR\_Yp$

3. Local position change of multi interaction site complexes, see Algorithm 3.9:

   A multi interaction site complex K_{coex} is defined as a combination of co-existing single interaction site complexes, where $K_{coex} \subseteq K^G$ and $|K_{coex}| > 1$. For each single interaction site complex $k \in K_{coex}$ it must be true, that there exists another single interaction site complex $k' \in K_{coex}$, $k' \neq k$, that shares a least one component c with k, $k \cap k' \neq \emptyset$. The set of components involved in the multi interaction site complex K_{coex} is given by $C^{K_{coex}} = \bigcup_{k \in K_{coex}} \bigcup_{c \in k} c$. For each multi interaction site complex $K_{coex} \subseteq K^G$, and each dimension i, $1 \leq i \leq d$, two transitions $t_{i,L}^{K_{coex}}$ and $t_{i,U}^{K_{coex}}$ are needed to incrementally decrease or increase the amount of tokens of the respective coordinate places given by $P_{ax,i}^{K_{coex}} = \bigcup_{k \in K_{coex}} \bigcup_{c \in k} p_{ax,i}^{c}$. Thus, the movement of the multi interaction site complex K_{coex} is described by set of transitions $T_{move}^{K_{coex}} = T_{move,L}^{K_{coex}} \cup T_{move,U}^{K_{coex}}$, where $T_{move,L}^{K_{coex}} = \{t_{1,L}^{K_{coex}}, \ldots, t_{d,L}^{K_{coex}}\}$ and $T_{move,U}^{K_{coex}} = \{t_{1,U}^{K_{coex}}, \ldots, t_{d,U}^{K_{coex}}\}$. The transition $t_{i,L}^{K_{coex}}$ with the firing rate $h(t_{i,L}^{K_{coex}})$ removes tokens from the set of coordinate places $P_{ax,i}^{K_{coex}}$ till at least one component $c \in C^{K_{coex}}$ reached the lower bound of its grid, $m(p_{ax,i}^{c}) = m_L(p_{ax,i}^{c})$. Thus, for each component $c \in C^{K_{coex}}$ the arcs $f_{SA}^{G}(p_{ax,i}^{c}, t_{i,L}^{K_{coex}}) = 1$

Algorithm 3.9: Local position change of multi interaction site complexes.

Require: $\mathcal{N}(G_{step3.2}) = \{P^G, T^G, F^G, f^G, v^G, m_0^G\}$ with $P^G = P_{org}^G \cup P_{ax} \cup P_{comp}$ and $T^G = T_{org}^G \cup T_{move1} \cup T_{move2}$

$T_{move}^{K_{coex}} = \emptyset$
$T_{move3} = \emptyset$
$K' = K^G$
for all $k \in K^G$ **do** ▷ determine all possible combinations of co-existing complexes
  $K_{coex}^k = k$
  $K' = K' \setminus k$
  for all $k' \in K'$ **do**
    if $k \cap k' \neq \emptyset$ **&** $k \not\subseteq k'$ **&** $k' \not\subseteq k$ **&** $k' \in K_{coex}^{all}$ **then**
      $K_{coex}^k = K_{coex}^k \cup k'$
    end if
  end for
  generateMovement(1,$\emptyset$,K_{coex}^k)
end for
$T^G = T^G \cup T_{move3}$

Result: $\mathcal{N}(G_{step3.3}) = \{P^G, T^G, F^G, f^G, v^G, m_0^G\}$

function generateMovement(z,K_{coex}^{curr},K_{coex}^k)
  ▷ encode movement of each possible combination of co-existing complexes
  for $i = z$ **to** $|K_{coex}^k|$ **do**
    $K_{coex}^{curr'} = K_{coex}^{curr} \cup k_{coex,i}$
    if $z > 1$ **then**
      for j=1 **to** d **do**
        $T_{move}^{K_{coex}} = T_{move}^{K_{coex}} \cup \{t_{j,L}^{K_{coex}^{curr'}}, t_{j,U}^{K_{coex}^{curr'}}\}$
        for all $k \in K_{coex}^{curr'}$ **do**
          for all $c \in k$ **do**
            $f_{SA}^G(p_{ax,j}^c, t_{j,L}^{K_{coex}^{curr'}}) = 1$
            $f_{RA}^G(p_{ax,j}^c, t_{j,L}^{K_{coex}^{curr'}}) = m_L(p_{ax,j}^c) + 1$
            $f_{SA}^G(t_{j,U}^{K_{coex}^{curr'}}, p_{ax,j}^c) = 1$
            $f_{IA}^G(p_{ax,j}^c, t_{j,L}^{K_{coex}^{curr'}}) = m_U(p_{ax,j}^c)$
            ▷ connect coordinate places and test for lower/upper bound
            for all $k' \in K^G \setminus K_{coex}^{curr'}$ **do**

              for all $k'' \in K_{coex}^{curr'}$ **do**
                if $k'' \cap k' \neq \emptyset$ **then**
                  for all $p \in P_{org}^G$ **do**
                    if $k' = \lambda^c(\varrho_{p \to s}(p))$ **then**
                      $f_{IA}^G(p, t_{i,L}^{K_{coex}^{curr'}}) = 1$
                      $f_{IA}^G(p, t_{i,U}^{K_{coex}^{curr'}}) = 1$
                      ▷ test if places representing other interactions of the involved components are empty
                    end if
                  end for
                end if
              end for
            end for
          end for
          for all $p \in P_{org}^G$ **do**
            if $k = \lambda^c(\varrho_{p \to s}(p))$ **then**
              $f_{RA}^G(p, t_{i,L}^{K_{coex}^{curr'}}) = 1$
              $f_{RA}^G(p, t_{i,U}^{K_{coex}^{curr'}}) = 1$
              ▷ test if places representing the respective interaction are marked
            end if
          end for
        end for
      end for
      $T_{move3} = T_{move3} \cup T_{move}^{K_{coex}}$
      $T_{move}^{K_{coex}} = \emptyset$
    end if
    generateMovement(i,$K_{coex}^{curr'}$,K_{coex}^k)
  end for
end function

and $f^G_{RA}(p^c_{ax,i}, t^{K'_{coex}}_{i,L}) = m_L(p^c_{ax,i}) + 1$ have to be introduced. The transition $t^{K_{coex}}_{i,U}$ with the firing rate $h(t^{K_{coex}}_{i,U})$ adds tokens to the set of coordinate places $P^{K_{coex}}_{coor,i}$ till at least one component $c \in C^{K_{coex}}$ reached the upper bound of its grid, $m(p^c_{ax,i}) = m_U(p^c_{ax,i})$. Accordingly, for each component $c \in C^{K_{coex}}$ the arcs $f^G_{SA}(t^{K_{coex}}_{i,U}, p^c_{ax,i}) = 1$ and $f^G_{IA}(p^c_{ax,i}, t^{K_{coex}}_{i,U}) = m_U(p^c_{ax,i})$ have to be introduced. The total set of transition describing the movement of components is given by $T_{move3} = \{T^{K_{coex,1}}_{move}, \ldots, T^{K_{coex,n}}_{move}$. Therefore, the set of places T^G is now defined as $T^G = T^G \cup T_{move3}$.
A multi interaction site complex $K_{coex} \subseteq K^G$ can only move if all places representing the respective interactions involved in multi interaction site complex are marked, $m(P^{K_{coex}}_{IS}) \neq 0$ with $P^{K_{coex}}_{IS} = \{p \in P^G : k = \lambda^c(\varrho_{p\to s}(p)) \wedge k \in K_{coex}\}$. This condition is tested for each transition $t \in T^{K_{coex}}_{move}$ and each place $p \in P^{K_{coex}}_{IS}$ through a read arc $f_{RA}(p,t) = 1$.
Furthermore, a multi interaction site complex $K_{coex} \subseteq K^G$ can only move if all places representing other interactions of a component $c \in C^{K_{coex}}$ are empty, $m(P^{K_{coex}}_{coexIS}) = 0$ with $P^{K_{coex}}_{coexIS} = \{p \in P^G : k = \lambda^c(\varrho_{p\to s}(p)) \wedge k \in K^{K_{coex}}_{coex}\}$ and $K^{K_{coex}}_{coex} = \{k \in K^G : k \notin K_{coex} \wedge \exists k' \in K'_{coex} : k \cap k' \neq \emptyset, k \neq k'\}$. This condition is tested for each transition $t \in T^{K_{coex}}_{move}$ and each place $p \in P^{K_{coex}}_{coexIS}$ through an inhibitory arc $f_{IA}(p,t) = 1$.

The local position change of individual components, single and multi interaction site components can either be regarded in a discrete or deterministic manner that may occur stochastically or deterministically, respectively. According to the choice, the introduced coordinate places P_{ax}, see Algorithm 3.5, and transitions representing the local position change $T_{move} = T_{move1} \cup T_{move2} \cup T_{move3}$, see Algorithm 3.7- 3.9, have to be declared as discrete or as continuous nodes. Theoretically, the firing rates of the transitions representing the local position change $h(t)$, $t \in T_{move}$, can be specified as mass-action kinetics, a Brownian motion kinetics, diffusion kinetics according to Fick's laws of diffusion [1] or any other kinetics compatible with the modelled system and assumptions.

Running Example. As before, to represent the movement of the multi interaction site complexes, see Figure 3.20, we added two transitions for each axis of each multi interaction site complex to increase and decrease the coordinate values:

- receptor protein, adaptor protein and ligand:
  x_pR, x_pA, $x\_L \to txL\_pR\_pA\_L$, $txU\_pR\_pA\_L$;
  y_pR, y_pA, $y\_L \to tyL\_pR\_pA\_L$, $tyU\_pR\_pA\_L$
- receptor protein, adaptor protein and receptor gene:
  x_pR, x_pA, $x\_gR \to txL\_pR\_pA_gR$, $txU\_pR\_pA_gR$;
  y_pR, y_pA, $y\_gR \to tyL\_pR\_pA_gR$, $tyU\_pR\_pA_gR$
- receptor protein, adaptor protein, receptor gene and ligand:
  x_pR, x_pA, x_L, $x\_gR \to txL\_pR\_pA\_gR\_L$, $txU\_pR\_pA\_gR\_L$;
  y_pR, y_pA, y_L, $y\_gR \to tyL\_pR\_pA\_gR\_L$, $tyU\_pR\_pA\_gR\_L$

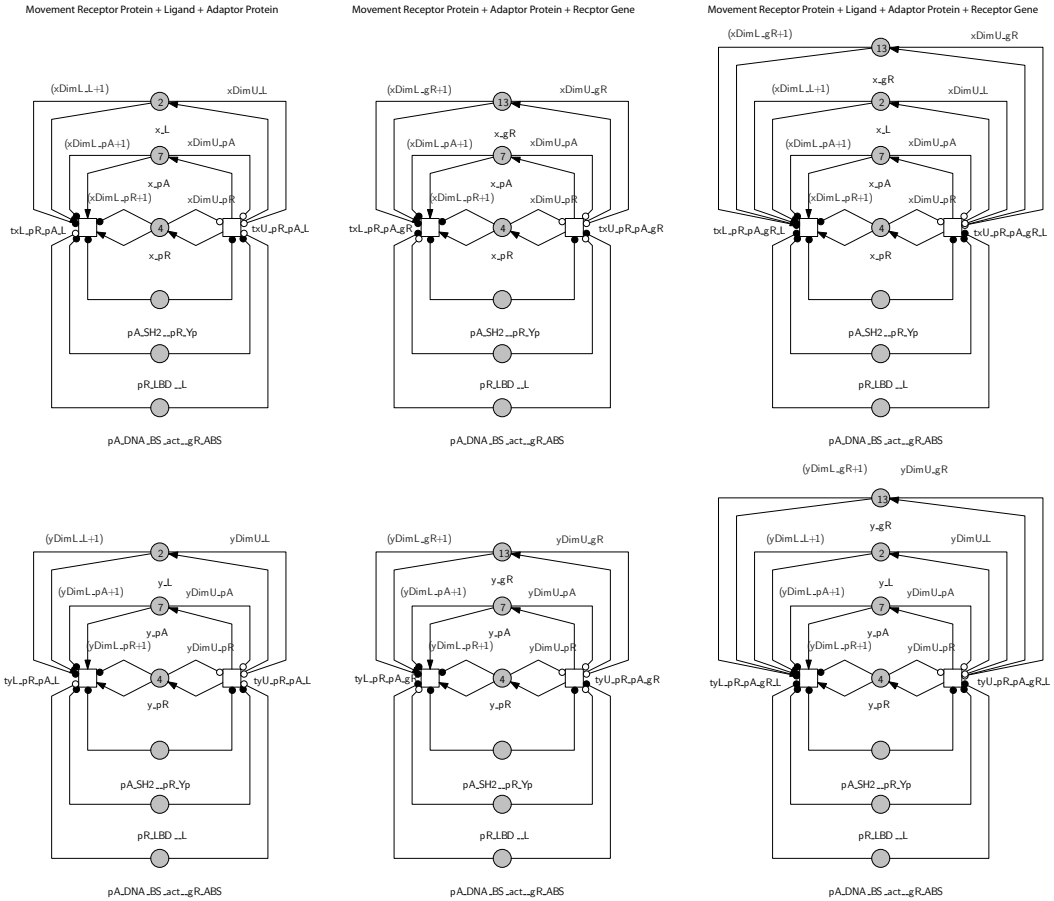


Figure 3.20: Movement of multi interaction site complexes of the running example (top - x-axis, bottom - y-axis). For each axis and multi interaction site complex, two transitions are added to increase and decrease the marking of the coordinate place on the defined grid size. Places representing the interaction of the multi interaction site complex must be marked, which is tested by read arcs. Places representing other interactions of components part of the multi interaction site complex must be empty, which is tested by inhibitory arcs. Three multi interaction site complexes can be determined in the running example: receptor protein, adaptor protein and ligand; receptor protein, adaptor protein and receptor gene; receptor protein, adaptor protein, receptor gene and ligand. Compare also with Figure 3.18 on how to read the Petri net graphs. *Note:* logical nodes have been used for a decluttered representation.

The marking of the coordinate places can only be changed if the components of the multi interaction site complex stay in their defined grid. Multi interaction site complexes can only move if all places representing the respective complex are marked. Those places are added to the respective transitions above via read arcs:

- receptor protein, adaptor protein and ligand: $pR\_LBD\_\_L$, $pA\_SH2\_\_pR\_Yp$
- receptor protein, adaptor protein and receptor gene: $pA\_SH2\_\_pR\_Yp$, $pA\_DNA\_BS\_act\_\_gR\_ABS$
- receptor protein, adaptor protein, receptor gene and ligand: $pR\_LBD\_\_L$, $pA\_SH2\_\_pR\_Yp$, $pA\_DNA\_BS\_act\_\_gR\_ABS$

Also, multi interaction site complexes can only move if all places representing other complexes of the involved components in the multi interaction site complex are empty. Those places are added to the respective transitions above via inhibitory arcs:

- receptor protein, adaptor protein and ligand: $pA\_DNA\_BS\_act\_\_gR\_ABS$
- receptor protein, adaptor protein and receptor gene: $pR\_LBD\_\_L$

Extended Model Instantiation using Coloured Petri Nets.

The encoding of component instances has already been introduced in Section 3.3.2. In order to represent the movement of a number of instances of each component, Algorithm 3.1 needs to be applied to the original modularly composed model and to be extended to capture the modifications that have been introduced in the previous steps, which include:

Algorithm 3.10: Encoding of Component Instances by applying coloured Petri nets.

Require: $\mathcal{N}(G_{step3.3}) = \{P^G, T^G, F^G, f^G, v^G, m_0^G\}$ with $T^G = T^G_{org} \cup T_{move}$, $T^G_{move} = T_{move1} \cup T_{move2} \cup T_{move3}$ and $P^G = P^G_{org} \cup P_{ax} \cup P_{comp}$

1: Run Algorithm 3.1 for $\mathcal{N}(G) = \{P^G, T^G, F^G, f^G, v^G, m_0^G\}$
2: $P^{G_{col}} = P^{G_{col}} \cup P_{ax}$
3: $T^{G_{col}} = T^{G_{col}} \cup T_{move}$
4: **for all** $p^c_{ax,i} \in P_{ax}$ **do**
5: $\quad \psi(p^c_{ax,i}) = w'_c$ ▷ assign colour set to respective coordinate places
6: $\quad$ **for all** $t \in T^{G_{col}}$ **do**
7: $\quad\quad f^{G_{col}}_{SA}(p^c_{ax,i}, t) = f^G_{SA}(p^c_{ax,i}, t)\text{`}a_c$
8: $\quad\quad f^{G_{col}}_{SA}(t, p^c_{ax,i}) = f^G_{SA}(t, p^c_{ax,i})\text{`}a_c$
9: $\quad\quad f^{G_{col}}_{RA}(p^c_{ax,i}, t) = f^G_{RA}(p^c_{ax,i}, t)\text{`}a_c$
10: $\quad\quad f^{G_{col}}_{IA}(p^c_{ax,i}, t) = f^G_{IA}(p^c_{ax,i}, t)\text{`}a_c$ } ▷ adjust arc-expression according to colour set of the coordinate place
11: $\quad$ **end for**
12: $\quad m^{G_{col}}(p^c_{ax,i}) = m^G(p^c_{ax,i})\text{`}1 + \ldots + m^G(p^c_{ax,i})\text{`}n_{ini,c}$ ▷ adjust marking according to colour set of the coordinate place
13: **end for**
14: **for all** $p^c \in P^c$ **do**
15: $\quad$ **for** $i = 1$ **to** d **do**
16: $\quad\quad f^{G_{col}}_{RA}(p^c, t^c_{i,L}) = 1\text{`}a_c$
17: $\quad\quad f^{G_{col}}_{RA}(p^c, t^c_{i,U}) = 1\text{`}a_c$ } ▷ connect component places to transitions representing the local position change of individual components.
18: $\quad$ **end for**
19: **end for**
20: **for all** $p^c_{ax,i} \in P_{ax}$ **do**
21: $\quad$ **for all** $t \in T^{G_{col}}$ **do**
22: $\quad\quad$ **if** $c \notin \bigcup_{p \in {}^\bullet t} \lambda^c(\varrho_{p \to s}(p))$ & $c \in \bigcup_{p \in t^\bullet} \lambda^c(\varrho_{p \to s}(p))$ **then**
23: $\quad\quad\quad f^{G_{col}}_{SA}(t, p^c_{ax,i}) = x^c_i\text{`}a_c$
24: $\quad\quad$ **end if**
25: $\quad\quad$ **if** $c \notin \bigcup_{p \in t^\bullet} \lambda^c(\varrho_{p \to s}(p))$ & $c \in \bigcup_{p \in {}^\bullet t} \lambda^c(\varrho_{p \to s}(p))$ **then**
26: $\quad\quad\quad f^{G_{col}}_{XA}(p^c_{ax,i}, t) = a_c$
27: $\quad\quad$ **end if**
28: $\quad$ **end for**
29: **end for** } ▷ connect coordinate places of a component to transitions describing its synthesis or degradation
30: **for all** $\{t^c_{i,L} \cup t^c_{i,U}\} \in T^{G_{col}}_{move1}$ **do**
31: $\quad$ **for all** $p \in P^{G_{col}}_{org}$ **do**
32: $\quad\quad$ **if** $c \in \lambda^c(\varrho_{p \to s}(p))$ & $|\lambda^c(\varrho_{p \to s}(p))| > 1$ **then**
33: $\quad\quad\quad A = \emptyset$
34: $\quad\quad\quad$ **for all** $c' \in \lambda^c(\varrho_{p \to s}(p))$ **do**
35: $\quad\quad\quad\quad$ **if** $c' = c$ **then**
36: $\quad\quad\quad\quad\quad A = A \cup a_{c'}$
37: $\quad\quad\quad\quad$ **else**
38: $\quad\quad\quad\quad\quad A = A \cup "all()"$
39: $\quad\quad\quad\quad$ **end if**
40: $\quad\quad\quad$ **end for**
41: $\quad\quad\quad f^{G_{col}}_{IA}(p, t^c_{i,L}) = f^G_{IA}(p, t^c_{i,L})\text{`}A$
42: $\quad\quad\quad f^{G_{col}}_{IA}(p, t^c_{i,U}) = f^G_{IA}(p, t^c_{i,U})\text{`}A$ } ▷ adjust arc-expression of inhibitory arcs
43: $\quad\quad$ **end if**
44: $\quad$ **end for**
45: **end for**

Algorithm 3.10: Encoding of component instances by applying coloured Petri nets (cont.).

for all $\{t^k_{i,L} \cup t^k_{i,U}\} \in T^{G_{col}}_{move2}$ **do**
  for all $k' \in K^G \setminus k$ **do**
    if $k \cap k' \neq \emptyset$ **then**
      for all $p \in P^{G_{col}}_{org}$ **do**
        if $k' = \lambda^c(\varrho_{p\to s}(p))$ **then**
          $A = \emptyset$
          for all $c \in k'$ **do**
            if $c \in k$ **then**
              $A = A \cup a_c$
            else
              $A = A \cup "all()"$
            end if
          end for
          $f^{G_{col}}_{IA}(p, t^k_{i,L}) = f^G_{IA}(p, t^k_{i,L})`A$
          $f^{G_{col}}_{IA}(p, t^k_{i,U}) = f^G_{IA}(p, t^k_{i,U})`A$
          ▷ adjust arc-expression of inhibitory arcs
        end if
      end for
    end if
  end for
end for
for all $\{t^{K_coex}_{i,L} \cup t^{K_coex}_{i,U}\} \in T^{G_{col}}_{move3}$ **do**
  for all $k' \in K^G \setminus K_coex$ **do**
    for all $k \in K_coex$ **do**
      if $k \cap k' \neq \emptyset$ **then**
        for all $p \in P^{G_{col}}_{org}$ **do**
          if $k' = \lambda^c(\varrho_{p\to s}(p))$ **then**
            $A = \emptyset$
            for all $c \in k'$ **do**
              if $c \in k$ **then**
                $A = A \cup a_c$
              else
                $A = A \cup "all()"$
              end if
            end for
            $f^{G_{col}}_{IA}(p, t^{K_coex}_{i,L}) = f^G_{IA}(p, t^{K_coex}_{i,L})`A$
            $f^{G_{col}}_{IA}(p, t^{K_coex}_{i,U}) = f^G_{IA}(p, t^{K_coex}_{i,U})`A$
            ▷ adjust arc-expression of inhibitory arcs
          end if
        end for
      end if
    end for
  end for
end for

Result: $\mathcal{N}^{col}(G_{step4}) = \{P^{G_{col}}, T^{G_{col}}, F^{G_{col}}, W^{G_{col}}, \psi^{G_{col}}, g^{G_{col}}, f^{G_{col}}, v^{G_{col}}, m_0^{G_{col}}\}$

- Extending the set of coloured places $P^{G_{col}} = P^{G_{col}} \cup P_{ax} \cup P_{comp}$
- Setting the marking for each coordinate place $p^c_{ax,i} \in P_{ax}$, $m^{G_{col}}(p^c_{ax,i}) = m^G(p^c_{ax,i})`1 + \ldots + m^G(p^c_{ax,i})`n_{ini,c}$
- Mapping of a coordinate place $p^c_{ax,i} \in P_{ax}$ of component $c \in C^G$ to the simple colour set $\psi(p^c_{ax,i}) = w'_c$
- Connecting each coordinate place $p^c_{ax,i} \in P_{ax}$ of a component $c \in C^G$to the transitions representing its synthesis or degradation:
  - Synthesis: For each $t \in T^{G_{col}}$, where $c \notin \bigcup_{p\in\bullet t} \lambda^c(\varrho_{p\to s}(p))$ and $c \in \bigcup_{p\in t\bullet} \lambda^c(\varrho_{p\to s}(p))$ set the arc-expression to $f^{G_{col}}_{SA}(t, p^c_{ax,i}) = x^c_i`a_c$
  - Degradation: For each $t \in T^{G_{col}}$, where $c \notin \bigcup_{p\in t\bullet} \lambda^c(\varrho_{p\to s}(p))$ and $c \in \bigcup_{p\in\bullet t} \lambda^c(\varrho_{p\to s}(p))$ set the arc-expression to $f^{G_{col}}_{XA}(p^c_{ax,i}, t) = a_c$

- Extending the set of coloured transitions $T^{G_{col}} = T^{G_{col}} \cup T_{move}$, where $T^{G}_{move} = T_{move1} \cup T_{move2} \cup T_{move3}$
- Setting the firing rates for each transition $t \in T^{G}_{move}$, such that $h^{G_{col}}(t) = h(t)`all()$
- Setting the arc-expression for in-going and out-going arcs of a coordinate place $p^{c}_{ax,i} \in P_{ax}$ of a component c to $f^{G_{col}}(p^{c}_{ax,i}, t) = f^{G}(p^{c}_{ax,i}, t)`a_c$ and $f^{G_{col}}(t, p^{c}_{ax,i}) = f^{G}(t, p^{c}_{ax,i})`a_c$, where $t \in T^{G_{col}}$
- Setting the arc-expression of read arcs connecting a component place $p_c \in P_{comp}$ with a transition $t \in T_{move1}$ to $f^{G_{col}}_{RA}(p_c, t) f^{G}(p_c, t)`a_c$
- Setting the arc-expression for newly introduced in-going and out-going arcs for each place $p \in P^{G}_{org}$ with $\psi(p) = w''_k$:
  1. $f^{G_{col}}_{RA}(p,t) = f^{G}_{RA}(p,t)`\bigcup_{c \in k} a_c$, where $t \in T^{G}_{move}$
  2. $f^{G_{col}}_{IA}(p,t) = f^{G}_{IA}(p,t)`A$, where $t \in T^{G}_{move}$:
     - $A = \bigcup_{c \in \lambda^c(\varrho_{p \to s}(p))} \begin{cases} a_c, & c = c' \\ all(), & c \neq c' \end{cases}$ if $t^{c'}_{i,L/U} \in T_{move1}$
     - $A = \bigcup_{c \in \lambda^c(\varrho_{p \to s}(p))} \begin{cases} a_c, & c \in k \\ all(), & c \notin k \end{cases}$ if $t^{k}_{i,L/U} \in T_{move2}$
     - $A = \bigcup_{c \in \lambda^c(\varrho_{p \to s}(p))} \begin{cases} a_c, & c \in \bigcup_{k \in K_{coex}} \bigcup_{c' \in k} c' \\ all(), & c \notin \bigcup_{k \in K_{coex}} \bigcup_{c' \in k} c' \end{cases}$ if $t^{K_{coex}}_{i,L/U} \in T_{move3}$

Running Example. Transforming the he spatial model into a coloured Petri net to represent multiple instances of components requires inheriting the declarations defined for the running example in Section 3.3.2, compare also Figure 3.21 and 3.22:

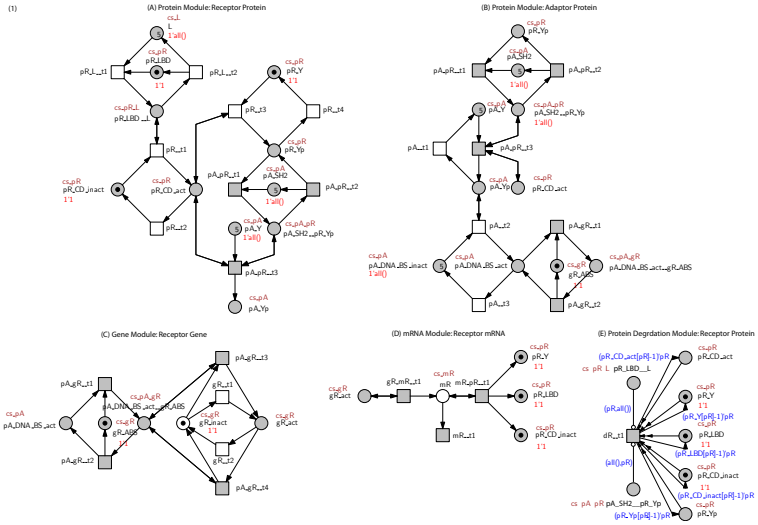
- simple colour sets:
  - $cs\_pR := 1 - n_{max,pR}$
  - $cs\_pA := 1 - n_{max,pA}$
  - $cs\_gR := 1 - n_{max,gR}$
  - $cs\_mR := 1 - n_{max,mR}$
  - $cs\_L := 1 - n_{max,L}$
- compound colour sets:
  - $cs\_pR\_L := cs\_pR \times cs\_L$
  - $cs\_pA\_pR := cs\_pA \times cs\_pR$
  - $cs\_pA\_gR := cs\_pA \times cs\_gR$
- constants:
  - $n_{max,pR} = 10$, $n_{ini,pR} = 1$
  - $n_{max,pA} = 10$, $n_{ini,pA} = 5$
  - $n_{max,gR} = 1$, $n_{ini,gR} = n_{max,gR}$
  - $n_{max,mR} = 3$, $n_{ini,mR} = 0$
  - $n_{max,L} = 10$, $n_{ini,L} = 5$
- variables:
  - $pR := cs\_pR$
  - $pA := cs\_pA$
  - $gR := cs\_gR$
  - $mR := cs\_mR$
  - $L := cs\_L$

Figure 3.21: Spatial model of the running example (Part I), see also Figure 3.22 (Part II). (A) declaration of simple and compound colour sets, constants and variables; (B) modular composed model representing the molecular mechanism as described in Figure 3.2, compare also Figure 3.3; (C) component places, compare Figure 3.12; (D) coordinate places, compare Figure 3.16; (E) local restriction of molecular events representing interactions, compare Figure 3.17; (F) additional modifications for the synthesis and degradation of the receptor mRNA and protein; compare Figure 3.12. *Note:* logical nodes have been used for a decluttered representation; colour set of a place is given maroon letters; marking of a place is given in red letters; arc-expressions are given in blue letters (only non-trivial arc-expression are shown for clearness)

(A)

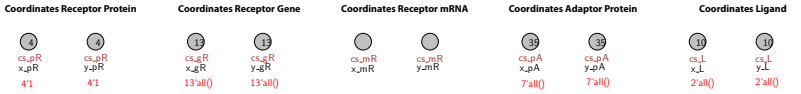
- simple colour sets:
  - $cs\_pR := 1 - n_{max,pR}$
  - $cs\_pA := 1 - n_{max,pA}$
  - $cs\_gR := 1 - n_{max,gR}$
  - $cs\_mR := 1 - n_{max,mR}$
  - $cs\_L := 1 - n_{max,L}$
- compound colour sets:
  - $cs\_pR\_L := cs\_pR \times cs\_L$
  - $cs\_pA\_pR := cs\_pA \times cs\_pR$
  - $cs\_pA\_gR := cs\_pA \times cs\_gR$
- constants:
  - $n_{max,pR} = 10$, $n_{ini,pR} = 1$
  - $n_{max,pA} = 10$, $n_{ini,pA} = 5$
  - $n_{max,gR} = 1$, $n_{ini,gR} = n_{max,gR}$
  - $n_{max,mR} = 3$, $n_{ini,mR} = 0$
  - $n_{max,L} = 10$, $n_{ini,L} = 5$
- variables:
  - $pR := cs\_pR$
  - $pA := cs\_pA$
  - $gR := cs\_gR$
  - $mR := cs\_mR$
  - $L := cs\_L$

(B)

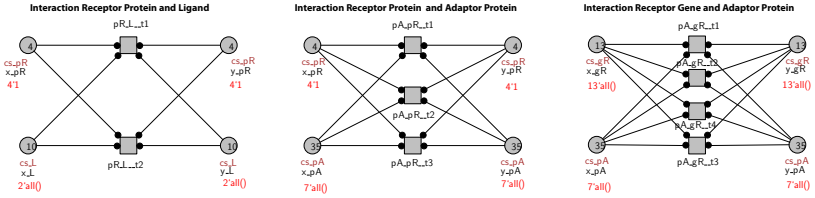


(C)

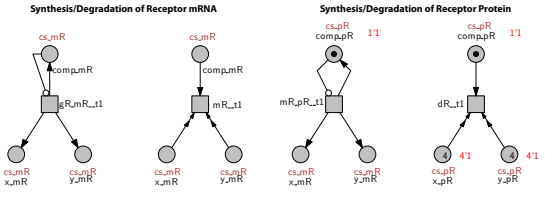
(D)



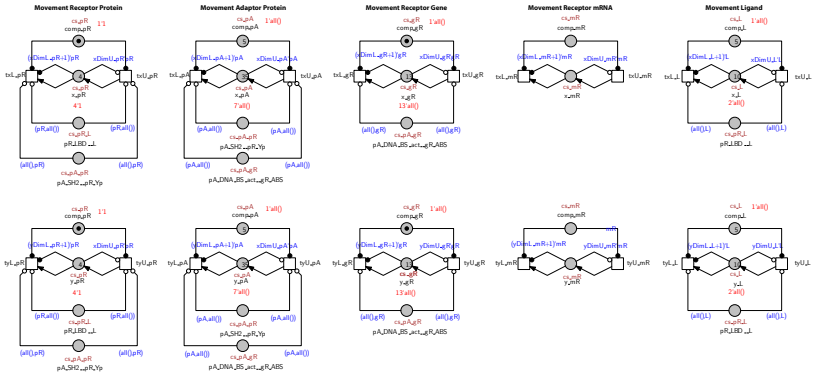
(E)



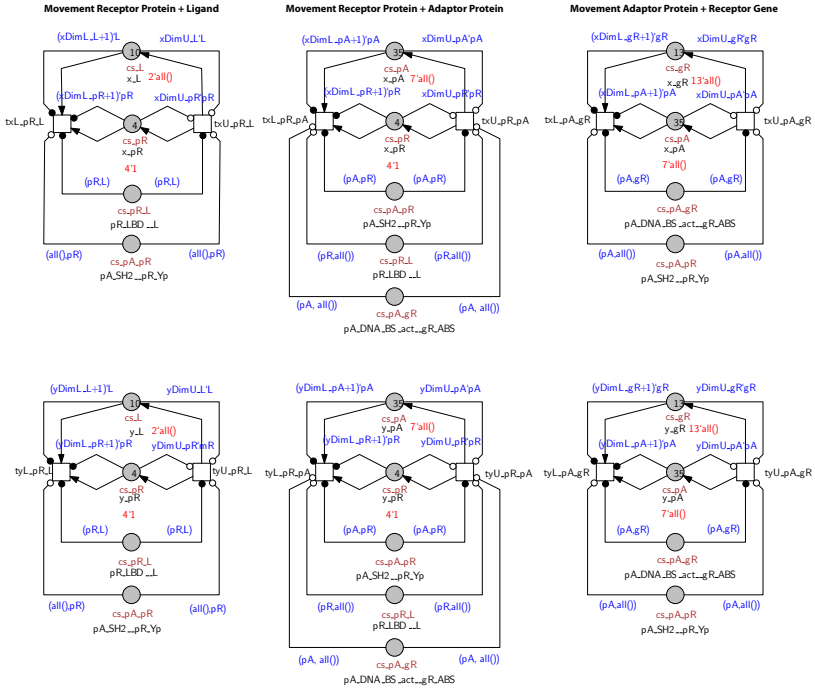
(F)



(G)



(H)



(J)

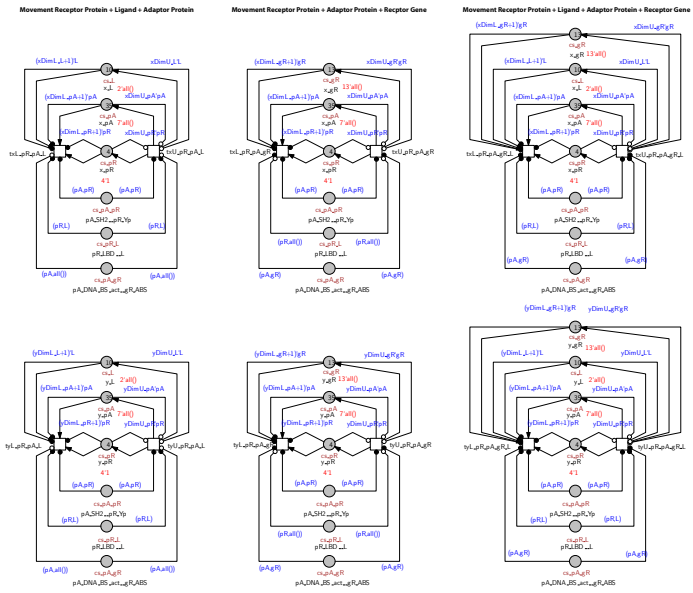


Figure 3.22: Spatial model of the running example (Part II), see also Figure 3.21 (Part I). (G) movement of components as single entities, compare Figure 3.18; (H) movement of single interaction site complexes, compare Figure 3.19; (J) movement of multi interaction site complexes, compare Figure 3.20 *Note:* logical nodes have been used for a decluttered representation; colour set of a place is given maroon letters; marking of a place is given in red letters; arc-expressions are given in blue letters (only non-trivial arc-expression are shown for clearness). See also text for an explanation of arc-expression of type $(x, all())$.

All subnets defined in Figure 3.21 and 3.22 are connected via the use of logical nodes and thus, have to be interpreted as one single coherent coloured Petri net model.

Coordinate places must also be connected to all transitions representing the synthesis or degradation of a component, see Figure 3.21(F). The coordinate places of the receptor mRNA x_mR, y_mR are post-places of the transition gR_mR_t1 (synthesis) connected by standards arcs, and are pre-places of the transition mR_t1 (degradation) connected by a reset arcs. In case of the receptor protein, the coordinate places x_pR, y_pR are post-places of the transition mR_pR_t1 (synthesis) connected by standards arcs, and are pre-places of the transition dR_t1 (degradation) connected by a reset arcs.

Additional arc-expressions have to be introduced in Figure 3.18 - 3.18 for the inhibitory arcs connecting transitions representing the movement of a component or complex and places representing interaction states that block the corresponding movement, compare Figure 3.22. For example, each transition describing the movement of the receptor protein as a single entity (txL_pR, txU_pR, tyL_pR, tyU_pR) is blocked by the interaction states $pR\_LBD\_\_L$, $pA\_SH2\_\_pR\_Yp$. Due to the instantiation using coloured Petri nets, we assumed that the receptor protein, ligand, adaptor protein can exits in multiple copies. Thus, the instances 1, ..., 10 of the receptor protein can interact with instance 1, 2, ..., 10 of the ligand and/or with instance 1, 2, ..., 10 of the adaptor protein. An instance of the receptor protein is only allowed to move as a single entity if it does not interact with any instance of the ligand or adaptor protein. Therefore, the arc-expression $(pR, all())$ (order of terms must be set according to order of simple colour sets in the corresponding compound set, here $cs\_pR\_L := cs\_pR \times cs\_L$) is applied to the inhibitory arcs connecting the place $pR\_LBD\_\_L$ with transitions txL_pR, txU_pR, tyL_pR, tyU_pR. The arc-expression $(pR, all())$ is a short notation for the multiset $(pR, 1) + +(pR, 2) + + \ldots + +(pR, 10)$, where the first record in each tuple defines the instance of the receptor protein and the second record defines the instance of the ligand. Thus, the inhibitory arc now blocks the movement of an instance of the receptor protein given by pR as a single entity if it is in complex with any instance of the ligand. The arc-expression $(all(), pR)$ (order of terms must be set according to order of simple colour sets in the corresponding compound set, here $cs\_pA\_pR := cs\_pA \times cs\_pR$) is applied to the inhibitory arcs connecting the place $pA\_SH2\_\_pR\_Yp$ with transitions txL_pR, txU_pR, tyL_pR, tyU_pR. The arc-expression $(all(), pR)$ is a short notation for the multiset $(1, pR) + +(2, pR) + + \ldots + +(10, pR)$, where the first record in each tuple defines the instance of the adaptor protein and the second record defines the instance of the receptor protein. Thus, the inhibitory arc now blocks the movement of an instance of the receptor protein given by pR as a single entity if it is in complex with any instance of the adaptor protein. All similar arc-expression in Figure 3.22 have to be interpreted accordingly.

Definition of Compartments

The spatial transformation algorithm generates an unwrought spatial configuration of the modularly composed model. It might be necessary to adapt or modify the generated spatial model according to the spatial properties of the considered molecular network. Modification might include the variation of the size or form of the grid to represent different cell sizes and shapes. The compartmentalisation of the grid allows adapting cellular compartments, membranes, pools, and any other intracellular structure.

Therefore, a number of compartments $\mathbf{\Omega} = \Omega_1, \ldots, \Omega_n$ and their position on the grid has to be defined. For each compartment Ω_i a boundary has to be given, e.g. in the case of a rectangular the x-intercept is defined by $x_L^{\Omega_i}$ and $x_U^{\Omega_i}$, and the y-intercept by $y_L^{\Omega_i}$ and $y_U^{\Omega_i}$. For each component $c \in C^G$ it must be specified, in which compartments the component is allowed to stay $\mathbf{\Omega}(c) \subseteq \mathbf{\Omega}$. Each firing-rate $h(t)$ of a transition t describing the movement of component c has to be multiplied by a Boolean expression $\omega(\mathbf{\Omega}(c))$ evaluating, whether the respective component stays within its assigned compartments or not. Only if the Boolean expression is evaluated to true, the transition might fire, and the movement will be executed.

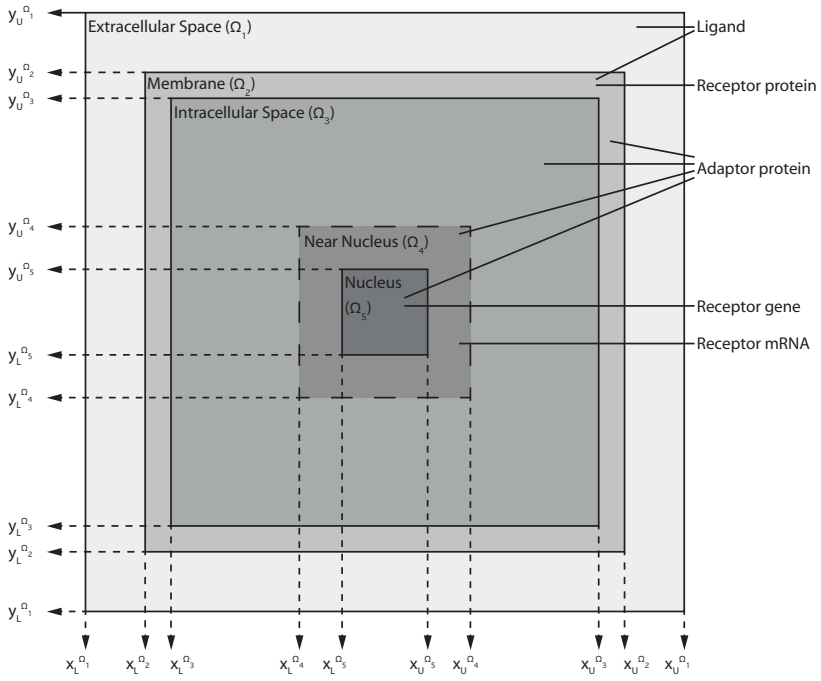


Figure 3.23: Definition of nested compartments in the grid defined for the spatial model of the running example. The grid is divided into five nested compartments: extracellular space (Ω_1), membrane (Ω_2), intracellular space (Ω_3), near nucleus (Ω_4), nucleus (Ω_5). The ligand can stay in the extracellular space and access the membrane region. The receptor protein is only allowed to stay in the membrane. The adaptor protein can move in all compartments inside the membrane. The receptor mRNA is allowed to stay in the region near the nucleus, and the receptor gene stays inside the nucleus. *Note:* logical nodes have been used for a decluttered representation.

Running Example. According to the initial assumption, all components can move over the entire grid. In the biomolecular system, the movement of components might be restricted by the compartmentalization of a cell, e.g. ligands of membrane receptors stay in the extracellular space, membrane receptors themselves are located in the membrane, signal molecules downstream of membrane receptors move inside the cell, and genes are part of the DNA that stays in the

nucleus. For the running example, we assume that the grid can be divided into five nested compartments (see also Figure 3.23): extracellular space (Ω_1), membrane (Ω_2), intracellular space (Ω_3), near the nucleus (Ω_4), and nucleus (Ω_5). The x-intercept of a compartment Ω_i is defined by $x_L^{\Omega_i}$ and $x_U^{\Omega_i}$, and the y-intercept by $y_L^{\Omega_i}$ and $y_U^{\Omega_i}$.

The ligand is allowed to move in the extracellular space. The receptor protein can only move within the membrane. For the adaptor protein, we assume that it can move in the entire intracellular space including the nucleus, and the region near the nucleus. Both, adaptor protein and ligand are allowed to access the membrane to interact with the receptor. We assume that the receptor mRNA stays in the region near the nucleus, where the nucleus hosts the receptor gene. Due to the assumed compartmentalisation, none of the multi interaction site complexes can exist. Thus, we can neglect the subnetworks introduced in Figure 3.22(J). The firing rates of transitions representing the movement of components as single entities and single interaction site complexes must be multiplied by a Boolean expression to evaluate, whether a component stays in its assigned compartments or not. If the Boolean expression is evaluated to $false$, the respective firing rate is set to zero. The transition cannot fire, if the component is not allowed to move to the new position. The new firing rates are defined as follows:

- receptor protein:
  - $h'(txL\_pR) = \omega_{xL}(pR, \Omega_2, \Omega_3) * h(txL\_pR)$
  - $h'(txU\_pR) = \omega_{xU}(pR, \Omega_2, \Omega_3) * h(txU\_pR)$
  - $h'(tyL\_pR) = \omega_{yL}(pR, \Omega_2, \Omega_3) * h(tyL\_pR)$
  - $h'(tyU\_pR) = \omega_{yU}(pR, \Omega_2, \Omega_3) * h(tyU\_pR)$
- receptor gene:
  - $h'(txL\_gR) = \omega_{xL}(gR, \Omega_5) * h(txL\_gR)$
  - $h'(txU\_gR) = \omega_{xU}(gR, \Omega_5) * h(txU\_gR)$
  - $h'(tyL\_gR) = \omega_{yL}(gR, \Omega_5) * h(tyL\_gR)$
  - $h'(tyU\_gR) = \omega_{yU}(gR, \Omega_5) * h(tyU\_gR)$
- receptor mRNA:
  - $h'(txL\_mR) = \omega_{xL}(mR, \Omega_4, \Omega_5) * h(txL\_mR)$
  - $h'(txU\_mR) = \omega_{xU}(mR, \Omega_4, \Omega_5) * h(txU\_mR)$
  - $h'(tyL\_mR) = \omega_{yL}(mR, \Omega_4, \Omega_5) * h(tyL\_mR)$
  - $h'(tyU\_mR) = \omega_{yU}(mR, \Omega_4, \Omega_5) * h(tyU\_mR)$
- adaptor protein:
  - $h'(txL\_pA) = \omega_{xL}(pA, \Omega_2) * h(txL\_pA)$
  - $h'(txU\_pA) = \omega_{xU}(pA, \Omega_2) * h(txU\_pA)$
  - $h'(tyL\_pA) = \omega_{yL}(pA, \Omega_2) * h(tyL\_pA)$
  - $h'(tyU\_pA) = \omega_{yU}(pA, \Omega_2) * h(tyU\_pA)$
- ligand:
  - $h'(txL\_L) = \omega_{xL}(L, \Omega_1, \Omega_3) * h(txL\_L)$
  - $h'(txU\_L) = \omega_{xU}(L, \Omega_1, \Omega_3) * h(txU\_L)$
  - $h'(tyL\_L) = \omega_{yL}(L, \Omega_1, \Omega_3) * h(tyL\_L)$
  - $h'(tyU\_L) = \omega_{yU}(L, \Omega_1, \Omega_3) * h(tyU\_L)$
- receptor protein and ligand complex:
  - $h'(txL\_pR\_L) = \omega_{xL}(pR, \Omega_2, \Omega_3) * \omega_{xL}(L, \Omega_1, \Omega_3) * h(txL\_pR\_L)$

  - $h'(txU\_pR\_L) = \omega_{xU}(pR, \Omega_2, \Omega_3) * \omega_{xU}(L, \Omega_1, \Omega_3) * h(txU\_pR\_L)$
  - $h'(tyL\_pR\_L) = \omega_{yL}(pR, \Omega_2, \Omega_3) * \omega_{yL}(L, \Omega_1, \Omega_3) * h(tyL\_pR\_L)$
  - $h'(tyU\_pR\_L) = \omega_{yU}(pR, \Omega_2, \Omega_3) * \omega_{yU}(L, \Omega_1, \Omega_3) * h(tyU\_pR\_L)$
- receptor protein and adaptor protein complex:
  - $h'(txL\_pR\_pA) = \omega_{xL}(pR, \Omega_2, \Omega_3) * \omega_{xL}(pA, \Omega_2) * h(txL\_pR\_pA)$
  - $h'(txU\_pR\_pA) = \omega_{xU}(pR, \Omega_2, \Omega_3) * \omega_{xU}(pA, \Omega_2) * h(txU\_pR\_pA)$
  - $h'(tyL\_pR\_pA) = \omega_{yL}(pR, \Omega_2, \Omega_3) * \omega_{yL}(pA, \Omega_2) * h(tyL\_pR\_pA)$
  - $h'(tyU\_pR\_pA) = \omega_{yU}(pR, \Omega_2, \Omega_3) * \omega_{yU}(pA, \Omega_2) * h(tyU\_pR\_pA)$
- adaptor protein and receptor gene complex:
  - $h'(txL\_pA\_gR) = \omega_{xL}(pA, \Omega_2) * \omega_{xL}(gR, \Omega_5) * h(txL\_pA\_gR)$
  - $h'(txU\_pA\_gR) = \omega_{xU}(pA, \Omega_2) * \omega_{xU}(gR, \Omega_5) * h(txU\_pA\_gR)$
  - $h'(tyL\_pA\_gR) = \omega_{yL}(pA, \Omega_2) * \omega_{yL}(gR, \Omega_5) * h(tyL\_pA\_gR)$
  - $h'(tyU\_pA\_gR) = \omega_{yU}(pA, \Omega_2) * \omega_{yU}(gR, \Omega_5) * h(tyU\_pA\_gR)$

The Boolean functions $\omega_{xL}(c, a, b)$, $\omega_{xU}(c, a, b)$, $\omega_{yL}(c, a, b)$, and $\omega_{yU}(c, a, b)$, where $c \in C^G$ defines the component, $a \in \mathbf{\Omega}$ the outer boundary compartment, and $b \in \mathbf{\Omega}$ the inner boundary compartment, are evaluated to 1 (true) or 0 (false) and are defined as follows:

$$\omega_{xL}(c,a,b) = \begin{cases} (((y\_\{c\} >= y_L^a \;\&\; y\_\{c\} <= y_U^a) \;\&\; !(y\_\{c\} >= y_L^b \;\&\; y\_\{c\} <= y_U^b)) \;\&\; (x\_\{c\} > x_L^a \;\&\; x\_\{c\} <= x_U^a)) \;| \\ ((y\_\{c\} >= y_L^b \;\&\; y\_\{c\} <= y_U^b) \;\&\; ((x\_\{c\} > x_L^a \;\&\; x\_\{c\} <= x_U^a - 1) \;\&\; !(x\_\{c\} >= x_L^b \;\&\; x\_\{c\} <= x_U^b + 1))) & \text{if } b \neq \emptyset \\ (y\_\{c\} >= y_L^a \;\&\; y\_\{c\} <= y_U^a) \;\&\; (x\_\{c\} > x_L^a \;\&\; x\_\{c\} <= x_U^a) & \text{if } b = \emptyset \end{cases}$$

$$\omega_{xU}(c,a,b) = \begin{cases} (((y\_\{c\} >= y_L^a \;\&\; y\_\{c\} <= y_U^a) \;\&\; !(y\_\{c\} >= y_L^b \;\&\; y\_\{c\} <= y_U^b)) \;\&\; (x\_\{c\} >= x_L^a \;\&\; x\_\{c\} < x_U^a)) \;| \\ ((y\_\{c\} >= y_L^b \;\&\; y\_\{c\} <= y_U^b) \;\&\; ((x\_\{c\} >= x_L^a \;\&\; x\_\{c\} < x_U^a - 1) \;\&\; !(x\_\{c\} >= x_L^b - 1 \;\&\; x\_\{c\} <= x_L^b))) & \text{if } b \neq \emptyset \\ (y\_\{c\} >= y_L^a \;\&\; y\_\{c\} <= y_U^a) \;\&\; (x\_\{c\} >= x_L^a \;\&\; x\_\{c\} < x_U^a) & \text{if } b = \emptyset \end{cases}$$

$$\omega_{yL}(c,a,b) = \begin{cases} (((x\_\{c\} >= x_L^a \;\&\; x\_\{c\} <= x_U^a) \;\&\; !(x\_\{c\} >= x_L^b \;\&\; x\_\{c\} <= x_U^b)) \;\&\; (y\_\{c\} > y_L^a \;\&\; y\_\{c\} <= y_U^a)) \;| \\ ((x\_\{c\} >= x_L^b \;\&\; x\_\{c\} <= x_U^b) \;\&\; ((y\_\{c\} > y_L^a \;\&\; y\_\{c\} <= y_U^a - 1) \;\&\; !(y\_\{c\} >= y_L^b \;\&\; y\_\{c\} <= y_U^b + 1))) & \text{if } b \neq \emptyset \\ (x\_\{c\} >= x_L^a \;\&\; x\_\{c\} <= x_U^a) \;\&\; (y\_\{c\} > y_L^a \;\&\; y\_\{c\} <= y_U^a) & \text{if } b = \emptyset \end{cases}$$

$$\omega_{yU}(c,a,b) = \begin{cases} (((x\_\{c\} >= x_L^a \;\&\; x\_\{c\} <= x_U^a) \;\&\; !(x\_\{c\} >= x_L^b \;\&\; x\_\{c\} <= x_U^b)) \;\&\; (y\_\{c\} >= y_L^a \;\&\; y\_\{c\} < y_U^a)) \;| \\ ((x\_\{c\} >= x_L^b \;\&\; x\_\{c\} <= x_U^b) \;\&\; ((y\_\{c\} >= y_L^a \;\&\; y\_\{c\} < y_U^a - 1) \;\&\; !(y\_\{c\} >= y_L^b - 1 \;\&\; y\_\{c\} <= y_U^b))) & \text{if } b \neq \emptyset \\ (x\_\{c\} >= x_L^a \;\&\; x\_\{c\} <= x_U^a) \;\&\; (y\_\{c\} >= y_L^a \;\&\; y\_\{c\} < y_U^a) & \text{if } b = \emptyset \end{cases}$$

3.4 BMK - Module Construction

The application of direct engineering and reverse engineering approaches allows generating modules. In general, direct engineering is the most common way to construct models. In Section 3.4.1, we shortly discuss how to construct modules by applying direct engineering approach. Reverse engineering approaches allow generating models from experimental data sets. In Section 3.4.2, we show how to integrate such data into modules. The reverse engineering approach is often inevitable, especially in the case of genome-wide (omics) data analysis, where molecular details on regulatory mechanisms are simply not known. In this context, the generation of gene modules by transforming Boolean models into Petri net modules is particularly powerful, see Section 3.4.3 for details. But also SBML models are a valuable source for the generation of gene, mRNA, protein, and protein degradation modules by algorithmically decomposing them into modules, see Section 3.4.4.

3.4.1 Direct Engineering Approach

The most convenient way to construct modules of any type is the application of a direct engineering approach. Therefore, collected knowledge about a component, its molecular mechanisms, and kinetics are translated in modules of appropriate type. The information used to construct modules can be taken from, e.g. literature, relevant biomolecular databases as given in Table 3.8, or experimental data. In this case, experimental data might include data on:

- molecular structures of components and their conformational changes due to regulatory mechanisms;
- interactions with other components;
- molecular mechanisms;
- gene, mRNA and protein expression;
- kinetics;
- ...

Certainly, hypotheses on molecular mechanisms or kinetics can be integrated into a module as well and marked as such in the BMKML module annotation. In Chapter 5, we provide more case studies for directly engineered modules.

This section requires:

- *Petri nets [82], Section 2.1*
- *Module definition, Section 3.1.1*
- *Module types, Section 3.1.2*

3.4.2 Reverse Engineering Approach

Reverse engineered modules allow the fully automated generation of models from complex experimental data sets for the application in genome-wide (omics) approaches. This allows to employ and account for genotype/phenotype relationships for modelling and simulation. For example, reverse engineered modules of gene expression data can directly be linked to bottom-up models of protein-protein interactions. Here, we introduce approaches to integrate reverse engineered

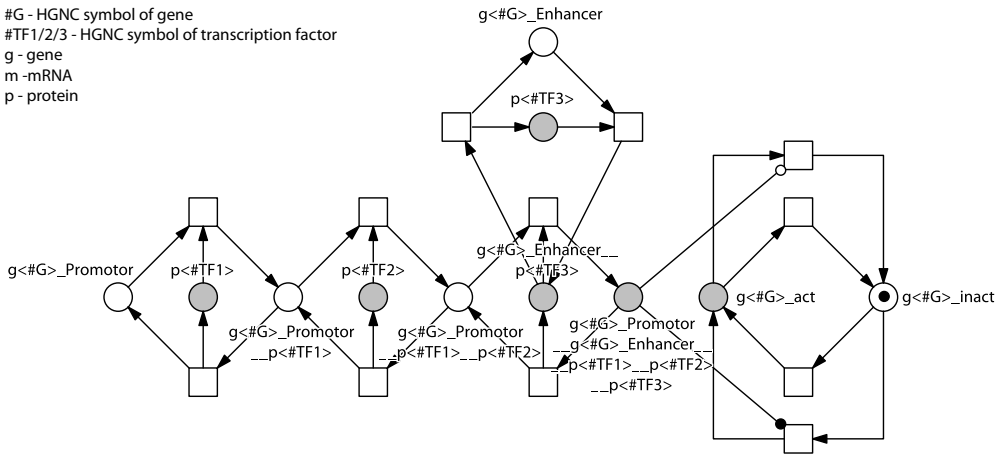


Figure 3.24: Gene prototype modules for eukaryotes. The prototype module of a gene shown here defines the regulation of the gene activity by transcription factors. First, the transcription factor *p<#TF1>* binds to the gene promoter *g<#G>_Promoter* forming the complex *g<#G>_Promoter__p<#TF1>*. Next, the transcription factor *p<#TF2>* binds to the previous complex forming the complex *g<#G>_Promoter__p<#TF1>__p<#TF2>*. The transcription factor *p<#TF3>* binds to the enhancer element of the gene *g<#G>_Enhancer* forming the complex *g<#G>_Enhancer__p<#TF3>*. The two complexes form the transcription complex *g<#G>_Enhancer__g<#G>_Promoter __p<#TF1>__p<#TF2>__p<#TF3>* (all of the bindings are reversible). Th transcription complex triggers the gene to switch from its transcriptionally inactive state *g<#G>_inact* to the transcriptionally active state *g<#G>_act*, and inhibits the deactivation of the gene. There are also two molecular events representing the basal activation and inactivation of the gene. *Note:* Logical nodes indicate interface networks used to connect the gene module with corresponding protein and mRNA modules. (adapted from [116])

experimental data by (1) using prototype modules to model genes and mRNAs and (2) generating allelic and causal influence modules.

Reverse Engineering using Gene/mRNA Prototype Modules

Because of their standardised, generally applicable format, gene and mRNA prototype modules permit the fully automated generation of modules, e.g. from a list of gene names [116]. Figure 3.24 and 3.25 show examples for gene and mRNA prototype modules valid for eukaryotes. In the standard cases, it might be sufficient to match gene names to the pre-defined prototype modules. Otherwise, if prototype modules need to be customized to represent individual regualory mechanisms, a list with all relevant information needs to be algorithmically processed. Such a list might contain information on e.g. transcription factors, splice factors or other regulatory proteins etc., next to the list of gene and mRNA names. According to the provided information places and transitions need to be added to describe additional components and molecular events involved in the regulatory processes of the gene or mRNA.

Such approaches allow the automatic construction of models representing hundreds or even thousands of genes, their mRNAs and the mere presence of the proteins they form. Through the possibility to import e.g. transcriptomic or proteomic data sets obtained in high-throughput experiments [115], regulatory mechanisms can be reverse engineered based on the modular modelling concept. A modularly composed model obtained this way may predict changes in the proteome in response to differential gene regulation. Such models might also be useful to support the interpretation of phenomena observed in systematic RNAi screens, where individual genes are knocked down [64, 96, 98].

Reverse Engineering using Allelic/Causal Influence Modules

Allelic influence modules differ from gene modules in representing the regulatory effects exerted by an allele on cellular processes by controlling the firing activities of respective transitions through read or inhibitory arcs in the Petri net graph. Regarding molecular mech-

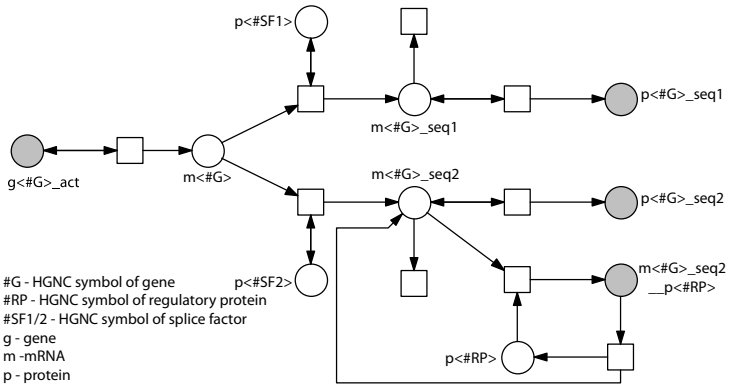


Figure 3.25: mRNA prototype modules for eukaryotes. The mRNA prototype module defines the synthesis of the mRNA (primary transcript) if its gene is transcriptionally active. Dependent on splice factors, the primary transcript is processed into mature transcripts. Transcripts may also interact with other regulatory proteins. Here, the transcriptionally active state of a gene g<#G>_*act* triggers the synthesis of an mRNA m<#G>. Splice factor p<#*SF*1> triggers the processing of the mRNA m<#G> into the product m<#G>_*seq*1, which can either be degraded or start the synthesis of protein p<#G>_*seq*1. Splice factor p<#*SF*2> triggers the processing of the mRNA m<#G> into the product m<#G>_*seq*2, which can either be degraded, start the synthesis of protein p<#G>_*seq*2, or interact with a regulatory protein p<#*RP*> to form a complex m<#G>_*seq*2__p<#*RP*> (reversible). In this state, the processed mRNA cannot be translated. *Note:* Logical nodes indicate interface networks used to connect the mRNA module with corresponding protein and gene modules. (adapted from [116])

anisms, regulatory influences might be rather indirect by involving numerous other, potentially unknown components. Accordingly, the allelic influence module may represent the control of molecular events like the biosynthesis of RNA by transcription or even more complex processes of potentially unknown molecular mechanism as inferred from functional studies. Figure 3.26 shows, how a mutated allele of a gene might affect the regulation of other genes. In this context, causal influence modules might represent all other kinds of factors like environmental conditions or individual correlations among components, that have a known impact on certain molecular events without being mechanistically defined.

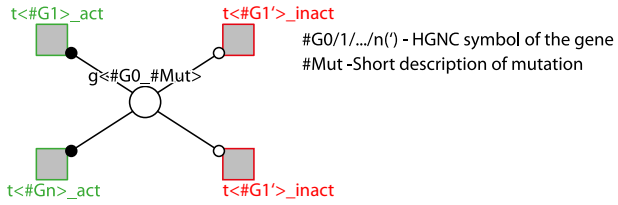


Figure 3.26: Allelic influence modules. The allelic influence module indicates the effect of the mutated allele of gene g<#G1_#*Mut*> on the (de-)activation of other genes (red transitions with inhibitory arcs - inhibition; green transitions with read arcs - activation). The regulatory effect also affects the expression of the gene-specific mRNAs. (adapted from [116])

3.4.3 *Transformation of Boolean Gene Regulatory Network Models into Modules and a Modular Network*

This section requires:

- *Petri nets [82], Section 2.1*
- *Boolean networks [8, 23], see text*
- *Module definition, Section 3.1.1*
- *Petri net representation of modules, Section 3.1.3*
- *Modular model composition, Section 3.3.1*

In systems biology, Boolean models are widely used to describe gene regulatory networks qualitatively to understand the diverse functionalities of a given network under different conditions [83]. It has already been shown, that Petri nets offer another complementary framework to express gene regulatory networks by systematically rewriting the logical expressions of the Boolean model into a Petri net model [46]. The rigorous transformation preserves the representation of dynamical roles of specific feedback structures in a gene regulatory network and facilitates the application of algebraic tools in the Petri net framework to study fundamental dynamical properties [74, 76].

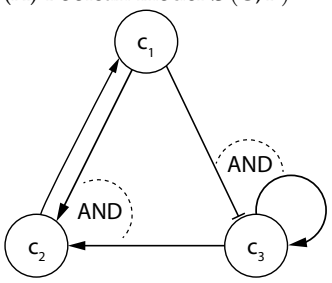
The modular Petri net transformation of gene regulatory networks is based on the formal concepts described in [74, 76], who apply the Quine-McClusky algorithm and logic minimization in their transformation. To be consistent with the modular idea of the BMKfr, we represent the regulation of each gene involved in a specific gene

regulatory network as a Boolean gene module, which can be interpreted as a special type of gene module as defined in Section 3.1.2. Using the concept of the modular model composition described in Section 3.3, the set of generated Boolean gene modules reconstitutes the gene regulatory network that served as input. Also, due to the principle of modular compositionality, the generated Boolean gene modules can also be reused and recombined in any other combination of modules with matching interface networks. Thus, the modular Petri net transformation of gene regulatory networks increases the universal applicability of the BMKfr, but also facilitates the reuse of gene regulatory networks and their integration with other network types.

Algorithms for the automatic transformation of Boolean-type gene regulatory networks into modularly composed Petri net models have been elaborated and implemented in the master thesis of Lisa Jehrke [134], which was supervised as part of this thesis. An additional important part of her master thesis was the implementation of algorithms for the mutation of gene regulatory networks as originally performed by Krumsiek et al. [113] for the Petri net formalism. As has already been discussed in Section 3.3.3, the model mutation algorithms allow the systematic and exhaustive generation of *in silico* mutation experiments to investigate the effect of perturbations on the behaviour of components, which often interact in a complicated and non-obvious manner. These analyses are especially valuable to predict phenotypes expected to be obtained in mutational screens performed in the wet-lab.

Before specifying the details of the transformation of gene regulatory networks into modularly composed Petri net model and the mutation algorithms, we will briefly recapitalize the formal description of gene regulatory networks as well as the Quine-McClusky algorithm. For demonstration purposes, we refer to the case study of hierarchical differentiation of myeloid progenitors from Krumsiek et al. [113], which has been adopted in the master thesis of Lisa Jehrke [134]. The Boolean model of the hierarchical differentiation of myeloid progenitors from Krumsiek et al. [113] describes the differentiation of a hematopoietic stem cell (HSC) to a common myeloid progenitor cell (CMP), which proliferates into megakaryocyte-erythrocyte (MegE) progenitors and granulocyte-monocyte (GM) progenitors, which further give rise to megakaryocytes, erythrocytes, granulocytes, monocytes and others. The modules obtained from the modular Petri net transformation of a model describing the hierarchical differentiation of myeloid progenitors from Krumsiek et al. [113] is given in the running example at the end of this section, see also Figure 3.31 for the resulting state space of the modularly composed model.

(A) Boolean model $\mathcal{B}(C, F)$



$$c_1 = c_2$$
$$c_2 = c_1 \cdot c_3$$
$$c_3 = \overline{c_1} \cdot c_3$$

(B) Truth Table

τ			$\tau+1$		
c_1	c_2	c_3	c_1'	c_2'	c_3'
0	0	0	0	0	1
0	0	1	0	0	1
0	1	0	1	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	1	1	0

Figure 3.27: Boolean model. (A) Simple example of a Boolean model $\mathcal{B}(C, F)$ with three components c_1, c_2, c_3 represented as network graph and set of equations. In the network graph, components with a positive regulatory effect (activation) are indicated by $\longrightarrow$ and components with a negative regulatory effect (inhibition) are indicated by $\dashv$. The dashed arch connecting two arcs represent the Boolean conjunction operator *AND*. (B) The truth table shows the dynamic evolution of the model for each possible initial state ($2^3 = 8$).

Boolean Network Models of Gene Regulatory Networks

A Boolean network $\mathcal{B}(C, F)$ [8, 23] is defined as a set of n binary-valued nodes (components, here in particular genes) $C = \{c_1, \ldots, c_n\}$, $c_i \in \{0, 1\}$, and a list of Boolean functions $F = (f_1, \ldots, f_n)$. Each node c_i has k_i parent nodes (regulators) chosen from C, and its value at time $\tau + 1$ is determined by its parent nodes at τ through a Boolean function f_i, $c_i(\tau + 1) = f_i(C_i'(\tau))$, $C_i' \subseteq C$, see also Figure 3.27. If component $c_i \in C_i'$, then f_i includes the auto-regulation of component c_i.

The notation $\bar{c}$, $c + c'$ and $c \cdot c'$ represent the Boolean operators *NOT*, *OR*, and *AND* [81]. Semantically, the dynamic behaviour of a Boolean model can be interpreted as either asynchronous, where genes update their state independently; or synchronous where all genes update their state together [33]. In our approach, we focus on the asynchronous behaviour.

In contrast to the Petri net representation of a gene regulatory network model, the Boolean network graph is only definite if a set of Boolean equations is provided.

[10] QMC algorithm has been developed by W.V. Quine and extended by Edward J. McCluskey [2, 3]

Quine-McClusky Algorithm

The Quine-McCluskey algorithm[10] is a minimization method for Boolean functions to obtain prime implicants. In general, a "covering" (sum term or product term) of one or more minterms in a sum of products (or max terms in a product of sums) of a Boolean function is called an implicant. More formally, an implicant of the Boolean function f (Boolean function with n variables) is given by a product term I in a sum of products. The implicant I (product term) implies f. The Boolean function f takes the value 1 (0) whenever I is 1 (0). A prime implicant of a Boolean function f is an implicant that cannot be covered by a more general (more reduced - meaning with fewer literals) implicant. According to W.V. Quine a prime implicant of a Boolean function f is defined as an implicant that is minimal - that is, the removal of any literal from I results in a non-implicant for the Boolean function f.

In the first step of the Quine-McCluskey algorithm, the prime implicants of the Boolean function are determined. Afterwards, the prime implicant chart allows identifying essential prime implicants

of the Boolean function, as well as other prime implicants that are necessary to cover the function, resulting into a list of final prime implicants $\mathbf{Pr} = \{Pr_i \mid i = 1, \dots n\}$, of the Boolean function f, where $Pr_i = \prod_{c \in C'_i} \begin{cases} c & \text{, if } c = 1 \\ \bar{c} & \text{, if } c = 0 \end{cases}$, $C'_i \subseteq C'$.

Modular Petri Net Transformation of Boolean Networks

Let $\mathcal{B}(C, F)$ be a Boolean model of a gene regulatory network, each component (gene) $c_i \in C$ and its related Boolean function $c_i = f_i(C'_i)$, $C'_i \subseteq C$ and $f_i \in F$ yield one Boolean gene module $M_{g,c_{0,i}}$ ($c_{0,i} = c_i$) with its Petri net representation $\mathcal{N}(M_{g,c_{0,i}})$. To transform the Boolean function $c_{0,i} = f_i(C'_i)$ into a Petri net, the Quine-McClusky algorithm has to be applied to determine the final list of prime implicants $\mathbf{Pr}_{c_{0,i}}$, which can be divided into two list of prime implicants, see also Figure 3.28:

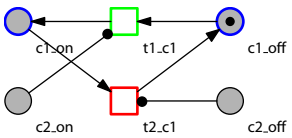
- activating component $c_{0,i}$ (positive regulation), where $f_i(C'_i) = 1$: $\mathbf{Pr}_{c_{0,i},1} = \{Pr_j^{c_{0,i}} \in \mathbf{Pr}_{c_{0,i}} \mid Pr_j^{c_{0,i}} \to f_i(C'_i) = 1\}$, and
- inactivating component $c_{0,i}$ (negative regulation), where $f_i(C'_i) = 0$: $\mathbf{Pr}_{c_{0,i},0} = \{Pr_j^{c_{0,i}} \in \mathbf{Pr}_{c_{0,i}} \mid Pr_j^{c_{0,i}} \to f_{C'_i} = 0\}$

Accordingly, the Petri net graph of the Boolean gene module $\mathcal{N}(M_{g,c_{0,i}}) = \{P, T, F, f, v, m_0\}$ can be defined as follows:

- Set of places $P = P_{on} \cup P_{off}$, where
  - $P_{on} = \{p_{on}^{c_{0,i}}, \bigcup_{c \in C'_i} p_{on}^{c}\}$, and
  - $P_{off} = \{p_{off}^{c_{0,i}}, \bigcup_{c \in C'_i} p_{off}^{c}\}$
- Set of transitions $T = T_{on} \cup T_{off}$, where
  - $T_{on} = \bigcup_{Pr_j^{c_{0,i}} \in \mathbf{Pr}_{c_{0,i},1}} t_{on}^{j}$, and
  - $T_{off} = \bigcup_{Pr_j^{c_{0,i}} \in \mathbf{Pr}_{c_{0,i},0}} t_{off}^{j}$
- Set of arcs $F = F_{SA} \cup F_{RA}$, where
  - $F_{SA} \subseteq (P \times T) \cup (T \times P)$
  - $F_{RA} \subseteq (P \times T)$
- Arc-weights f: $F \to \mathbb{N}_0$
  - $\forall t_{on}^{j} \in T_{on}$: $f_{SA}(p_{off}^{c_{0,i}}, t_{on}^{j}) = 1$ and $f_{SA}(t_{on}^{j}, p_{on}^{c_{0,i}}) = 1$
  - $\forall t_{off}^{j} \in T_{off}$: $f_{SA}(p_{on}^{c_{0,i}}, t_{off}^{j}) = 1$ and $f_{SA}(t_{off}^{j}, p_{off}^{c_{0,i}}) = 1$
  - $\forall t_{on/off}^{j} \in T$ with $Pr_j^{c_{0,i}}$:
    - $\forall c \in C'_j$: $\begin{cases} f_{RA}(p_{on}^{c}, t_{on/off}^{j}) = 1 & \text{, if } c = 1 \\ f_{RA}(p_{off}^{c}, t_{on/off}^{j}) = 1 & \text{, if } c = 0 \end{cases}$, $C'_i \subseteq C'$
- Firing rates v: $T \to H$, $H = \bigcup_{t \in T} h(t)$
  - $\forall t \in T : h(t) = 1$
- Initial marking m_0: $P \to \mathbb{N}_0$
  - $p_{off}^{c_{0,i}} \in P$: $m_0(p_{off}^{c_{0,i}}) = 1$,

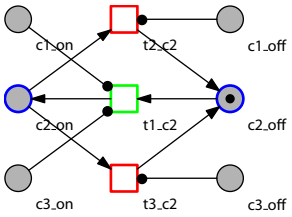
(A) Boolean Gene Module $M_{g,c1}$

$$c_1 = c_2 \begin{cases} \mathbf{Pr}_{c_1,1} = \overbrace{c_2}^{Pr_1^{c_1}} \\ \mathbf{Pr}_{c_1,0} = \underbrace{\overline{c_2}}_{Pr_2^{c_1}} \end{cases}$$



(B) Boolean Gene Module $M_{g,c2}$

$$c_2 = c_1 \cdot c_3 \begin{cases} \mathbf{Pr}_{c_2,1} = \overbrace{c_1 \cdot c_3}^{Pr_1^{c_2}} \\ \mathbf{Pr}_{c_2,0} = \underbrace{\overline{c_1}}_{Pr_2^{c_2}} + \underbrace{\overline{c_3}}_{Pr_3^{c_2}} \end{cases}$$



(C) Boolean Gene Module $M_{g,c3}$

$$c_3 = \overline{c_1} \cdot c_3 \begin{cases} \mathbf{Pr}_{c_3,1} = \overbrace{\overline{c_1} \cdot c_3}^{Pr_1^{c_3}} \\ \mathbf{Pr}_{c_3,0} = \underbrace{c_1}_{Pr_2^{c_3}} + \underbrace{\overline{c_3}}_{Pr_3^{c_3}} \end{cases}$$

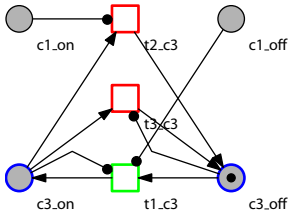


Figure 3.28: Modular Petri net transformation of a Boolean network. The Boolean model of Figure 3.27 has been transformed into three modules in the form of Petri nets for components $c1$, $c2$ and $c3$. Transitions with a green outline represent prime implicants with positive regulation, whereas transitions with a red outline represent prime implicants with negative regulation. Places outlined in blue represent the main component that is regulated in each module. *Note:* Logical nodes are used for a decluttered representation of the modular composed model of the Boolean gene modules of components $c1$, $c2$ and $c3$.

- $\forall p \in P, p \neq p_{off}^{c_{0,i}}$: $m_0(p) = 0$

The set of Boolean gene modules $G = \{M_{g,c_{0,1}}, \ldots, M_{g,c_{0,n}}\}$ generated from the Boolean model $\mathcal{B}(C, F)$ of a gene regulatory network, can be committed to the modular model composition explained Section 3.3.1. To experience the behaviour of the model, the initial marking of the modular composed model $\mathcal{N}(G)$ needs to be systematically pertubated, under the assumption that $\forall c \in C$: $m_0^G(p_{on}^c) + m_0^G(p_{off}^c) = 1$, which yields 2^n different markings according to the state space of the Boolean model [23]. Also, the generated Boolean gene modules can also be recombined with other modules.

Boolean Model Mutation

Krumsiek et al. [113] suggested three types of mutations of Boolean models: factor knock-out, factor over-expression, and interaction knock-out. We analogously implemented the suggested model mutation types for the Petri net representation of Boolean models:

1. The factor knock-out is performed in the Boolean model $\mathcal{B}(C, F)$ by setting the Boolean equation of a component $c_i \in C$ to zero, such that $f_i = 0 \rightarrow c_i = 0$. In the Petri net representation, we have to exclude the respective Boolean gene module $M_{g,c_i} \in G$ before the modular model composition, similar to Algorithm 3.2, and we also need to ensure that in the composed model $\mathcal{N}(G_{KO})$, $G_{KO} = G \setminus M_{g,c_i}$, the initial marking of $m_0^G(p_{on}^c) = 0$ and $m_0^G(p_{off}^c) = 1$, see Figure 3.29(A).
2. The factor over-expression is performed in the Boolean model $\mathcal{B}(C, F)$ by setting the Boolean equation of a component $c_i \in C$ to

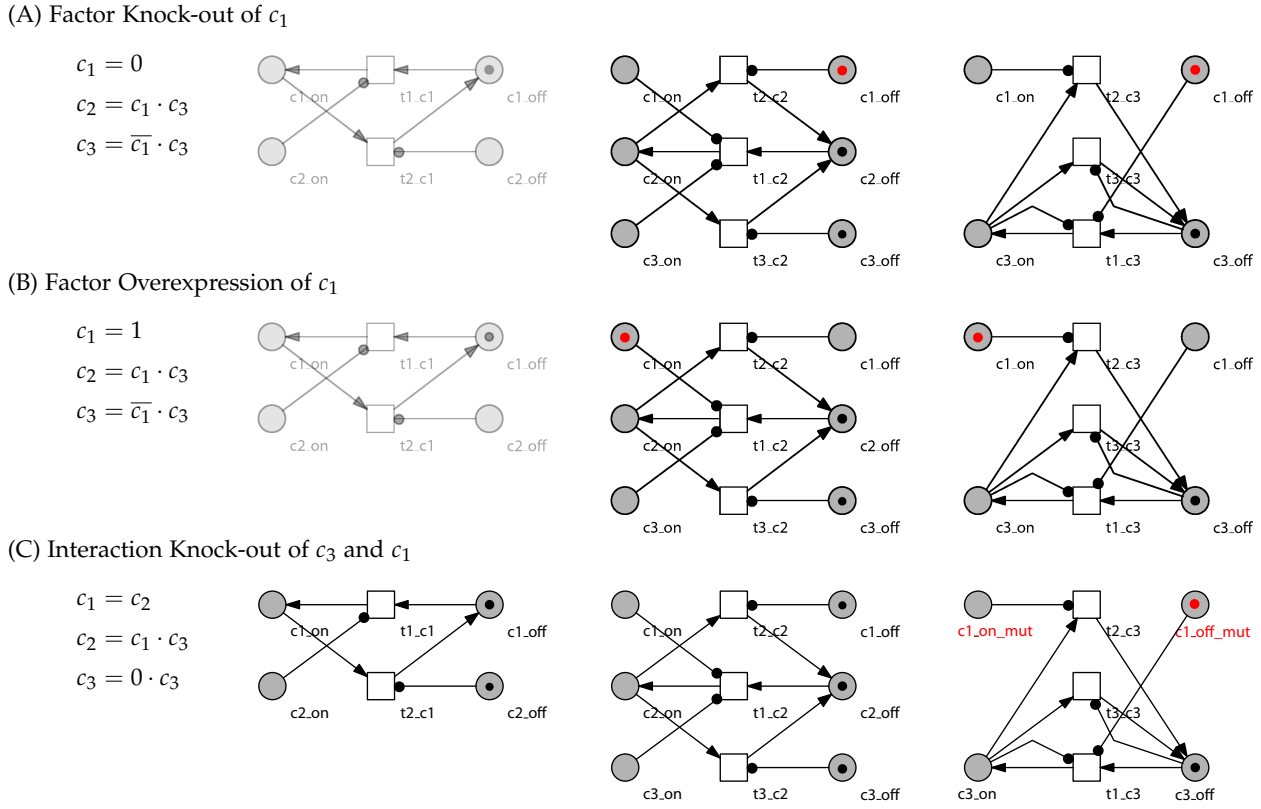


Figure 3.29: Boolean model mutation. The Figure shows how to perform the Boolean model mutation on the Petri net level. (A) The factor knock-out of $c1$ in the Boolean model requires to delete the module of M_{g,c_1} (faded module) and to set the marking of place $c1\_off$ to 1 (red tokens) in the composed model. (B) The factor over-expression of $c1$ in the Boolean model requires to delete the module of M_{g,c_1} (faded module) and to set the marking of place $c1\_on$ to 1 (red tokens) in the composed model. (C) The interaction knock-out of $c3$ and $c1$ requires to rename the places of component $c1$ in module of M_{g,c_3} ($c1\_on\_mut$,$c1\_off\_mut$, red labels) and to to set the marking of place $c1\_off\_mut$ to 1 (red token). For each mutation, the tokens indicated in red are set according to the requirements of the specific mutation scenarios as described in text. Due to the resulting structure of the modules, the marking of places holding the red token is permanently fixed to 1. *Note:* Logical nodes are used for a decluttered representation and to distinguish the Boolean gene modules of components $c1$, $c2$ and $c3$.

one, such that $f_i = 1 \rightarrow c_i = 1$. In the Petri net representation, we have to exclude the respective Boolean gene module $M_{g,c_i} \in G$ in the modular model composition, similar to Algorithm 3.2, and we also need to ensure that in the composed model $\mathcal{N}(G_{OE})$, $G_{OE} = G \setminus M_{g,c_i}$, the initial marking of $m_0^G(p_{on}^c) = 1$ and $m_0^G(p_{off}^c) = 0$, see Figure 3.29(B).

3. INTERACTION KNOCK-OUT in a Boolean model $\mathcal{B}(C, F)$ between to components $c_i, c_j \in C$, where $c_j \in C_i$ is performed by setting component c_j in the Boolean equation f_i to zero. In the Petri net representation, we have to modify the respective Boolean gene module $M_{g,c_i} \in G$ in the modular model composition, such that place $p_{on}^{c_j}$ is renamed to $p_{on}^{c_{j,mut}}$ and place $p_{off}^{c_j}$ is renamed to $p_{off}^{c_{j,mut}}$. Afterwards, the marking of the renamed places is set to $m_0^G(p_{on}^{c_{j,mut}}) = 0$ and $m_0^G(p_{off}^{c_{j,mut}}) = 1$. The renaming ensures, that marking of $p_{on/off}^{c_{j,mut}}$ is fixed, but only affects the respective interaction in module M_{g,c_i}, see Figure 3.29(C).

Running Example. To demonstrate the modular Petri net transformation of gene regulatory networks, we investigated the Boolean model of hierarchical differentiation of myeloid progenitors controlled by mutual regulation of transcription factors from Krumsiek et al. [113]. All mature blood cells (megakaryocytes, erythrocytes, granulocytes and monocytes) emerge through a hierarchical series of lineage decisions via different progenitor cells from a single hematopoietic stem cell (HSC). The common myeloid progenitor (CMP) is one of

(A) Boolean Equation

$$GATA-2 = GATA-2 \cdot \overline{(GATA-1 \cdot FOG-1)} \cdot \overline{PU.1}$$
$$GATA-1 = (GATA-1 + GATA-2 + Fli-1) \cdot \overline{PU.1}$$
$$FOG-1 = GATA-1$$
$$EKLF = GATA-1 \cdot \overline{Fli-1}$$
$$Fli-1 = GATA-1 \cdot \overline{EKLF}$$
$$SCL = GATA-1 \cdot \overline{PU.1}$$
$$CEBP\alpha = CEBP\alpha \cdot \overline{(GATA-1 \cdot FOG-1 \cdot SCL)}$$
$$PU.1 = (CEBP\alpha + PU.1) \cdot \overline{GATA-1 + GATA-2}$$
$$cJun = PU.1 \cdot \overline{Gfi-1}$$
$$ErgNab = (PU.1 \cdot cJun) \cdot \overline{Gfi-1}$$
$$Gfi-1 = CEBP\alpha \cdot \overline{EgrNab}$$

(B) Boolean Gene Modules

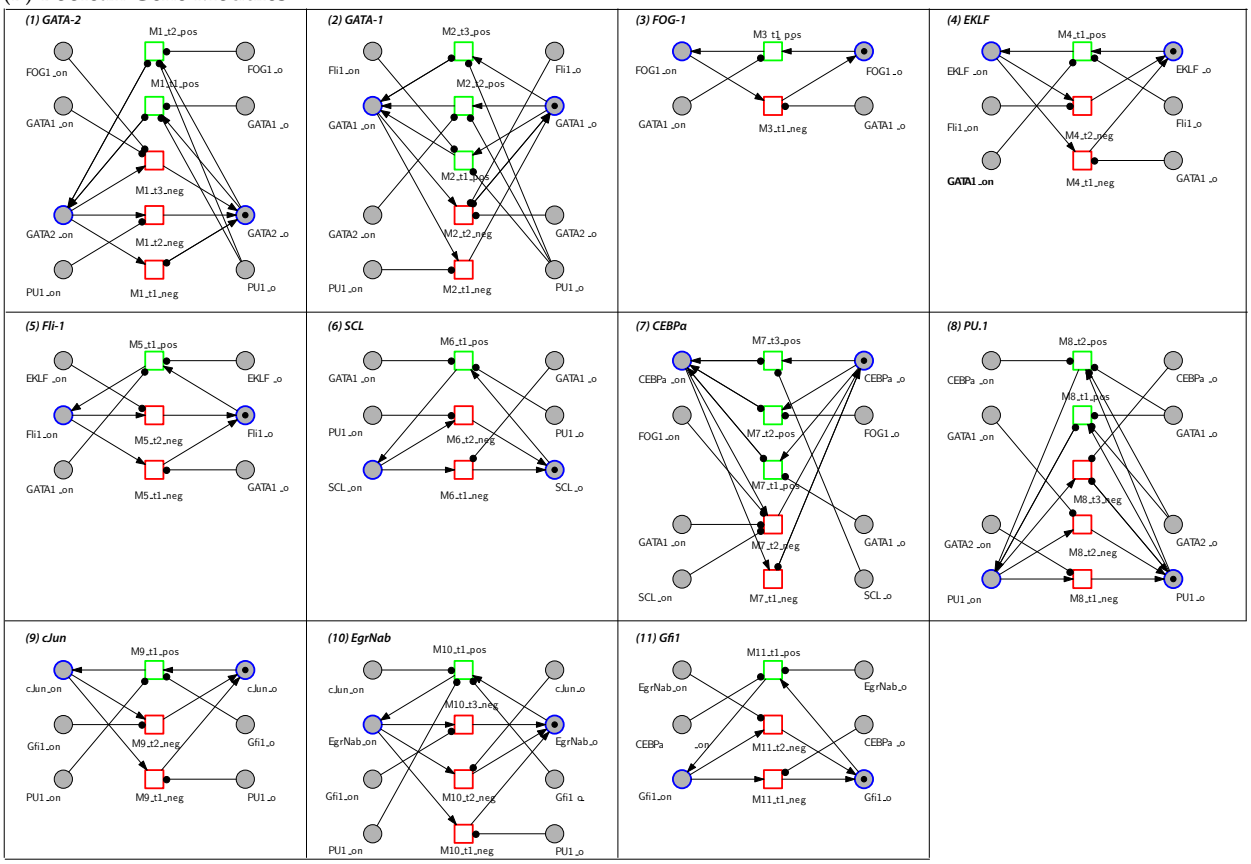


Figure 3.30: (A) Boolean model for hierarchical differentiation of myeloid progenitors, describing the interaction and behaviour of 11 genes. (B) Boolean gene modules of the Boolean model shown in (A). For each gene defined by its corresponding Boolean equation one Boolean gene module has been generated. Transitions with a green outline represent prime implicants with positive regulation, whereas transitions with a red outline represent prime implicants with negative regulation. Places outlined in blue represent the main component that is regulated in each module. (adapted from [134])

the intermediate cellular states in the differentiation process that proliferates into megakaryocyte-erythrocyte (MegE) progenitors and granulocyte-monocyte (GM) progenitors, which further give rise to megakaryocytes, erythrocytes, granulocytes, monocytes and others.

Experimental setups unravelled several important transcription factors that control the differentiation process of myeloid progenitors, see [69] for a review. The model of the hierarchical differentiation of myeloid progenitors has been initially generated by investigating potential regulatory interactions proposed by the Bibliosphere [56] text-mining tool, see [113] for detailed instructions and references used to construct the model. The Boolean equations in Figure 3.30(A) define the model of the hierarchical differentiation of myeloid pro-

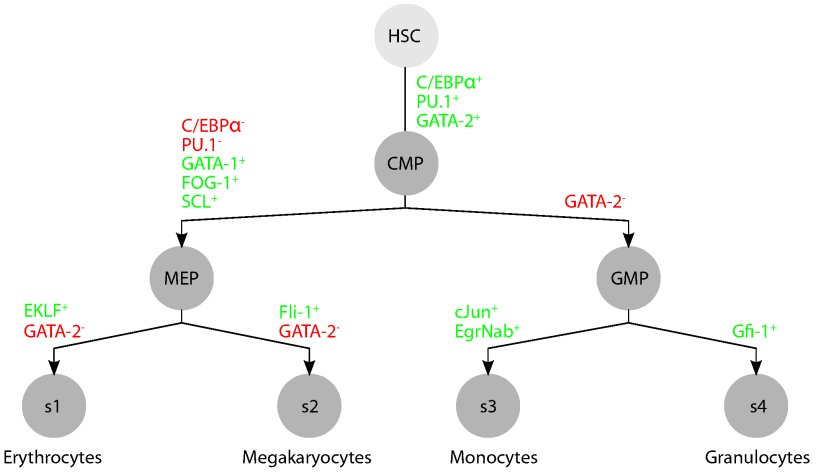


Figure 3.31: State space representing the hierarchical differentiation of myeloid progenitors. As described in [113], we observed a hierarchical partitioning with subsequent splits between the granulocyte-monocyte (GMP) and megakaryocyte-erythrocyte (MEP) lineages from the hematopoietic stem cells (HSP) over the common myeloid progenitor lineages (CMP), followed by splits of the granulocyte and monocyte lineages, and the erythrocyte and megakaryocyte lineages, respectively. Arrows in the diagram represent expression changes on the respective branch of the differentiation tree. Components highlighted in green have to be active, where components highlighted in red have to be inactive. (adapted from [134])

genitors [113].

Each Boolean function represents the regulation of one particular component and defines a Boolean gene module, see Figure 3.30(B). By composing the modules shown in Figure 3.30(B) to the modular model composition as described in Section 3.3.1, we obtained the composed modular Petri net representation of the Boolean model for the regulation of myeloid progenitor differentiation as shown above.

With eleven different genes that regulate each other, the state space for each possible initial marking ($2^{11} = 2048$) of the composed modular Petri net model has been computed with Marcie [129]. For analysis purposes, we focused on the early, unstable undifferentiated state, where only GATA-2, C/EBPa, and PU.1 are active (see [66, 85] for experimental evidence) as Krumsiek et al. [113]. We were able to reproduce the state space of the Boolean network as described by Krumsiek et al. [113] with its five non-trivial attractors, of which four attractors are thought to represent the cell lineages of granulocyte, monocyte, erythrocyte and megakaryocyte, see Figure 3.31. The fifth attractor is not of relevance in this work; it cannot be reached from the assumed early state as described above.

By applying the model mutation for factor knock-out, factor over-expression and interaction knock-out on a Petri net level, we could also confirm the results reported by Krumsiek et al. [113]; details are documented in [134].

3.4.4 *Modular Transformation of SBML Models*

In systems biology, many models are described as a system of ordinary differential equations (ODEs) and are encoded in systems biology mark-up language (SBML) [40], which is a common standard format to store models of biological systems [48]. Thus, there already exists a large number of SBML models. A majority of them are stored in model repositories like KEGG [27], BioModels Database [101], and cellML [37] to name only a few. In the context of our BMKFR, SBML models are a valuable source for the generation of modules. But be-

This section requires:

- *Petri nets [82], Section 2.1*
- *XML [22]*
- *SBML[40], see text*
- *Module definition, Section 3.1.1*
- *Module types, Section 3.1.2*
- *Petri net representation of modules, Section 3.1.3*
- *Modular model composition, Section 3.3.1*

fore integrating an SBML model into the BMKfr, the model needs to be transformed into modules according to definitions in Section 3.1.1. Since SBML models might describe and integrate different biological network types, the transformation might generate gene, mRNA, protein and protein degradation modules. Modules that have been generated from an SBML model can be integrated into the BMKfr, and can thus also be recombined with other modules. The modular Petri net transformation of SBML models increases the universal applicability of the BMKfr, but also facilitates the reuse of SBML models and their integration into new modularly composed models.

Algorithms for the automatic transformation of SBML models into Petri net modules have been elaborated and implemented in the master thesis of Maxi Soldmann [136], which was supervised as part of this thesis.

Before, specifying the details of the transformation of SBML models into sets of Petri net modules, we will briefly recapitalize the structure of SBML to explain subsequently the steps of the SBML preprocessing and the decomposition into modules.

SBML

SBML is an XML [22] based data exchange format, which allows to store and exchange models of molecular networks of any desired complexity. Each element (components, reactions, compartments, functions, units, etc.) part of the modelled molecular network is delineated by a specific data object, that keeps track of all relevant information about the element. The sbml root element contains the child element model, which itself might include one or more of the lists specified by the element ListOf[Object]. Objects defined in SBML are:

- Functions,
- Units,
- CompartmentTypes,
- SpeciesTypes,
- Compartments,
- Species,
- Parameters,
- InitialAssignments,
- Constraints,
- Rules,
- Reactions, and
- Events.

All elements included in the SBML body can be attached with child-elements metadata, note and annotation. In particular the annotation element allows to embed distinct XML specifications, that are not compliant with the SBML format but with other external standards. A detailed documentation of the XML scheme of SBML 3.1., the latest SBML version, can be found in Hucka et al. [142].

Modular Petri Net Transformation of SBML Models

In our interpretation molecular states in the SBML model are represented by the SBML species elements. Since, the modularisation approach of the BMKfr is based on components, each SBML species element must be assigned to a set of components in order to transform

an SBML model into modules. Thus, we defined XML elements of the BMK namespace and appended them to ANNOTATION element in the SBML MODEL and SPECIES elements.

In the ANNOTATION element of the SBML MODEL element, we introduce the COMPLIST element, which holds a number of COMPONENT child-elements, according to the number of components defined in the SBML LISTOFSPECIES element. The COMPONENT element has five attributes, see also Figure 3.32(A):

- **type** specifies the component type: gene, mRNA, protein, non-genetic component or pseudo component (if its neither a genetic, nor a non-genetic component, e.g. pH-value, temperature, artificial components, components without detailed reference, etc.; required).
- **name** specifies the name of the component (required):
  - genetic-components - HGNC symbol [140]
  - non-genetic components - offical PubChem synonym [75]
  - pseudo componets - an abbreviation uniquely describing the component in the context of the current SBML model
- **dbName** specifies the reference database (required):
  - genetic-components - Ensembl [78]
  - non-genetic components - PubChem [75]
  - pseudo components - Pseudo
- **id** specifies a unique id according to the related database and type of component (required):
  - genetic-components - Ensembl id (ENSG... -genes, ENST... - mRNAs, ENSPG... - proteins) [78]
  - non-genetic components - PubChem id (CID) [75]
  - pseudo components - Pseudo id (unique numeric value in the context of current SBML model)
- **iid** specifies the internal identifier (unique numeric value in the context of current SBML model)

In the ANNOTATION element of each SBML SPECIES element, we introduce the COMPLISTOFSPECIES element, which holds a number of COMPOFSPECIES child-elements, according to the number of components defined by the SPECIES element. The COMPOFSPECIES element has a single attribute, see also Figure 3.32(B):

- **iid** specifies the internal identifier, which refers to the component defined in the COMPLIST element.

Each COMPONENT child-element in the COMPLIST element yields one module M. To construct the Petri net graph $\mathcal{N}(M)$, the modified SBML model code needs to be decomposed, see also Algorithm 3.4.4. First, all REACTION child-element in the LISTOFREACTIONS element have to be determined, that involve any species representing a molecular state of the respective component defined by the current COMPONENT element. Each of the determined reactions is translated into a

(A) XSD of compList element

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified" elementFormDefault="qualified" targetNamespace="http://www.biomodelkit.de">
  <xs:element xmlns:BMK="http://www.biomodelkit.de" name="compList" type="BMK:compListType"/>
  <xs:complexType name="compListType">
    <xs:sequence>
      <xs:element xmlns:BMK="http://www.biomodelkit.de" type="BMK:componentType" name="component" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="componentType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="name" use="required"/>
        <xs:attribute name="type" use="required">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="gene"></xs:enumeration>
                    <xs:enumeration value="mRNA"></xs:enumeration>
                    <xs:enumeration value="protein"></xs:enumeration>
                    <xs:enumeration value="non-genetic component"></xs:enumeration>
                    <xs:enumeration value="pseudo component"></xs:enumeration>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="dbName" use="required">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="Ensembl"></xs:enumeration>
                    <xs:enumeration value="PubChem"></xs:enumeration>
                    <xs:enumeration value="Pseudo"></xs:enumeration>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="id" use="required"/>
        <xs:attribute name="iid" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:schema>
```

(B) XSD of compListofSpecies element

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified" elementFormDefault="qualified" targetNamespace="http://www.biomodelkit.de">
  <xs:element xmlns:BMK="http://www.biomodelkit.de" name="compListofSpecies" type="BMK:compListofSpeciesType"/>
  <xs:complexType name="compListofSpeciesType">
    <xs:sequence>
      <xs:element xmlns:BMK="http://www.biomodelkit.de" type="BMK:compOfSpeciesType" name="compOfSpecies" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="compOfSpeciesType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="iid" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:schema>
```

Figure 3.32: XSD of additional SBML elements for the modular Petri net transformation. (A) compList and (B) compListofSpecies element

transition. Afterwards, for each affected reaction, all of the involved species acting as product, educt or modifier have to be detected. Each of the identified species is translated into a place. The initial marking of a place representing a particular species is set to the value of the attribute **initialAmount** of the respective species element. A place p representing a species element that is defined as product (educt) of a particular reaction element given by a transition t are connected by a standard arc $f_{SA}(t,p)$ ($f_{SA}(p,t)$). If the place p representing a species element is defined as modifier of a particular reaction element given by transition t, both are connected by a read arc $f_{RA}(p,t)$. The arc-weight is set to the value of the attribute **stoichiometry** of the reaction element. The firing rate of a transition t representing a reaction element is set according to its child-element kineticLaw. If the value of the attribute **reversible** of the respective reaction element represented by a transition t is set to true, a transition t_{rev} has to be added with reverse educt-product relation. The firing rate of transition t_{rev} (backward reaction) has to be ascertained according to transition t representing the forward reaction. In addition, the specifications made for each module type in Section 3.1.1 and 3.1.3

have to be considered. After the SBML model transformation, each generated module needs to be checked if it is consistent with the structural properties defined for the respective module types, see Section 3.1.3. The validation can be either performed by the user by validating the modules with an analysis tool, e.g. Charlie [141] or Marcie [129], or by the administrator of the BMKDB after submitting the modules. In a similar way, the transformation algorithm generates modules for non-genetic components, called pseudo-modules.

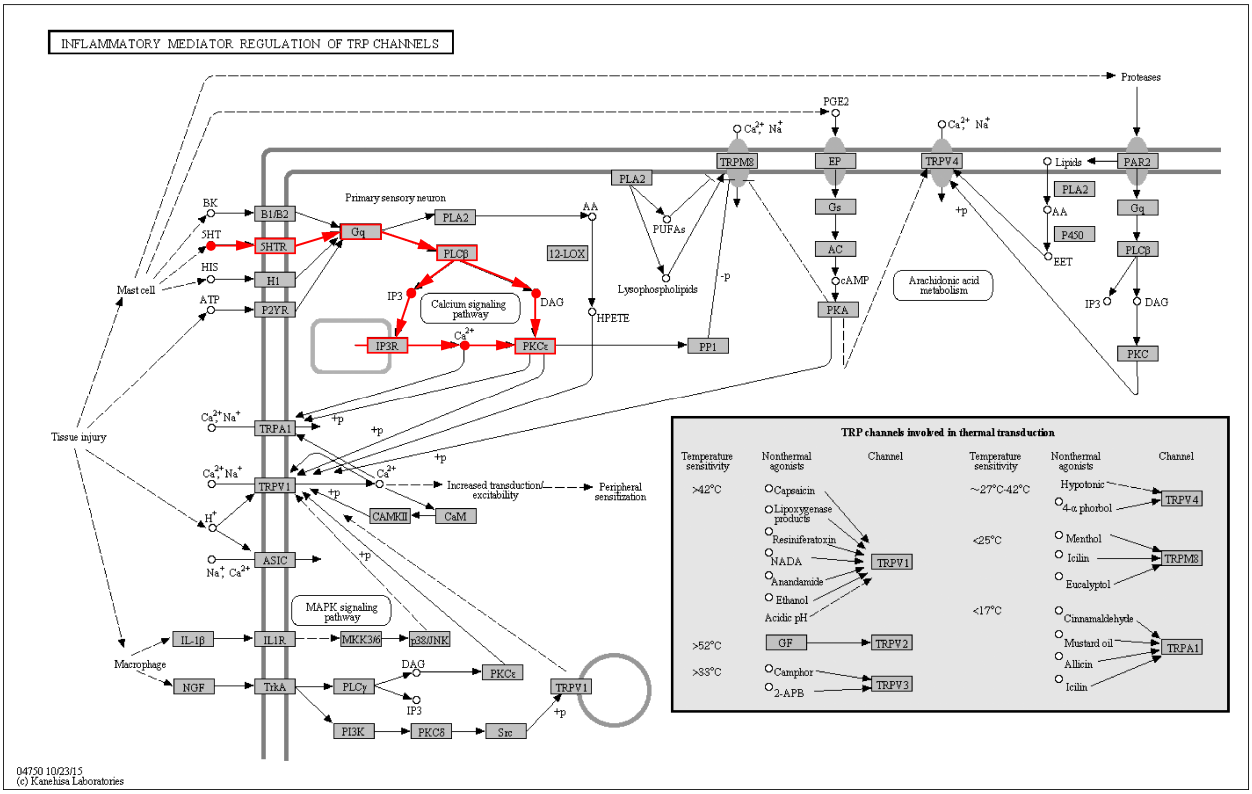


Figure 3.33: KEGG map for inflammatory mediator regulation of TRP channels (map04750). The PKCϵ pathway (depicted in red) is used a running example for the modular Petri net transformation of SBML models.

Running Example. To demonstrate the modular Petri net transformation of SBML models, we refer to a part of the KEGG map for inflammatory mediator regulation of TRP channels (map04750) describing the activation of PKCϵ, see Figure 3.33.

Ligand-bound GPCRs, such as the serotonin receptor HTR2, can interfere with G_q-Proteins and induce the GDP/GTP exchange of their $G_q\alpha$ subunits. The GTP-loaded $G_q\alpha$ subunit activates PLCβ, which breaks down PIP2 into IP3 and DAG. The second messenger IP3 interacts with IP3 receptor channels in the endoplasmatic membrane, which in turn release Ca^{2+} ions. Both, DAG and Ca^{2+} ions bind and activate PKCϵ.

Before transforming the SBML model of the running example into modules, the SBML model code needs to be equipped with the COMPLIST element and each species element with the COMPLISTOFSPECIES element. In Figure 3.35, we show the COMPLIST and COMPLISTOFSPECIES element. Based on the inserted code fragments, the trans-

Algorithm 3.11: Modular Petri net transformation of SBML models.

Require: Modified SBML model code

function findComp(s_{id}, reaction, listOfReactants)
  for all speciesReference in listOfReactants **do**
    s'_{id} = **species** of speciesReference
    $R = \emptyset$
    if $s'_{id} = s_{id}$ **then**
      r_{id} = **id** of reaction
      $R = R \cup r_{id}$
    end if
  end for
  return R
end function

▷ find all species that are involved in a certain reaction

for all component in compList **do**
  c_{id} = **idd** of component
  $S = \emptyset$
  for all species in listOfSpecies **do**
    s_{id} = **id** of species
    for all compOfSpecies in compListOfSpecies **do**
      c'_{id} = **idd** of compOfSpecies
      if $c'_{id} = c_{id}$ **then**
        for all reaction in listOfReactions **do**
          $R_{Product}$ = findComp(s_{id}, reaction, listOfProducts)
          R_{Educt} = findComp(s_{id}, reaction, listOfEducts)
          $R_{Modifier}$ = findComp(s_{id}, reaction, listOfModifiers)
          $R = R_{Product} \cup R_{Educt} \cup R_{Modifier}$
        end for
      end if
    end for
    if $R \neq \emptyset$ **then**
      $S = S \cup s_{id}$
    end if
  end for

▷ find all species and reactions of the current component

  Define $\mathcal{N}(M_{\text{component}}) = \{P, T, F, f, v, m_0\}$ with $T = \emptyset$ and $P = \emptyset$
  for all $r_{id} \in R$ **do**
    $T = T \cup t_{r_{id}}$
    $h(t_{r_{id}})$ = kineticLaw of reaction with **id** = r_{id}
    rev = **reversible** of reaction with **id** = r_{id}
    if rev **then**
      $T = T \cup t_{r_{id},rev}$
      $h(t_{r_{id},rev})$ = reversed kineticLaw of reaction with **id** = r_{id}
    end if
    for all speciesReference in listOfProducts **do**
      s_{id} = **species** of speciesReference
      $P = P \cup p_{s_{id}}$
      $f_{SA}(t_{r_{id}}, p_{s_{id}})$ = **stoichiometry** of speciesReference
      if rev **then**
        $f_{SA}(p_{s_{id}}, t_{r_{id},rev})$ = **stoichiometry** of speciesReference
      end if
    end for
    for all speciesReference in listOfEducts **do**
      s_{id} = **species** of speciesReference
      $P = P \cup p_{s_{id}}$
      $f_{SA}(p_{s_{id}}, t_{r_{id}})$ = **stoichiometry** of speciesReference
      if rev **then**
        $f_{SA}(t_{r_{id},rev}, p_{s_{id}})$ = **stoichiometry** of speciesReference
      end if
    end for
    for all speciesReference in listOfModifiers **do**
      s_{id} = **species** of speciesReference
      $P = P \cup p_{s_{id}}$
      $f_{RA}(p_{s_{id}}, t_{r_{id}})$ = **stoichiometry** of speciesReference
      if rev **then**
        $f_{RA}(p_{s_{id}}, t_{r_{id},rev})$ = **stoichiometry** of speciesReference
      end if
    end for
  end for

▷ define the Petri net module of the current component

  for all $s_{id} \in S$ **do**
    $m_0(p_{s_{id}})$ = **initialAmount** of species with **id** = s_{id}
  end for
end for

Result: $G_{SBML} = \{M_i \mid i = 1, \ldots, n\}$ with $\mathcal{N}(M_i) = \{P, T, F, f, v, m_0\}$

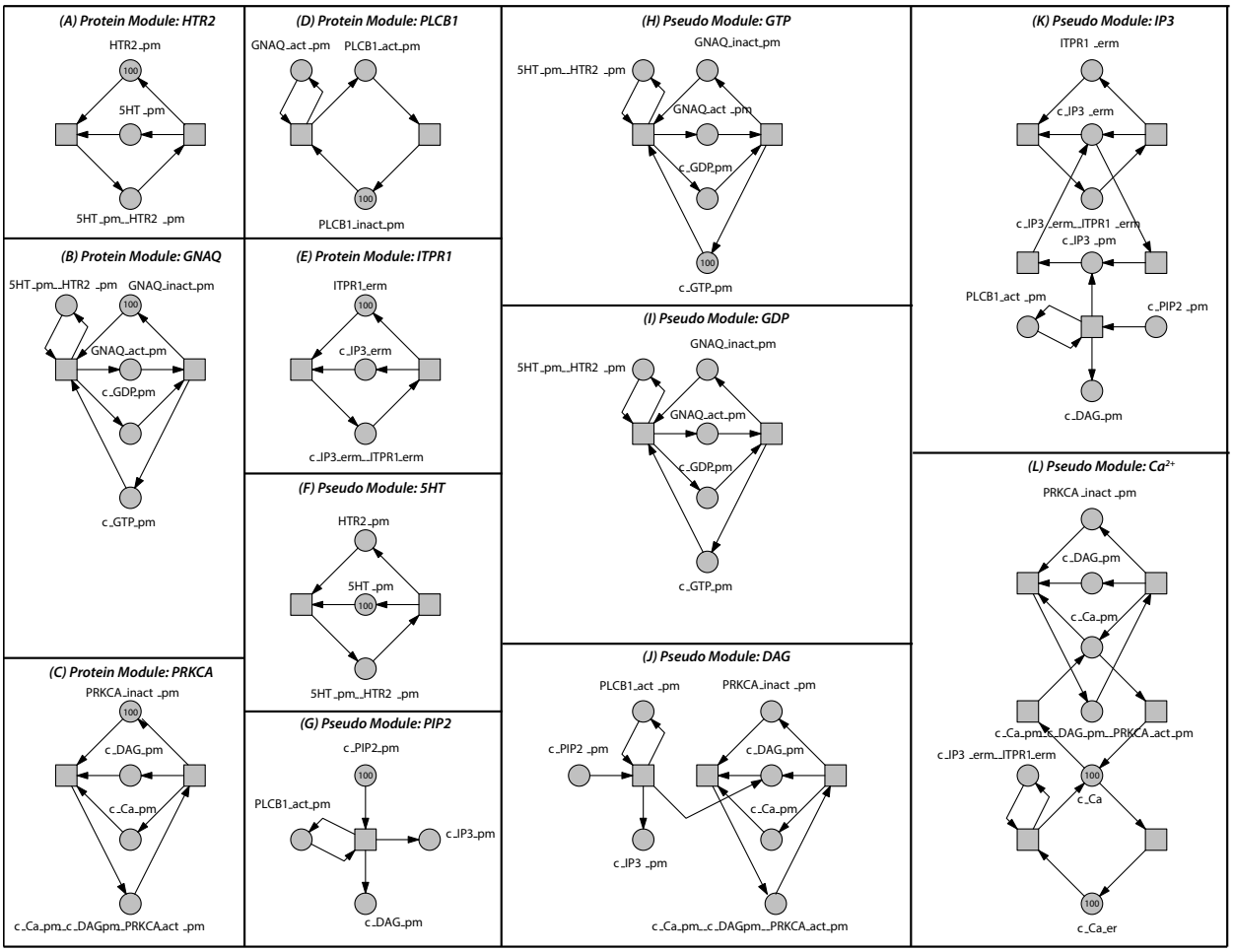


Figure 3.34: Modular Petri net transformation of the SBML model for PKCKϵ. The algorithm generated five protein modules (A)-(E), and seven pseudo modules for non-genetic components (F)-(L).

formation of the SBML model generated five protein modules, see Figure 3.34(A)-(E):

- serotonin receptor HTR2A,
- $G_q\alpha$ subunit (GNAQ),
- PLCβ (PLCB1),
- IP3 receptor channel (ITPR1), and
- PKCϵ (PRKCA)

In addition, the transformation yielded seven pseudo-modules for the non-genetic components, see Figure 3.34(F)-(L):

- GTP,
- GDP,
- PIP2,
- IP3,
- DAG, and
- Ca^{2+} ions

(A) compList Element

```
<bmk:listOfComponents xmlns:bmk="www.biomodelkit.de">
  <bmk:component dbName="Ensembl" name="HTR2A"
    id="ENSG00000102468" type="protein" idd="idd01"/>
  <bmk:component dbName="PubChem" name="5HT"
        id="CID5202" type="non-genetic component" idd="idd02"/>
  <bmk:component dbName="Ensembl" name="GNAQ"
        id="ENSG00000156052" type="protein" idd="idd03"/>
  <bmk:component dbName="Ensembl" name="PLCB1"
        id="ENSG00000182621" type="protein" idd="idd04"/>
  <bmk:component dbName="Ensembl" name="PRKCA"
        id="ENSG00000154229" type="protein" idd="idd05"/>
  <bmk:component dbName="Ensembl" name="ITPR1"
        id="ENSG00000150995" type="protein" idd="idd06"/>
  <bmk:component dbName="PubChem" name="PIP2"
        id="CID5497157" type="non-genetic component" idd="idd07"/>
  <bmk:component dbName="PubChem" name="IP3"
        id="CID55310" type="non-genetic component" idd="idd08"/>
  <bmk:component dbName="PubChem" name="DAG"
        id="SID3465" type="non-genetic component" idd="idd09"/>
  <bmk:component dbName="PubChem" name="Ca"
        id="CID5460341" type="non-genetic component" idd="idd10"/>
  <bmk:component dbName="PubChem" name="GDP"
        id="CID8977" type="non-genetic component"idd="idd11"/>
  <bmk:component dbName="PubChem" name="GTP"
        id="CID6830" type="non-genetic component" idd="idd12"/>
</bmk:listOfComponents>
```

(B) compListOfSpecies Elements

```
<listOfSpecies>
    ...
    <species id="ID_10" name="DAG" ...>
        <annotation>
            <bmk:compListofSpecies xmlns:bmk="www.biomodelkit.de">
                <bmk:compOfSpecies idd="idd09"/>
            </bmk:compListofSpecies>
        </annotation>
    </species>
    <species id="ID_11" name="PRKCA_inact" ...>
        <annotation>
            <bmk:compListofSpecies xmlns:bmk="www.biomodelkit.de">
                <bmk:compOfSpecies idd="idd05"/>
            </bmk:compListofSpecies>
        </annotation>
    </species>
    <species id="ID_12" name="PRKCA_CA_DAG" ...>
        <annotation>
            <bmk:compListofSpecies xmlns:bmk="www.biomodelkit.de">
                <bmk:compOfSpecies idd="idd05"/>
                <bmk:compOfSpecies idd="idd09"/>
                <bmk:compOfSpecies idd="idd10"/>
            </bmk:compListofSpecies>
        </annotation>
    </species>
    ...
</listOfSpecies>
```

Figure 3.35: Modifications of the PKC SBML model. (A) attached compList element and (B) attached compListofSpecies elements

4
BioModelKit (BMK) Web-Tool

The BMKfr offers more than a repository of model files in the form of modules. Modules are explicitly stored in a relational MySQL database [20]. Both, the Snoopy file [120] of the Petri net graph including places, transitions, arcs, firing rates, and markings and the BMKml file holding the module annotation with all its content, see Section 3.2, is stored in the BMKdb as displayed in Figure 4.1. Storing modules in the BMKdb allows to keep track of their versioning, changes and updates. Through the BMKwi the user can interact with the BMKdb, which allows the user to browse and search for modules, as well as to display the graph of a module and the annotations. Also, detailed information about places and transitions can be shown separately as well. The user can automatically compose a model from a set of *ad hoc* chosen modules, which are stored in user-defined collections. Also, the user can apply the variations of the algorithmic model mutation to the model composition, see Section 3.3.3 or the spatial model transformation, see Section 3.3.4. The submission and curation of modules is supported by the BMKwi as well.

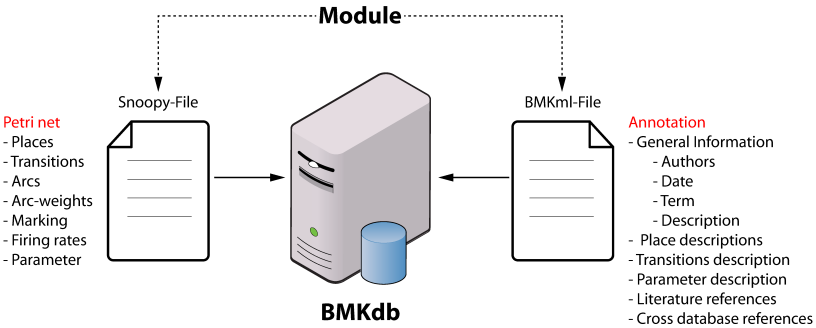


Figure 4.1: BMK database. The Snoopy file [120] of the Petri net graph of a module including places, transitions, arcs, parameters, firing rates, and markings and the BMKml file holding the module annotation with all its content, see Section 3.2, is stored in the BMKdb.

In section 4.1 we introduce the general web-site structure of the BMKwi and all of its features mentioned above in more detail. The structure of the BMKdb to store the Petri net graph and annotations of a module, as well as the module versioning is discussed in Section 4.2. Afterwards, in Section 4.3, we explain how the Petri net graph and annotations of a module are mapped to the structure of the BMKdb.

4.1 BMK - Web-Interface

This section requires:

- *Module definition, Section 3.1.1*
- *Module types, Section 3.1.2*
- *Petri net representation of modules, Section 3.1.3*

In order to publicly access the modules and annotations stored in the BMKDB (see Section 4.2), a web-interface, BMKWI, has been implement, which is accessible online at `www.biomodelkit.com`. The BMKWI is running on a Linux/Apache 2.2.14 web server with MySQL client version 5.1.67, PHP 5.2.0, and Javascript 1.8.5. The URL-based and form-based navigation allows navigating hierarchically from a list of modules to a particular module and nodes of the underlying Petri net graph, as well as parameters. On each level, annotations of the particular element are displayed, see below for more details. The user can also query for modules matching his criteria through a form. Each website in the BMKWI is equipped with the menu bar and a log-in/registration menu, see Figure 4.2. The model and annotation file of a module can be downloaded separately. Furthermore the web-interface of the BMKDB facilitates the automatic composition of models based on a set of *ad hoc* chosen modules, as well as the algorithmic model mutation and spatial extension of composed models. The web-interface also provides features to submit and curate modules.

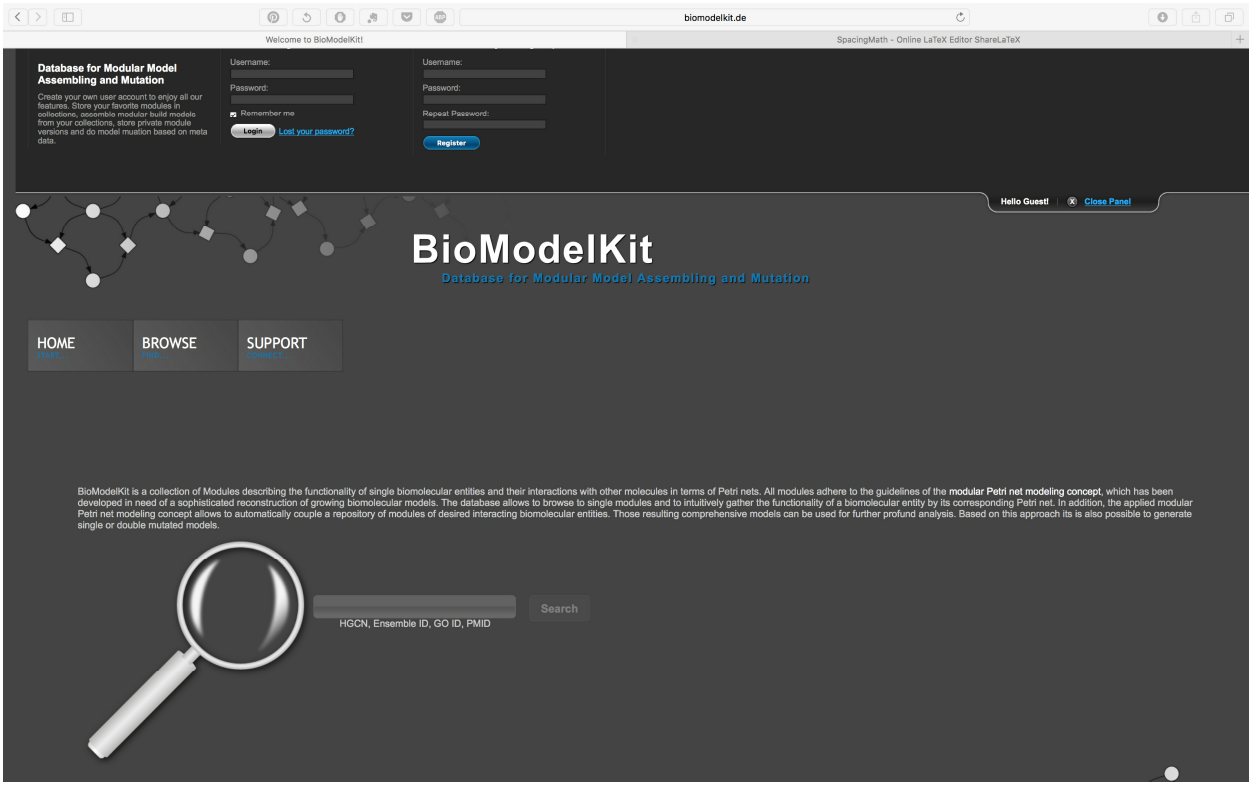


Figure 4.2: Home view. Each website in the BMKWI is equipped with a menu bar and a log-in-/registration menu like the home view.

The menu bar allows the URL-based navigation to access the home view via the "home" item, the module list view via the "Browse" item and support sites (help, terms of use, about, and contact) via the "Support" item. The "Profile" item in the menu bar is only displayed

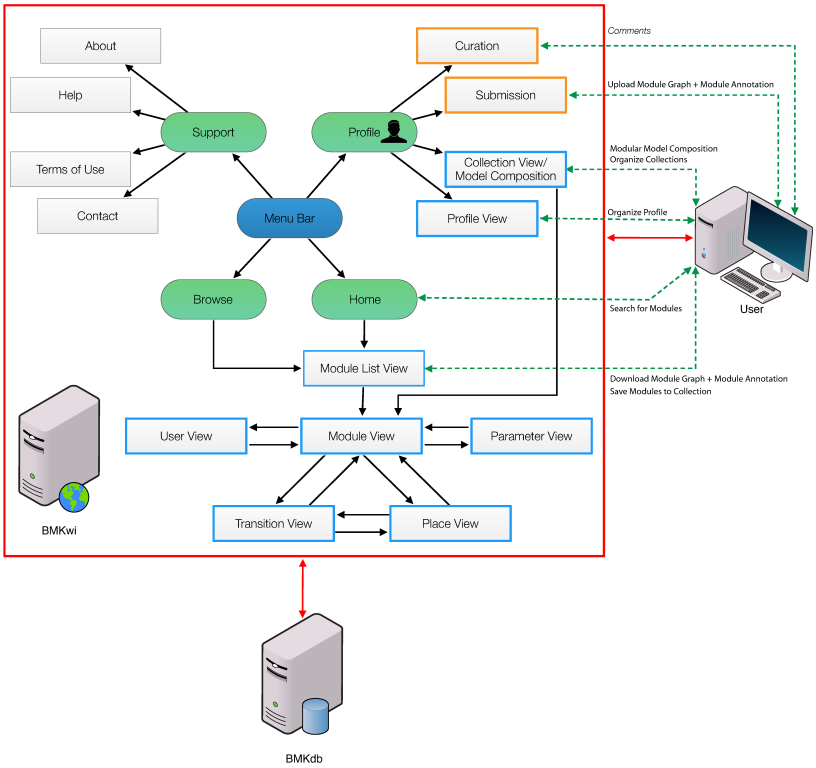


Figure 4.3: Navigation scheme of the BMK web-interface. The scheme illustrates how the different features (grey boxes), e.g. views and other web-pages of the BMKwi can be accessed from the items (green box) of the menu bar (blue box) and how they are linked to each other (black arcs). The red arcs indicate the interaction between the BMKwi hosted on a web server, the BMKdb hosted by a database server with a MySQl Client, and the user. All views marked in blue are generated by querying the BMKdb. Using the curation and submission features marked in orange, new modules and annotations can be stored in the BMKdb after manual validation. Arcs in red indicate additional user activities other than navigating through the website, like the up- and download of modules and their annotation, performance of the modular model composition, the organization of the user profile and collection, etc. Downloaded modules and modularly composed models can be directly executed in Snoopy [120] and Marcie [129].

for registered users, that are logged in. Thus, the BMKwi is divided into a public and a non-public area. In the public area of BMKfr, the user can browse through the modules stored in BMKdb and access the different views specified below and query for modules, see also Figure 4.2. In the non-public area of the BMKwi, the user can excess additional features. Through the log-in/registration panel at the top of the site, the user can register as an active member of the BMKfr. Now, using his registration data, the user can log-in to access the non-public area of the BMKwi. After a successful log-in, the "Profile" item in the menu bar is displayed. Via the "Profile" item the user can access the profile view, the collection view with the model composition features, the module submission feature, and curation feature. Most importantly, only registered users can manage modules in user-defined collections and thus, take advantage of the model composition features, including the standard model composition (see Section 3.3.1), three different approaches of the algorithmic model mutation (see Section 3.3.3) and the spatial model transformation (see Section 3.3.4). Also, only registered users are allowed to submit and curate modules. The navigation through the BMKwi is summarized in Figure 4.3. The functionality of the different views and features accessible in the BMKwi are explained in detail below.

Module List View

The module list view displays the modules stored in the BMKdb, see Figure 4.4. In the default setting, all modules are displayed. The

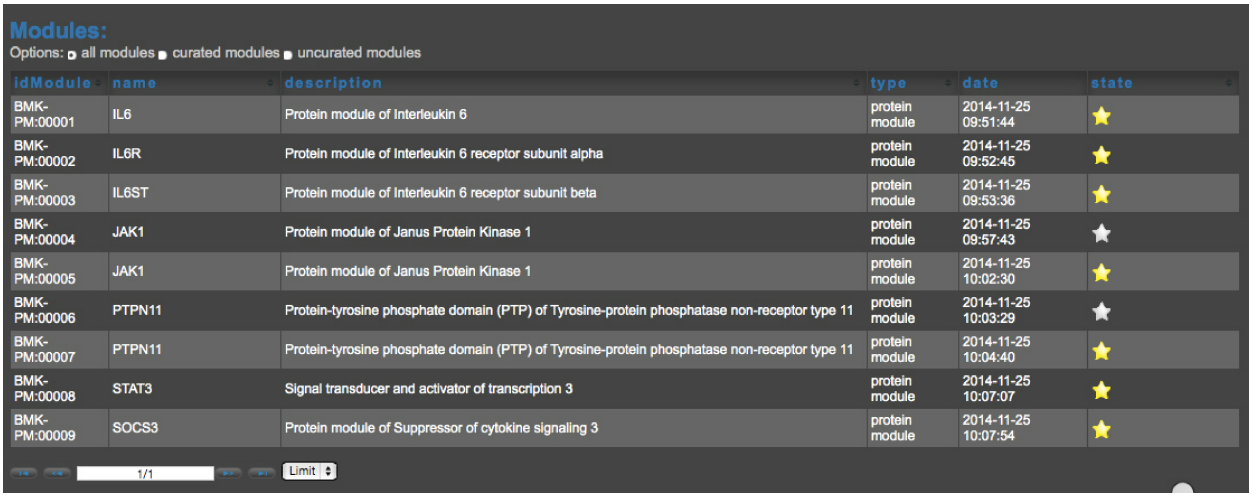


Figure 4.4: Module list view.

view can be restricted to list only curated or uncurated modules. Through form-based navigation, the module list view is restricted to the search criteria. As mentioned above from the home view, the user can access the search box and enter a search term. In the module list view, all modules are displayed matching the search term. For each module in the module list view the internal BMKID is displayed next to the module name, a short description of the module, the module type, release date, and curation state. The internal BMKID is a stable identifier for each module stored in the BMKDB of the form *BMK-<module type symbol><id>*, where *<module type symbol>* can be:

- GM - gene module
- MM - mRNA module
- PM - protein module
- DM - protein degradation module
- AM - allelic influence module
- CM - causal influence module

and *<id>* is a positive integer number assigned through the MySQL database management system [20] of BMKDB.

By entering a search term like the HGNC symbol [140] or name of a component in the search box at the home view, all modules matching the search term are displayed. For example, by entering the search term "IL-6", protein modules for the cytokine interleukin-6, interleukin-6 receptor subunit alpha and interleukin-6 receptor subunit beta are returned, see also the example in Section 5.1.

Module View

In the module view the user can view the module graph and the module annotation. The name of the module is displayed, e.g. Il6, next to its curation state (curated, yellow star) and the approved full gene name by the HGNC [140] (interleukin 6), as well as the corresponding gene Ensembl identifier [78] (ENSG00000136244), see

MODULE: IL6

interleukin 6 (ENSG00000136244)

General Information

Submitter(s): Blätke, Mary Ann
Curator(s): Schaper, Fred
Dittrich, Anna
Module ID: BMK-PM:00001
Release date: 2014-11-10 09:33:29
Description: Protein module of Interleukin 6
Module Type: Protein Module

Place Overview

Place Name	Description
IL6R_CBM	cytokine binding module (CBM) of Interleukin-6 receptor subunit alpha (IL6R)
IL6_siteI:IL6R_CBM	cytokine binding module (CBM) of Interleukin-6 receptor subunit alpha (IL6R) bound to binding site I (siteI) of Interleukin-6 (IL6)
IL6R_siteIIIb	site IIIb (siteIIIb) of Interleukin-6 receptor subunit beta (IL6R)
IL6R_siteIIb	site IIb (siteIIb) of Interleukin-6 receptor subunit beta (IL6R)
IL6_siteI	binding site I (siteI) of Interleukin-6 (IL6)
IL6_siteIIIa	binding site IIIa (siteIIIa) of Interleukin-6 (IL6)
IL6_siteIIa	binding site IIa (siteIIa) of Interleukin-6 (IL6)
IL6ST_Box1:IL6ST_Box2:JAK1_FERM	4.1 ezerin radixin and moesin domain of Tyrosine-protein kinase JAK1 (JAK1) bound to Box1 and Box2 of Interleukin-6 receptor subunit beta (IL6ST)
IL6ST_CBM	cytokine binding module (CBM) of Interleukin-6 receptor subunit beta (IL6ST)
IL6_siteIIa:IL6R_siteIIb:IL6ST_CBM	cytokine binding module (CBM) of Interleukin-6 receptor subunit beta (IL6ST) bound to binding site IIa (siteIIa) of Interleukin-6 (IL6) and binding site IIb (siteIIb) of Interleukin-6 receptor subunit alpha (IL6R)
IL6ST_IgLike	immunoglobulin-like domain (Iglike) of Interleukin-6 receptor subunit beta (IL6ST)
IL6_siteIIIa:IL6R_siteIIIb:IL6ST_IgLike	Immunoglobulin-like domain (Iglike) of Interleukin-6 receptor subunit beta (IL6ST) bound to binding site IIIa of Interleukin-6 (IL6) and binding site IIIb (siteIIIb) of Interleukin-6 receptor subunit alpha (IL6R)

Transition Overview

Transition Name	Description
IL6_IL6R_IL6ST_t1	complex formation of CBM of IL6ST, binding site IIa of IL6 and binding site IIb of IL6R if the CBM of IL6R is already bound to binding site I of IL6 and the FERM motif of JAK1 is bound to the box1 and box2 motif domain of IL6ST
IL6_IL6R_IL6ST_t2	dissociation of the complex between the CBM of IL6ST, binding site IIa of IL6 and binding site IIb of IL6R
IL6_IL6R_IL6ST_t3	complex formation of Iglike domain of IL6ST, binding site IIIa of IL6 and binding site IIIb of IL6R if the CBM of IL6R is already bound to binding site I of IL6 , the FERM motif of JAK1 is bound to the box1 and box2 motif domain of IL6ST and the CBM of IL6ST is bound to binding site IIa of IL6 and binding site IIIb of IL6R
IL6_IL6R_IL6ST_t4	dissociation of the complex between the Ig-like domain of IL6ST, binding site IIIa of IL6 and binding site IIIb of IL6R
IL6_IL6R_t1	binding of the binding site I of IL6 to the CBM of IL6R
IL6_IL6R_t2	dissociation of the binding site I of IL6 from the CBM of IL6R

Parameter Overview

Parameter	Description
k	constant

View/Download

IL6:
- SPN-Module
- XML-Metadata

(click left to view file, click right to download file)

Module Versions

No other versions in this database.

Connectable Modules

- BMK-PM:00002 (IL6R)
- BMK-PM:00003 (IL6ST)
- BMK-PM:00006 (PTPN11)
- BMK-PM:00007 (PTPN11)
- BMK-PM:00004 (JAK1)
- BMK-PM:00005 (JAK1)

Publications

- Kurth I, Horsten U, Pflanz S, Dahmen H, Küster A, Grötzinger J, Heinrich PC, Müller-Newen G. Activation of the signal transducer glycoprotein 130 by both IL-6 and IL-11 requires two distinct binding epitopes. (1999) Journal of immunology (Baltimore, Md. : 1950) Vol. 162, 1480-7 [PMID:9973404]
- Simpson RJ, Hammacher A, Smith DK, Matthews JM, Ward LD. Interleukin-6: structure-function relationships. (1997) Protein science : a publication of the Protein Society Vol. 6, 929-55 [PMID:9144766]
- Paonessa G, Graziani R, De Serio A, Savino R, Ciapponi L, Lahm A, Salvati AL, Toniatti C, Ciliberto G. Two distinct and independent sites on IL-6 trigger gp 130 dimer formation and signalling. (1995) The EMBO journal Vol. 14, 1942-51 [PMID:7744001]
- Hammacher A, Richardson RT, Layton JE, Smith DK, Angus LJ, Hilton DJ, Nicola NA, Wijdenes J, Simpson RJ. The immunoglobulin-like module of gp130 is required for signaling by interleukin-6, but not by leukemia inhibitory factor. (1998) The Journal of biological chemistry Vol. 273, 22701-7 [PMID:9712900]

Figure 4.5: Module view of IL6.

Figure 4.5. The gene Ensembl identifier [78] allows to directly navigate to the corresponding entry on the Ensembl website [78]. Further on, the module view gives some general information containing a list of submitters (modellers) and curators, the internal BMKid, the release date, a short description of the modelled genetic component, and its module type. The module view also provides an overview of all places, transitions, and parameters with their respective name next to a short description. By clicking on the name of a transition, place or parameter, one can access detailed views of the corresponding objects, see also below. From the module view the module file in the Snoopy-file format [120] containing the Petri net graph, as well as the annotation file, can be viewed or downloaded. The module view also lists alternative module version of the respective module and modules with matching interface networks. Again, by clicking on the names, the user can access the module view of the chosen module. A list of publications used to construct the module is also given in the module view. The provided PubMed identifier allows navigating directly to the corresponding publication on PubMed [32].

Place View

In the place view the name of the place, and the parent module is displayed, e.g. place $IL6\_siteI : IL6R\_CBM$ in the protein module for $IL6$, see Figure 4.6. Through the name of the parent module, the user can go back to the corresponding module view. Further on, the place view provides a short description, specifies the marking of the place, as well as its post- and pre-transitions. The place view states related places and modules. Related places are places, which represent other molecular states of the same functional unit represented by the original place. Related modules are those modules containing the original place as well. Thus, the user can quickly identify pre-and post transitions, related places and modules of interest; and directly view them by clicking on the displayed names. It also provides a list of publications and cross-references to other relevant biomolecular databases, see Table 3.8. The displayed reference identifiers allow accessing the reference sources directly on the corresponding websites.

Transition View

In the transition view, the name of the transition and the parent module is displayed, e.g. transition $IL6\_IL6R\_Il6ST\_t1$ in the protein module for $IL6$, see Figure 4.7. Through the name of the parent module, the user can go back to the corresponding module view. Further on, the transition view provides a short description, specifies the firing-rate of the transition, as well as its post- and pre-places. The transition view states related transitions and modules. Related transitions are transitions, which represent other molecular events of the same molecular process. Related modules are those modules containing the original transition as well. Thus, the user can quickly

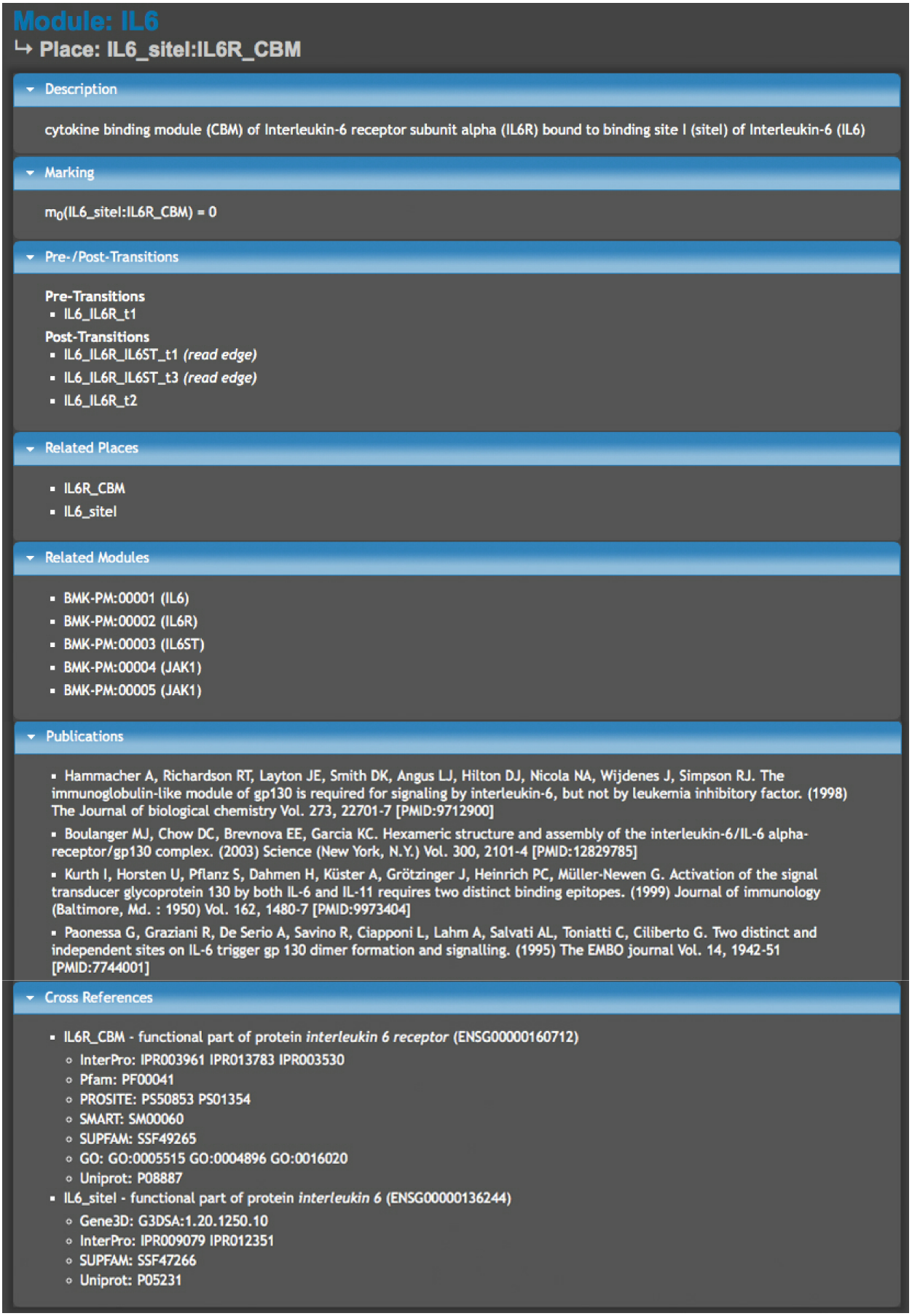


Figure 4.6: Place view of place IL6_siteI__IL6R_CBM of the Il6 module.

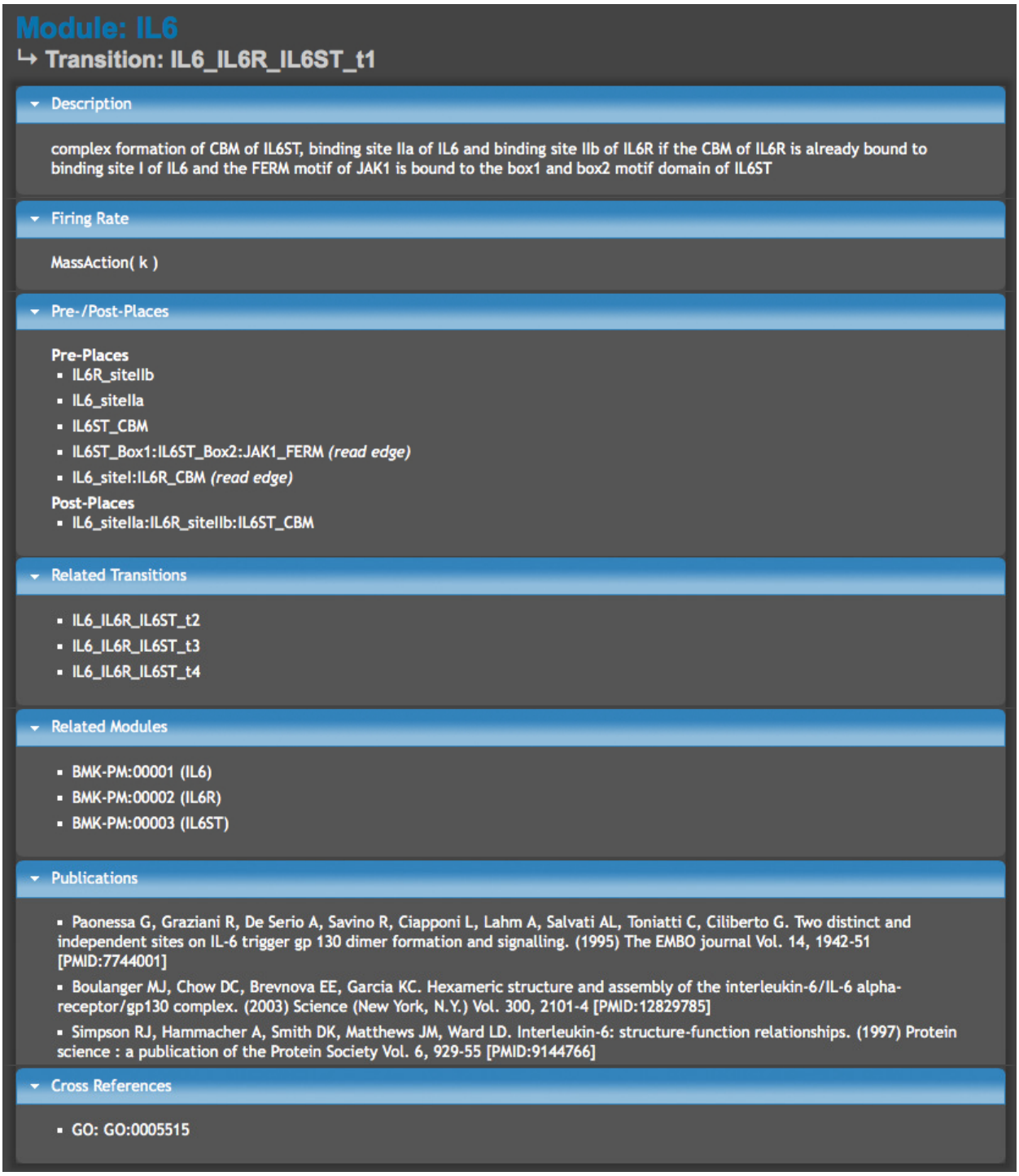


Figure 4.7: Transition view of transition IL6_IL6R_IL6ST_t1 in the Il6 module.

identify related transitions and modules of interest; and directly view them by clicking on the displayed names. It also provides a list of publications and cross-references to other relevant biomolecular databases, see Table 3.8. Again, the displayed reference identifiers allow accessing the reference sources directly on the corresponding websites.

Parameter View
In the parameter view, the name of the parameter and the parent module is displayed. Further on, the parameter view provides a short description and specifies the numeric value. The parameter view states all modules, where the respective parameter appears. Thus, the user can quickly identify modules of interest; and directly view them by clicking on the displayed names. The parameter view also provides a list of publications and cross-references to other relevant biomolecular databases, see Table 3.8. As before, the displayed reference identifiers allow to access the reference sources directly on the corresponding websites.

The following views can only be accessed by registered user, that are logged in:

User Profile View
In the user profile the user might specify his name and contact information, but it is not mandatory to provide such information. The information is not visible to other users.

Module Submission View
In the module submission view, the user can submit his modules, including the Petri net graph and annotation, through the BMKfr and thus, actively provide new content shared in the BMKfr community. The first step in the module submission process is to upload the Petri net graph of the module as a Snoopy-file format for uncoloured Petri nets [120]. The first step in the module submission process is to upload the Petri net graph of the module as a Snoopy-file for uncoloured Petri nets [120]. During the upload, a parser generates a form based on the respective Petri net graph to enter the module annotation. Thus, the user must not already provide the module annotation as a valid BMKml file, see Section 3.2. The prompted form helps the user entering the relevant annotations. The XML-coding of the BMKml file is done by a script. In addition, the user can also upload an existing module annotation as a valid BMKml file. Both, the model file and the annotation file are stored in a folder with a unique auto-generated name on the BMK web-server. Successfully approved modules by an administrator will be released and displayed in the module view.

Module Curation View
The user can submit their comments by stating the respective module,

node, or parameter via a form. The information helps to improve modules. This feature aims at increasing the quality of the content provided by the BMKfr.

Collection View

The collection view provides features to organise modules in collections and to compose models from a set of modules stored in a collection. A collection specifies a selection of modules, which can either be pre-defined by the BMKfr or defined by the user. There are two types of pre-defined collections, which cannot be changed by the user, one holding submitted modules and the other holding curated modules. The user can create user-defined collection by specifying a collection name. User-defined collections can be renamed and deleted. The user can add any number of modules to his user-defined collections according to his personal criteria, e.g. this can be modules of any interest, modules involved in a particular pathway, or modules representing related isoforms of a protein, etc. To add modules to collections, the module list view and the module view is extended with a feature to store modules in the user-defined collections for logged in users. Modules can also be deleted from a user-defined collection if necessary. Form each displayed collection the user can also access the modular model composition features, including:

- Composition of the unmodified model: The feature creates the unmodified modularly composed model, see also Section 3.3.1.
- Composition of alternative (mutated) models: The feature creates modified versions of the unmodified modularly composed model, see Section 3.3.3. The user has the options to choose between the module and transition knock-out. The transition knock-out can be constrained by the module annotation. Such that, the user has to specify cross-references that should be used to knock-out transitions.
- Composition of the spatial model[1]: The modularly composed model is extended with spatial information, to allow the movement of involved components in a defined space and their locally constrained interactions, see Section 3.3.4.

[1] Note: feature will be available in the next release.

The output of each applied feature results in a zip-file that contains the composed model(s) in the andl-format, respectively candl-format (in case of a spatial model) [2], and a text file containing information about the composition, modifications and resolved conflicts. Conflicts during model composition might occur due to mismatched names or the integration of different module version in one model.

[2] The andl-format is an abstract net description language for standard Petri net classes, where the candl-format is used for coloured Petri nets, which are compatible with Snoopy [120] and Marcie [129].

4.2 BMK - Database

This section requires:

- *Petri nets [82], Section 2.1*
- *MySQL [20], see text*
- *Module definition, Section 3.1.1*
- *Module types, Section 3.1.2*
- *Petri net representation of modules, Section 3.1.3*

In general, a database is an organised collection of data to model certain aspects of the reality and to support requiring information. In other words, a database is a collection of schemas, tables, queries, reports, views and other objects.

A database management system (DBMS) is a computer software application that allows the interaction of the user or other applications with the database itself to capture and analyse data. General tasks of a DBMS are to define, create, query, update and administrate a database. A Software system used to maintain relational databases is called relational database management system (RDBMS), which is the most popular database systems since the 1980s.

A relational database is a digital database that is organised according to the relational model proposed by E. F. Codd in 1970 [6]. The relational model uses a structure and language that is consistent with first-order predicate logic. All data is represented in terms of tuples, grouped into relations. It provides a declarative method for specifying data and queries. Users can declare the information content of the database and the information to retrieve from it. The RDBMS takes care of all the actions and provides data structures for storing and retrieving information.

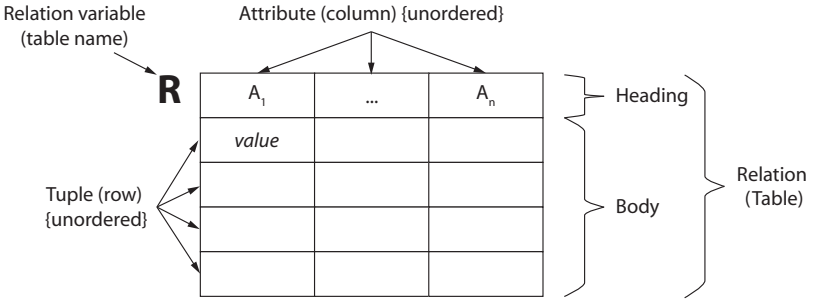


Figure 4.8: Concepts of the Relational Model. A table is relation and is represented by a relation variable, the table name. A table consists of a heading and a body. The heading consists of an unordered set of attributes, the columns of a table. An attribute is defined by the attribute name and type name, where type denotes a valid domain or data-type. The body is a set of n-tuples, the rows of a table, where n is the number of attributes. A tuple is an ordered set of attribute values, where the attribute value must be specific valid value for the type of the attribute.

In a relational database, a *table* is a relation and is represented by a relation variable, the table name. A table consists of a heading and a body. The *heading* consists of an unordered set of attributes, the columns of a table. An *attribute* is defined by the attribute name and type name, where type denotes a valid domain or data-type. The *body* is a set of n-tuples, the rows of a table, where n is the number of attributes. A *tuple* is an ordered set of attribute values, where the attribute value must be a specific valid value for the type of the attribute. Figure 4.8 summarises the terms introduced above. Additional constraints can be added to the attributes of a table, such as primary and foreign keys. Usually, one attribute of a table is chosen as a *primary key*. Meaning, the attribute values of an attribute selected as the primary key must be unique, none of the values is allowed to appear twice. Such that, a primary key is allowed to migrate to other entities (tables) to define the relationships that exist among the entities. In the migrated table, the primary key is now called

foreign key. The schema of the database represents the tables in a database and their relations to each other; it can be visualised using e.g. an entity relationship model (ER model). In the *ER model* boxes represent tables and their attributes, lines among the table indicate how attributes are linked to each other by the use of primary and foreign keys.

Most RDBMS employ SQL (Structured Query Language), which is a special-purpose programming language to manage data, which is originally based upon relational algebra and tuple relational calculus. SQL consists of a data definition language, a data manipulation language, and a data control language. In an SQL database schema, a table represents a predicate variable; where the content of a table corresponds to a relation; key constraints, other constraints, and SQL queries can be interpreted as predicates. MySQL is one of the most popular open-source RDBMS. Detailed specification on MySQL can be found in [20].

The BMKDB is implemented using the RDBMS MySQL [20]. The scheme of the BMKDB is shown by the ER model in Figure 4.9. The "root" of the BMKDB is table 'module', which provides a unique identifier ('idModule') for each module. The 'idModule' is the accession to all nodes, arcs, parameters, annotations, and parts of a module. In the BMKDB, the term "parts" comprises all molecular states of functional units except interaction states. The tables depicted in the BMKDB scheme in Figure 4.9 can be sub-divided into four types:

1. MAIN TABLES store information about the objects of the model and annotation of a module, including module instances, nodes, arcs, parts, parameters, references, etc. They provide a unique identifier (primary key) for the respective objects (depicted in red, see Figure 4.9). The user is considered as an object as well.

2. TYPE TABLES define all possible types that are allowed for the objects represented by the main tables (depicted in green, see Figure 4.9').

3. ATTRIBUTE TABLES provide additional information of the objects given by the main tables by using their primary key as a foreign key (depicted in orange, see Figure 4.9).

4. LINKAGE TABLES define the relation between two main tables or the main table, e.g. table 'module', and attribute tables (depicted in grey, see Figure 4.9).

The complex structure of the BMKDB scheme allows the module versioning. In the following, we provide a detailed description of the tables given in Figure 4.9 in alphabetical order:

COLLECTION (main table) - stores information about user-defined collections of modules by providing a unique collection identifier ('idCollection', primary key), the identifier of the user ('idUser'), and

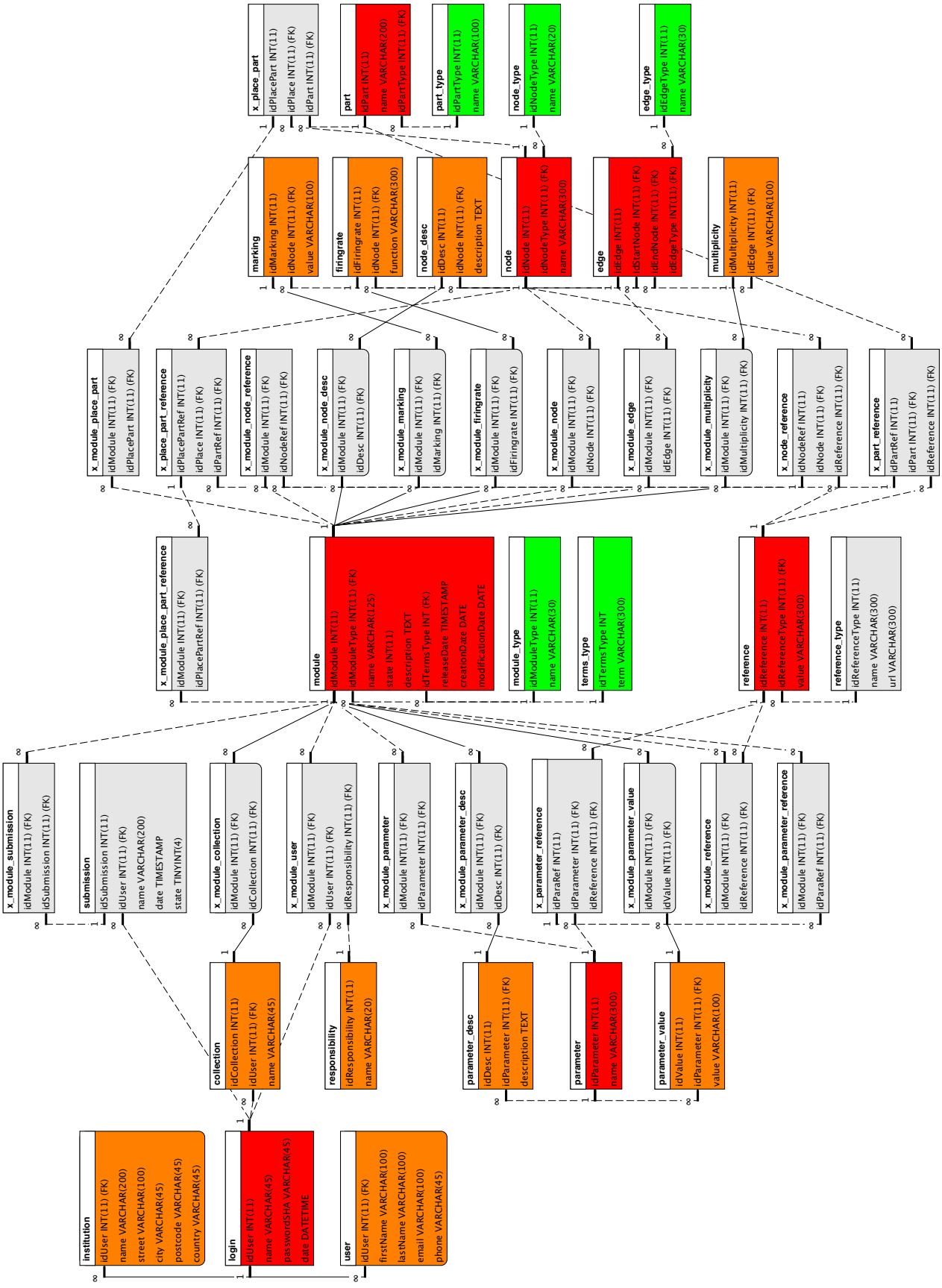


Figure 4.9: Entity relationship model of the BMK database. The ER model depicts all tables defined in the BMKDB and their relations to store the graph structure of the module, kinetic data (parameters, rates), and the provided module annotation. The table 'module' is the root table for each module. The module identifier ('idModule') provides a unique identifier for each module. Next to table 'module', there are some more main tables for nodes, arcs, parameters, references, parts, and user (red). All other tables link those tables (grey) or hold additional information (orange, green).

the name of the collection (‘name‘). The entity ‘idUser‘ is a foreign key to access table ‘login‘.

EDGE (main table) - stores information about the arcs by providing a unique identifier for each arc (‘idEdge‘, primary key), the identifier of the start node (‘idStartNode‘), and end node (‘idEndNode‘), as well as the identifier of the arc type (‘idEdgeType‘). The entries ‘idStartNode‘, ‘idEndnote‘, and ‘idEdgeType‘ are used as foreign keys to access table ‘node‘, and respectively table ‘edge_type‘.

EDGE_TYPE (type table of table ‘edge‘) - stores information about the arc types by providing a unique identifier for each arc type (‘idEdgeType‘, primary key) and a descriptive name (‘name‘). Arc types that are available are standard arc, read arc, inhibitory arc, reset arc, equal arc, and modifier arc.

FIRINGRATE (attribute table of table ‘node‘) - stores information about the firing-rates of the transitions by providing a unique identifier for each firing-rate (‘idFiringrate‘, primary key), the identifier of the referring node (‘idNode‘), and the firing-rate equation (‘function‘). The entry ‘idNode‘ is a foreign key to access the table ‘node‘. Furthermore in table ‘node‘, the corresponding entry of the node type (‘idNodeType‘) has to refer to a transition.

INSTITUTION (attribute table of table ‘login‘) - stores information about the institution of the users by providing the identifier of the user (‘idUser‘, primary key), the name of the institution (‘name‘), and the address (‘street‘, ‘city‘, ‘postcode‘, ‘country‘). The entry ‘idUser‘ is a foreign key to access table ‘login‘.

LOGIN (main table) - stores information about the user’s login details by providing a unique user identifier (‘idUser‘, primary key), a unique username (‘name‘), and password (‘passwordSHA‘), as well as the registration date (‘date‘).

MARKING (attribute table of table ‘node‘) - stores information about the marking of the places by providing a unique marking identifier (‘idMarking‘, primary key), the identifier of the referring node (‘idNode‘), and the marking value (‘value‘), which can either be an integer, a real number, or a name of a parameter. The entry ‘idNode‘ is a foreign key to access the table ‘node‘. Furthermore in table ‘node‘, the corresponding entry of the node type (‘idNodeType‘) has to refer to a place.

MODULE (main table) - stores information about modules by providing a unique module identifier (‘idModule‘, primary key), the identifier of the module type (‘idModuleType‘), a name (‘name‘)‘, the curation state (‘state‘), a short description (‘description‘), the release date (‘releaseDate‘), creation date (‘creationDate‘), and last date of modi-

fication (modificationDate‘), as well as the identifier of the module type of the terms type (‘idTermsType‘). The value for entry ‘state‘ is limited to 0 (not curated) and 1 (curated). The entries ‘idModuleType‘ and ‘idTermsType‘ are foreign keys to access table ‘module_type‘, respectively table ‘terms_type‘.

MODULE_TYPE (type table of table ‘module‘) - stores information about the module types by providing a unique identifier for each module type (‘idModuleType‘, primary key) and a descriptive name (‘name‘). Module types that are available are gene module, mRNA module, protein module, protein degradation module, allelic influence module, and causal influence module.

MULTIPLICITY (attribute table of table ‘node‘) - stores information about the multiplicities of the arcs by providing a unique identifier for each multiplicity (‘idMultiplicity‘, primary key), the identifier of the arc (‘idEdge‘), and the value of the multiplicity (‘value‘), which can either be an integer, a real number, or a name of a parameter. The entry ‘idEdge‘ is a foreign key to access table ‘edge‘.

NODE (main table) - stores information about the nodes by providing a unique node identifier (‘idNode‘, primary key), the identifier of the node type (‘idNodeType‘), and the name of the node (‘name‘). The entry ‘idNodeType‘ is a foreign key to access table ‘node_type‘.

NODE_DESC (attribute table of table ‘node‘) - stores more detailed description of each node by providing a unique identifier for each description (‘idDesc‘, primary key), the identifier of the node (‘idNode‘), and description (‘description‘). The entry ‘idNode‘ is a foreign key to access the table ‘node‘.

NODE_TYPE (type table of table ‘node‘) - stores information about the node type by providing a unique node type identifier (‘idNodeType‘, primary key) and a descriptive name (‘name‘). Node types that are available are place and transition.

PARAMETER (main table) - stores information about the parameters used in the module by providing a unique parameter identifier (‘idParameter‘, primary key) and a name (‘name‘).

PARAMETER_DESC (attribute table of table ‘parameter‘) - stores more detailed description for each parameter by providing a unique description identifier (‘idDesc‘, primary key), the parameter identifier (‘idParameter‘), and a description (‘description‘). The entry ‘idParameter‘ is a foreign key to access table ‘parameter‘.

PARAMETER_VALUE (attribute table of table ‘parameter‘) - stores information about the parameter value by providing a unique parameter value identifier (‘idValue‘, primary key), the parameter identifier (‘id-

Parameter'), and the value ('value'), which can either be an integer, a real number, or a name of a parameter. The entry 'idParameter' is a foreign key to access table parameter.

PART (main table) - stores information about parts by providing a unique part identifier ('idPart', primary key), the name of the part ('name'), and the identifier of the part type ('idPartType'). The entry 'idPartType' is a foreign key to access table 'part_type'.

PART_TYPE (type table of table 'part') - stores information about the part types by providing a unique part type identifier ('idPartType', primary key) and a descriptive name ('name'). Part types that are available are gene, mRNA, protein, and chemical compound (non-genetic components, i.e. small molecules).

REFERENCE (main table) - stores information about references by providing a unique reference identifier ('idReference', primary key), the identifier of the reference type ('idReferenceType'), and a unique identifier of the reference source ('value'). The entry 'idReferenceType' is a foreign key to access table 'reference_type'.

REFERENCE_TYPE (type table of table 'reference') - stores information about the reference type, which specifies a cross-referenced database by providing a unique reference type identifier ('idReferenceType'), the name of the cross-referenced database ('name'), see Table 3.8, and a valid url ('url').

RESPONSIBILITY (attribute table of table 'x_module_user') - stores information about possible user roles on a module by providing a unique responsibility identifier ('idResponsibility', primary key) and a descriptive name ('responsibility'). A user can be modeller, submitter, or curator.

SUBMISSION (main table) - stores information about the module submissions by providing a unique submission identifier ('idSubmission', primary key), the user identifier ('idUser'), the name of the submitted module ('name'), the submission date ('date'), and the submission state ('state'), which can either be 0 (not released) or 1 (released). The entry 'idUser' is a foreign key to access table 'login'.

TERMS_TYPE (type table of table 'module') - stores information about the terms of use by providing a unique identifier for each term type ('idTermType', primary key) and the term ('term').

USER (attribute table of table login) - stores more detailed information about the user by providing the user identifier ('idUser', primary key), his name ('firstName', 'lastName'), e-mail address ('e-mail'), and phone number ('phone'). The entry 'idUser' is a foreign key to access table 'login'.

x_module_collection (linkage table) - links entries in table 'module' and table 'collection' using the corresponding primary keys 'idModule' and 'idCollection' as foreign keys.

x_module_edge (linkage table) - links entries in table 'module' and table 'edge' using the corresponding primary keys 'idModule' and 'idEdge' as foreign keys.

x_module_firingrate (linkage table) - links entries in table 'module' and table 'firingrate' using the corresponding primary keys 'idModule' and 'idFiringrate' as foreign keys.

x_module_marking (linkage table) - links entries in table 'module' and table 'marking' using the corresponding primary keys 'idModule' and 'idMarking' as foreign keys.

x_module_multiplicity (linkage table) - links entries in table 'module' and table 'multiplicity' using the corresponding primary keys 'idModule' and 'idMultiplicity' as foreign keys.

x_module_node (linkage table) - links entries in table 'module' and table 'node' using the corresponding primary keys 'idModule' and 'idNode' as foreign keys.

x_module_node_desc (linkage table) - links entries in table 'module' and table 'node_desc' using the corresponding primary keys 'idModule' and 'idDesc' as foreign keys.

x_module_node_reference (linkage table) - links entries in table 'module' and table 'x_node_reference' using the corresponding primary keys 'idModule' and 'idNodeRef' as foreign keys.

x_module_parameter (linkage table) - links entries in table 'module' and table 'parameter' using the corresponding primary keys 'idModule' and 'idParameter' as foreign keys.

x_module_parameter_desc (linkage table) - links entries in table 'module' and table 'parameter_desc' using the corresponding primary keys 'idModule' and 'idDesc' as foreign keys.

x_module_parameter_reference (linkage table) - links entries in table 'module' and table 'parameter_reference' using the corresponding primary keys 'idModule' and 'idParaRef' as foreign keys.

x_module_parameter_value (linkage table) - links entries in table 'module' and table 'parameter_value' using the corresponding primary keys 'idModule' and 'idValue' as foreign keys.

x_module_place_part (linkage table) - links entries in table module and table ‘x_place_part‘ using the corresponding primary keys ‘idModule‘ and ‘idPlacePart‘ as foreign keys.

x_module_place_part_reference (linkage table) - links entries in table ‘module‘ and ‘table‘ x_place_part_reference using the corresponding primary keys ‘idModule‘ and ‘idPlacePartRef‘ as foreign keys.

x_module_reference (linkage table) - links entries in table ‘module‘ and table ‘reference‘ using the corresponding primary keys ‘idModule‘ and ‘idPlacePartRef‘ as foreign keys.

x_module_submission (linkage table) - links entries in table ‘module‘ and table ‘submission‘ using the corresponding primary keys ‘idModule‘ and ‘idSubmission‘ as foreign keys.

x_module_user (linkage table) - links entries in table ‘module‘, table ‘login‘, and table ‘responsibility‘ using the corresponding primary keys ‘idModule‘, ‘idUser‘ and ‘idResponsibility‘ as foreign keys.

x_node_reference (linkage table) - links entries in table ‘node‘ and table ‘reference‘ using the corresponding primary keys ‘idNode‘ and ‘idReference‘ as foreign keys. For each entry a unique identifier is defined (‘idNodeRef‘), which is used as foreign key in table ‘x_module_node_reference‘.

x_parameter_reference (linkage table) - links entries in table ‘parameter‘ and table ‘reference‘ using the corresponding primary keys ‘idParameter‘ and ‘idReference‘ as foreign keys. For each entry a unique identifier is defined (‘idParaRef‘), which is used as foreign key in table ‘x_module_parameter_reference‘.

x_part_reference (linkage table) - links entries in table ‘part‘ and table ‘reference‘ using the corresponding primary keys ‘idPart‘ and ‘idReference‘ as foreign keys. For each entry a unique identifier is defined (‘idPartRef‘), which is used as foreign key in table ‘x_place_part_reference‘.

x_place_part (linkage table) - links entries in table ‘node‘, where ‘idNodeType‘ refers to places, and table ‘part‘ using the corresponding primary keys ‘idNode‘ (named ‘idPlace‘) and ‘idPart‘ as foreign keys. For each entry a unique identifier is defined (‘idPlacePart‘), which is used as foreign key in table ‘x_module_place_part‘.

x_place_part_reference (linkage table) - links entries in table ‘node‘, where ‘idNodeType‘ refers to places, and table ‘x_part_reference‘ using the corresponding primary keys ‘idNode‘ (named ‘idPlace‘) and ‘idPartRef‘ as foreign keys. For each entry a unique identifier

is defined ('idPlacePartRef'), which is used as foreign key in table 'x_module_place_part_reference'.

4.3 *Relation between BMK Datbase and Module Definition*

This section requires:

- *Petri nets [82], Section 2.1*
- *MySQl[20], Section 4.2*
- *Module definition, Section 3.1.1*
- *Module types, Section 3.1.2*
- *Petri net representation of modules, Section 3.1.3*
- *BMKml, Section 3.2*

The BMKDB scheme in Section 4.2 holds information on the Petri net graph and the annotation of a module M_{c_0}. Here, we explain in more detail, how the physical instance of a module is matched with its database instance. In the following a column of a table is specified by 'table_name'.'column_name'.

First of all, only registered users, that are logged-in can submit a module to the BMKDB. The log-in information (email, password) of each registered user is stored in the 'login' table, see Section 4.2, such that

↪ 'login'.'idUser' = *auto-index*
'login'.'name' = email of user
'login'.'passwordSHA' = password of user
'login'.'date' = *time-stamp of registration date*

Each submitted module M_{c_0} generates a new entry in table 'submission', see Section 4.2, such that

↪ 'submission'.'idSubmission' = *auto-index*
'submission'.'idUser' = 'login'.'idUser' of user
'submission'.'name' = auto-generated folder-name
'submission'.'date' = *time-stamp of submission date*
'submission'.'state' = 0 (must be approved by the administrator)

After manual curation of the model and annotation file by the administrator, the files are processed and the content is stored in the BMKDB. The processing starts with the model file holding the Petri net graph $\mathcal{N}(M_{c_0}) = \{P, T, F, f, v, m_0\}$ of a module M_{c_0}:

- $\forall p \in P$ a new entry in the table 'node' is generated, such that
  ↪ 'node'.'idNode' = *auto-index*
  'node'.'idNodeType' = 'node_type'.'idNodeType' of place
  'node'.'name' = name of p
- $\forall t \in T$ a new entry in the table 'node' is generated, such that
  ↪ 'node'.'idNode' = *auto-index*
  'node'.'idNodeType' = 'node_type'.'idNodeType' of transition
  'node'.'name' = name of t
- $x \in P \cup T$ a new entry in the table 'x_module_node' is generated, such that
  ↪ 'x_module_node'.'idNode' = 'node'.'idNode' of x
  'x_module_node'.'idModule' = 'module'.'idModule' of M_{c_0}
- $\forall f_{type}(x, y) \in F$, $x, y \in P \cup T$ a new entry in the table 'edge' and 'multiplicity' is generated, as well as in table 'x_module_edge' and 'x_module_multiplicity', such that
  ↪ 'edge'.'idEdge' = *auto-index*,

'edge'.'idStartNode' = 'node'.'idNode' of x
'edge'.'idEndNode' = 'node'.'idNode' of y
'edge'.'idEdgeType' = 'edge_type'.'idEdgeType' of $type$
↪ 'multiplicity'.'idMultiplicity' = *auto-index*
'multiplicity'.'idEdge' = 'edge'.'idEdge' of $f_{type}(x, y)$
'multiplicity'.'value' = $f_{type}(x, y)$
↪ 'x_module_edge'.'idNode' = 'edge'.'idEdge' of $f_{type}(x, y)$
'x_module_edge'.'idModule' = 'module'.'idModule' of M_{c_0}
↪ 'x_module_multiplicity'.'idMultiplicity' = 'multiplicity'.'idMultiplicity' of $f_{type}(x, y)$
'x_module_multiplicity'.'idModule' = 'module'.'idModule' of M_{c_0}

- $\forall h(t)$,$t \in T$ a new entry in the table 'firingrate' and 'x_module_firingrate' is generated, such that
  ↪ 'firingrate'.'idFiringrate' = *auto-index*
  'firingrate'.'idNode' = 'node'.'idNode' of t
  'firingrate'.'function' = $h(t)$
  ↪ 'x_module_firingrate'.'idFiringrate' = 'firingrate'.'idFiringrate'
  'x_module_firingrate'.'idModule' = 'module'.'idModule' of M_{c_0}
- $\forall m_0(p)$, $p \in P$ a new entry in the table 'marking' and 'x_module_marking' is generated, such that
  ↪ 'marking'.'idMarking' = *auto-index*
  'marking'.'idNode' = 'node'.'idNode' of p
  'marking'.'value' = $m_0(p)$
  ↪ 'x_module_marking'.'idMarking' = 'marking'.'idMarking'
  'x_module_marking'.'idModule' = 'module'.'idModule' of M_{c_0}
- If a set of parameters $K = \{k_1, \ldots, k_n\}$, $n \geq 1$ is defined in the Snoopy-file [120] holding the Petri net graph of a module $(N)(M)$ to specify the marking or to formulate the firing rates, each parameter $k \in K$ generates a new entry in table 'parameter', 'parameter_value', 'x_module_parameter' and 'x_module_parameter_value', such that
  ↪ 'parameter'.'idParameter' = *auto-index*
  'parameter'.'name' = name of k
  ↪ 'parameter_value'.'idValue' = *auto-index*
  'parameter_value'.'idParameter' = 'parameter'.'idParameter'
  'parameter_value'.'value' = k
  ↪'x_module_parameter'.'idParameter' = 'parameter'.'idParameter'
  'x_module_parameter'.'idModule' = 'module'.'idModule' of M_{c_0}
  ↪ 'x_module_parameter_value'.'idValue' = 'parameter_value'.'idValue'
  'x_module_parameter_value'.'idModule' = 'module'.'idModule' of M_{c_0}

The BMKml of the module annotation file can be mapped to the BMKdb scheme as follows (attributes of an element are printed in bold):

- The MODULE element generates a new entry in table ‘module‘ and ‘reference‘, as well as in table ‘x_module_reference‘ table, such that
  ↪ ‘module‘.‘idModule‘ = *auto-index*
  ‘module‘.‘name‘ = **name** of MODULE element
  ‘module‘.‘state‘ = *set by administrator*
  ‘module‘.‘releaseDate‘ = *time-stamp of the release date*
  ‘module‘.‘idModuleType‘ = ‘module_type‘.‘idModuleType‘ of entry for **type** of MODULE element
  ↪ ‘reference‘.‘idReference‘ = *auto-index*
  ‘reference‘.‘idReferenceType‘ = reference_type.idReferenceType of entry for **dbName** of MODULE element
  ‘reference‘.‘value‘ = **id** of MODULE element
  ↪ ‘x_module_reference‘.‘idReference‘ = ‘reference‘.‘idReference‘
  ‘x_module_reference‘.‘idModule‘ = ‘module‘.‘idModule‘ of M_{c_0}
- The TERMS element, which is a child of the INFOLIST element, generates a new entry in the table ‘terms_type‘, and specifies the value for entry ‘module‘.‘idTermsType‘, such that
  ↪ ‘terms_type‘.‘idTermsType‘ = *auto-index*
  ‘terms_type‘.‘terms‘ = content of TERMS element
  ↪ ‘module‘.‘idTermsType‘ = ‘terms_type‘.‘idTermsType‘
- The DATE element, which is also a child of the INFOLIST element, specifies the value for the ‘module‘.‘creationDate‘ and ‘module‘.‘modificationDate‘, such that
  ↪ ‘module‘.‘creationDate‘ = **creationDate** of DATE element
  ‘module‘.‘modificationDate‘ = **modificationDate** of DATE element
- The AUTHORLIST element is also a child of the INFOLIST element. Each AUTHOR child element of AUTHORLIST generates a new entry in table ‘user‘ and ‘x_module_user‘, such that
  ↪ ‘user.‘idUser‘ = *auto-index*
  ‘user‘.‘firstName‘ = **firstName** of AUTHOR element
  ‘user‘.‘lastName‘ = **lastName** of AUTHOR element
  ‘user‘.‘email‘ = **email** of AUTHOR element
  ↪ ‘x_module_user‘.‘idUser‘ = ‘edge‘.‘idUser‘
  ‘x_module_user‘.‘idModule‘ = ‘module‘.‘idModule‘ of M_{c_0}
  ‘x_module_user‘.‘idResponsibility‘ = ‘responsibility‘.‘idResponsibility‘ of modeller
- The DESCRIPTION element, which is also a child of the INFOLIST element, specifies the value for entry ‘module‘.‘description‘, such that
  ↪ ‘module‘.‘description‘ = content of DESCRIPTION element

- Each PLACE child element of PLACELIST and each TRANSITION child element of TRANSITIONLIST generates a new entry in table ‘node_desc‘ and ‘x_module_node_desc‘, such that
  ↪ ‘node_desc‘.idDesc‘ = *auto-index*
  ‘node_desc‘.idNode‘ = ‘node‘.‘idNode‘ of **name** of PLACE or TRANSITION element

'node_desc'.'description' = content of DESCRIPTION element (child of PLACE or TRANSITION element)
↪ 'x_module_node_desc'.idDesc' = node_desc'.idDesc'
'x_module_node_desc'.idModule' = 'module'.'idModule' of M_{c_0}

- Each GCOMPONENT element (child of GCOMPONENTLIST element) and NGCOMPONENT element (child of NGCOMPONENTLIST element), both grandchildren of COMPONENTLIST element a child of each PLACE element, generates a new entry in table 'part','x_place_part', and 'x_module_place_part', such that
  ↪ 'part'.'idPart' = *auto-index*
  'part'.'name' = **name** of GCOMPONENT or NGCOMPONENT element
  'part'.'idPartType' = 'part_type'.'idPartType' of **type** (GCOMPONENT element) or non-genetic component (NGCOMPONENT element)
  ↪ 'x_place_part'.'idPlacePart' = *auto-index*
  'x_place_part'.'idPart' = 'part'.'idPart'
  'x_place_part'.'idPlace' = 'node'.'idNode' of **name** of PLACE element
  ↪ 'x_module_place_part'.'idPlacePart' = 'x_place_part'.'idPlacePart'
  'x_module_place_part'.'idModule' = 'module'.'idModule' of M_{c_0}
- Each PARAMETER child of PARAMETERLIST element generates a new entry in table 'parameter_desc' and 'x_module_parameter_desc', such that
  ↪ 'parameter_desc'.idDesc' = *auto-index*
  'parameter_desc'.idParameter' = 'node'.'idParameter' of **name** of PARAMETER element
  'parameter_desc'.'description' = content of DESCRIPTION element (child of PARAMETER element)
  ↪ 'x_module_parameter_desc'.idDesc' = parameter_desc'.idDesc'
  'x_module_parameter_desc'.idModule' = 'module'.'idModule' of M_{c_0}

- The DBREFLIST element is a child of GCOMPONENT and NGCOMPONENT elements, but also of TRANSITION and PARAMETER element. Each DBREF child of DBREFLIST element generates a new entry in table 'reference', such that
  ↪ 'reference'.'idReference' = *auto-index*
  'reference'.'idReferenceType' = reference_type.idReferenceType of **dbName** of DBREF element
  'reference'.'value' = **id** of DBREF element
  – In case of the parent elements GCOMPONENT and NGCOMPONENT an additional entry is generated in table 'x_part_reference', 'x_place_part_reference', and 'x_module_place_part_reference',

such that
↪ ‘x_part_reference‘.‘idPartRef‘ = *auto-index*
‘x_part_reference‘.‘idPart‘ = ‘part‘.‘idPart‘ of **name** of gComponent or ngComponent element
‘x_part_reference‘.‘idReference = ‘reference‘.‘idReference‘
↪ ‘x_place_part_reference‘.‘idPlacePartRef‘ = *auto-index*
‘x_place_part_reference‘.‘idPartRef‘ = ‘x_part_reference‘.‘idPartRef‘
‘x_place_part_reference‘.‘idPlace‘ = ‘node‘.‘idNode‘ of **name** of place element
↪ ‘x_module_place_part_reference‘.‘idPlacePartRef‘ = ‘x_place_part_reference‘.‘idPlacePartRef‘
‘x_module_place_part_reference‘.‘idModule‘ = ‘module‘.‘idModule‘ of M_{c_0}

– In case of the parent element transition an additional entry is generated in table ‘x_node_reference‘ and ‘x_module_node_reference‘, such that
↪ ‘x_node_reference‘.‘idNodeRef‘ = *auto-index*
‘x_node_reference‘.‘idReference‘ = ‘reference‘.‘idReference‘
‘x_node_reference‘.‘idNode‘ = ‘node‘.‘idNode‘ of **name** of transition element
↪ ‘x_module_node_reference‘.‘idNodeRef‘ = ‘x_node_reference‘.‘idNodeRef‘
‘x_module_node_reference‘.‘idModule‘ = ‘module‘.‘idModule‘ of M_{c_0}

– In case of the parent element parameter an additional entry is generated in table ‘x_parameter_reference‘ and ‘x_module_parameter_reference‘, such that
↪ ‘x_parameter_reference‘.‘idParaRef‘ = *auto-index*
‘x_parameter_reference‘.‘idReference‘ = ‘reference‘.‘idReference‘
‘x_parameter_reference‘.‘idParameter‘ = ‘node‘.‘idParameter‘ of **name** of parameter element
↪ ‘x_module_parameter_reference‘.‘idParaRef‘ = ‘x_node_parameter‘.‘idParaRef‘
‘x_module_parameter_reference‘.‘idModule‘ = ‘module‘.‘idModule‘ of M_{c_0}

- The pubRefList element is a child of the infoList, place, transition, and parameter element. Each pubRef child of pubRefList element generates a new entry in table ‘reference‘, such that
↪ ‘reference‘.‘idReference‘ = *auto-index*
‘reference‘.‘idReferenceType‘ = reference_type.idReferenceType of entry for **dbName** of pubRef element
‘reference‘.‘value‘ = **id** of pubRef element

– In case of the parent element infoList an additional entry is generated in table ‘x_module_reference‘, such that
↪ ‘x_module_reference‘.‘idReference‘ = ‘reference‘.‘idReference‘
‘x_module_reference‘.‘idModule‘ = ‘module‘.‘idModule‘ of M_{c_0}

- In case of the parent elements PLACE and TRANSITION an additional entry is generated in table ‘x_node_reference‘ and ‘x_module_node_reference‘, such that
  ↪ ‘x_node_reference‘.‘idNodeRef‘ = *auto-index*
  ‘x_node_reference‘.‘idReference‘ = ‘reference‘.‘idReference‘
  ‘x_node_reference‘.‘idNode‘ = ‘node‘.‘idNode‘ of entry for **name** of PLACE or TRANSITION element
  ↪ ‘x_module_node_reference‘.‘idNodeRef‘ = ‘x_node_reference‘.‘idNodeRef‘
  ‘x_module_node_reference‘.‘idModule‘ = ‘module‘.‘idModule‘ of M_{c_0}
- In case of the parent element PARAMETER an additional entry is generated in table ‘x_parameter_reference‘ and ‘x_module_parameter_reference‘, such that
  ↪ ‘x_parameter_reference‘.‘idParaRef‘ = *auto-index*
  ‘x_parameter_reference‘.‘idReference‘ = ‘reference‘.‘idReference‘
  ‘x_parameter_reference‘.‘idParameter‘ = ‘node‘.‘idParameter‘
  of entry for **name** of PARAMETER element
  ↪ ‘x_module_parameter_reference‘.‘idParaRef‘ = ‘x_node_parameter‘.‘idParaRef‘
  ‘x_module_parameter_reference‘.‘idModule‘ = ‘module‘.‘idModule‘ of M_{c_0}

As has been mentioned before, the user can create collections and store modules in collections, see Section 4.1. If the user adds a new collection, a new entry in table ‘collection‘ will be added, such that
↪ ‘collection‘.‘idCollection‘ = *auto-index*
‘collection‘.‘idUser‘ = ‘login‘.‘idUser‘ of user
‘collection‘.‘name‘ = collection_name
Each module M_{c_0} added to a collection generates a new entry in table ‘x_module_collection‘, such that
↪ ‘x_module_collection‘.‘idCollection‘ = ‘collection‘.‘idCollection‘
‘x_module_collection‘.‘idModule‘ = ‘module‘.‘idModule‘ of M_{c_0}

According to these relations between the BMKDB scheme and the Petri net graph and annotation of a module, MySQL queries can be designed to either select, insert or delete information. The BMKDB scheme supports the versioning of modules and all its related content, by linking each entry to the module instance of M_{c_0} in table ‘module‘.

5
Case Studies

So far, we introduced the modularization concept, module annotation, model composition and module construction as part of the BMKfr, as well the web-tool supporting the BMKfr. In this chapter, we will present three case studies to demonstrate our approach on experimental biological systems.

The first case study is the interleukin-6 induced JAK-STAT signalling in eukaryotes, which involves a negative gene regulatory feedback loop. JAK-STAT signalling is involved in several essential regulatory processes of human cells. The modular model is composed of ten modules representing important key players of the interleukin-6 induced JAK-STAT signalling. The model composed of highly detailed modules integrates different module types and reflects the experimental data obtained for the JAK-STAT signalling.

The second case study models signalling processes involved in nociception. Nociception is the encoding and processing of noxious stimuli in the nervous system. Thus, nociception is the ability of a body to sense potential harm, which might lead to the perception of pain. On the molecular basis, nociception involves a plethora of components like receptors, ion channels, kinases, signalling proteins, second messengers, etc. The modular model is composed of a huge number of modules. This case study demonstrates the advantageous of an *a priori* modularization concept to model complex systems.

The third case study addresses the phosphate regulatory network in enterobacteria. Inorganic phosphate is essential for the maintenance of the biosynthesis of nucleic acids and other cellular components in enteric bacteria like *Escherichia coli* by being involved in gene regulatory processes. The composed model integrates of protein, gene, mRNA, and protein degradation modules. Here, we show that the modular modelling concept can also be applied to biomolecular networks of prokaryotic organisms.

5.1 JAK-STAT Signalling

JAK-STAT signalling is of fundamental importance in regulatory processes of human cells. A variety of cytokines such as interleukins, interferons, and growth factors can activate the JAK-STAT signalling [39, 41]. Thus, JAK-STAT signalling plays a significant role

in development, cell proliferation, cell migration, and inflammatory processes [25]. In particular, dysregulation, especially constitutive activation of JAK-STAT signalling, was found in cancerous, autoimmune, and inflammatory diseases. In the following, we will first explain the molecular mechanism of JAK-STAT signalling induced by the cytokine interleukin-6. Next, we will introduce the modules describing the mechanistic details of the components involved in JAK-STAT signalling, and explain one of the components in more detail. Finally, we show a simulation experiment describing the dynamic behaviour of JAK/STAT signalling. We also refer to [118, 126], where this case study has been published previously.

As mentioned above, we consider interleukin-6 induced JAK-SAT signalling as shown in Figure 5.1. The cytokine interleukin-6 (IL-6) signals through a type I cytokine transmembrane receptor complex composed of the ligand binding IL-6 receptor-α chain (IL-6Rα) and the signal-transducing glycoprotein gp130. Binding of IL-6 to its receptor IL-6Rα and to gp130 (step 1 in Figure 5.1) causes the dimerization of the IL6Rα-gp130 receptor complex which leads to the activation of the JAK kinases by transphosphorylation (step 2). JAK is constitutively bound to gp130 in both its active and its inactive form. Active JAK phosphorylates several tyrosine residues of the cytosolic part of gp130

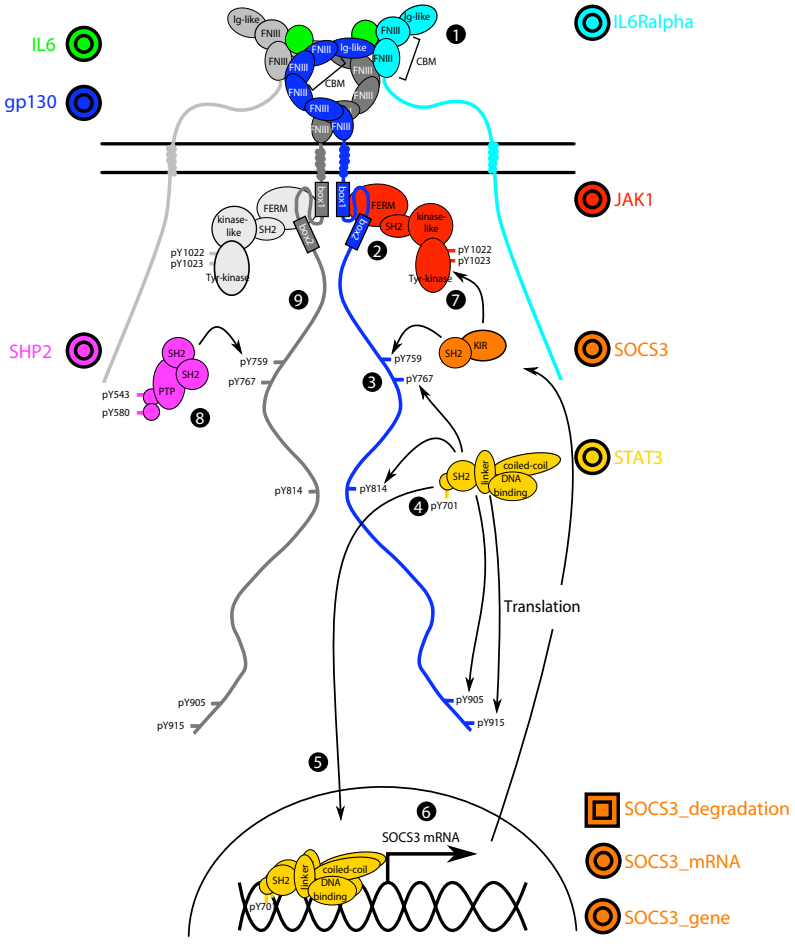


Figure 5.1: Molecular model of IL-6 induced JAK/STAT signalling. The molecular mechanism is explained in the text. Next to the involved protein, we depict the respective protein modules, in the form of coarse place. The gene and mRNA module of SOCS3 are also defined by a coarse place. The protein degradation module of SOCS3 is given as a coarse transition. The coarse nodes holding the Petri nets describe the molecular mechanisms of each component. In Figure 5.2 we exemplary depict the gp130 protein module. (adapted from [118])

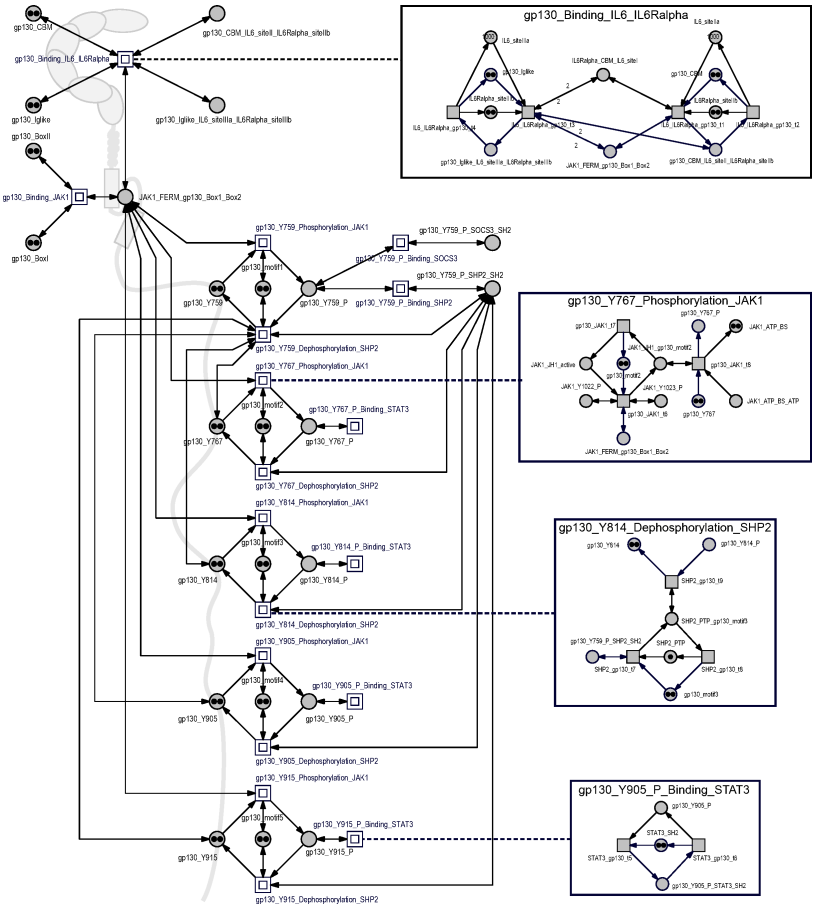


Figure 5.2: The gp130 protein module. At the extracellular part, the gp130 glycoprotein contains an Ig-like domain and a cytokine-binding module. Both domains are responsible for forming a complex with IL-6 and IL6-R. The intracellular part of gp130 has an interbox 1-2 region, where JAKs are constitutively bound. Downstream of this binding site the gp130 receptor has five critical tyrosine residues (Y759, Y767, Y814, Y905, Y915), which are phosphorylated by activated JAK. The phosphotyrosine pY759 is the binding site of SHP2 and SOCS3 via their SH2 domain, whereas STAT proteins can interact via their SH2 domain with the other phosphotyrosines in gp130. In the corresponding Petri net model of the gp130 protein, the extracellular binding site of IL-6 and the intracellular binding site of JAK1 to the Box 1 and Box 2 sites are represented as coarse transitions. Also, coarse transitions describe the phosphorylation and dephosphorylation reactions of the five tyrosines by JAK1 and SHP2, respectively. Specifically, three coarse transitions are assigned to each tyrosine residue downstream of Y759 describing the phosphorylation mechanism by JAK1, the dephosphorylation event by SHP2, and the binding interaction with STAT3. To the right of the coarse transitions representing the phosphorylation and dephosphorylation of Y759 (upper part of the figure), the Y759-P place is also connected with two coarse transitions describing the binding interaction with SHP2 and SOCS3. The boxes on the right side of the figure exemplify four interface networks that are included in the corresponding coarse transitions in the gp130 module. (text and figure adapted from [126])

(step 3). The STAT transcription factor binds to phosphotyrosines of gp130 and is subsequently phosphorylated by active JAK (step 4). Phosphorylated STAT proteins dimerize and translocate to the nucleus (step 5) to activate the transcription of multiple genes including SOCS3 (step 6). SOCS3 acts as a negative feedback regulator of JAK in binding to phosphorylated Y759 of gp130 (step 7) causing the inactivation of the JAK kinase and in turn decreasing the rate of STAT3 phosphorylation. The SHP2 phosphatase counteracts JAK by dephosphorylating phosphotyrosines of gp130 (step 8), while JAK phosphorylates SHP2, which releases SHP2 from the receptor complex (step 9).

For each of the involved components (IL-6, IL-6RA, gp130, JAK, STAT, SOCS, and SHP2), we created a protein module based on detailed knowledge of their interactions described in the literature, see Figure 5.1. In the case of SOCS3, we added a gene and an mRNA module to define its biosynthesis, as well as a protein degradation module. In Figure 5.2 we exemplary depict and expalin the gp130 protein module. The module of gp130 with its functional units (ligand binding site, binding sites for JAK1 and phosphotyrosine motifs representing binding sites for SHP2, STAT3 and SOCS3) is designed according to the 3D structural model of the gp130 protein. For the sake of a clearly readable model structure, we represent the crucial

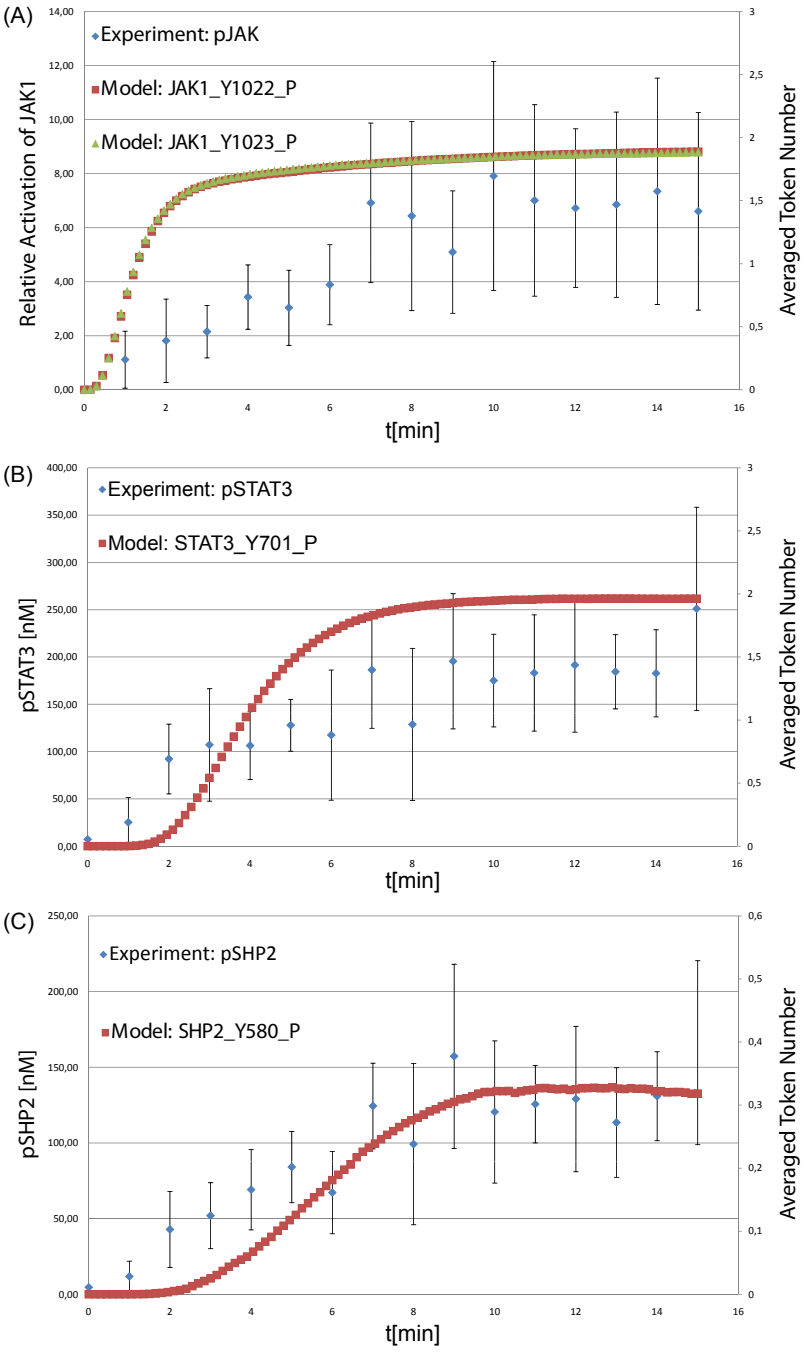


Figure 5.3: Simulation of the modularly composed JAK/STAT signalling model and comparison with experimental data. To collect experimental data, human embryonic kidney cells (HEK293) stably expressing the human IL-6 receptor a (gp80) were stimulated with a 6 minutes pulse of IL-6 (20 ng ml 1). Cells were harvested at the indicated time points and subsequently analysed by immunoblotting with specific antibodies against pSTAT3, pJAK1, pSHP2, and HSP70. The detection of HSP70 served as loading control. Data are presented as the mean standard deviation from n = 5 to 10 independent experiments. Absolute concentrations of pSTAT3 and pSHP2 were analysed by quantitative immunoprecipitation using recombinant STAT3 and SHP2 proteins as calibrators (see [119] for original data. In (A), (B), and (C) we plotted the experimental data next to the simulation results for (A) the relative activation of JAK1 (pJAK1), (B) pSTAT3, and (C) pSHP2. The model output qualitatively matches the experimental data on both the stimulus-induced phosphorylation of JAK1, STAT3 and SHP2 and the plateaus reached after removal of the IL-6 stimulus. (text and figure adapted from [126])

functional units and related molecular events by coarse transitions. Each phosphotyrosine (see also Figure 5.1) is connected to at least three coarse transitions, one describing the phosphorylation, another describing the dephosphorylation of the receptor motif.

The remaining coarse transitions describe the binding of an interaction partner to the respective phosphotyrosine motif. Phosphorylation and dephosphorylation of the receptor motif, as well as binding of a signalling component, occur in principle through the same mechanism for each phosphotyrosine motif within gp130. Because of reoccurring motifs, the network structures describing the detailed

kinetic mechanisms are represented as submodels of the respective coarse transitions (Figure 5.2, boxes).

The constructed modules have been tested for their structural integrity using Charlie [141]. The structural analysis of the modules confirmed the expected structural properties as given in Section 3.1.3. In a next step, the set of constructed and validated modules yields modularly composed model describing IL-6-induced JAK/STAT signalling. The behaviour of the composed model for IL-6-induced JAK/STAT signalling was then tested against previously published experimental data [119]. In the experimental set-up HEK293 cells stably expressing gp80 were stimulated with a 6 minutes pulse of IL-6. The cells were harvested in one minute time intervals to analyse the initial activation of the JAK/STAT pathway in detail. The dynamics of JAK1, STAT3 and SHP2 phosphorylation were measured by quantitative immunoblotting as described earlier [119]. All three species are not phosphorylated in the absence of IL-6. Stimulation with IL-6 induces a fast increase of phosphorylation of JAK1, STAT3 and SHP2 within minutes. About one minute after the initial stimulus has been removed by washing the cells, the phosphorylation of the three proteins reaches a plateau and remains constant within the experimentally analysed time frame. The model output qualitatively matches the experimental data on both the stimulus-induced phosphorylation of JAK1, STAT3 and SHP2 and the plateaus reached after removal of the IL-6 stimulus, see Figure 5.3.

5.2 *Signal Components involved in Nociception*

Pain is a complex phenomenon depending on physiological, psychological, and social aspects. On the physiological level, the sensation of pain is a result of the integration of information from the central, peripheral, and autonomous nervous system, as well as from the immune system [29]. Describing pain on a physiological level requires detailed knowledge about its molecular basis. A variety of membrane components and intracellular signalling molecules have been identified that play key roles in pain sensation, e.g. G-protein-coupled receptors (GPCR), ion channels, receptor tyrosine kinases, cytokine and hormone receptors. Those membrane components, in turn, activate a plethora of signalling cascades like the cAMP pathway and calcium signalling [29, 68]. However, the quantitative and qualitative relationships between the different intracellular signalling mechanisms acting downstream of the receptor to which those substances bind are still poorly understood [68]. Due to the complexity of the various pain aspects and the lack of knowledge about the involved molecular mechanisms, pain therapies are in many cases not successful. A mechanism-based pain therapy is mostly missing, rendering undertreated pain a serious public health issue (see [68] and references therein).

In this case study, we focus on molecular components involved in the peripheral nervous system, in particular in nociceptors, the

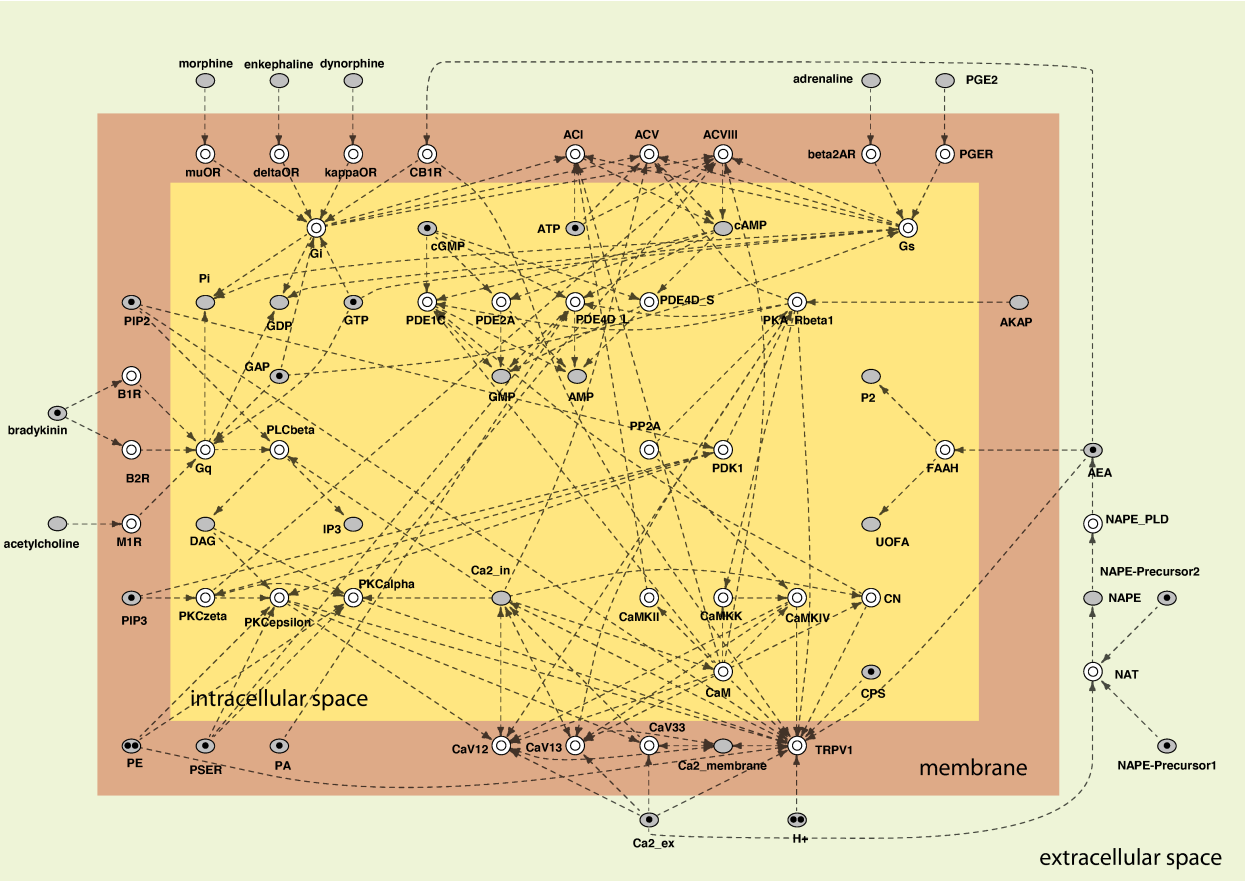


Figure 5.4: Modular nociceptive signalling model. Each coarse place contains a protein module of its related nociceptive component, see text. The oval grey shaded logical places represent the involved non-genetic components. The modules are arranged according to their localization (intracellular, membrane components, extracellular). The dashed arcs indicate the interactions among the involved components that are hidden on the top-level due to the hierarchical structuring and logical nodes. (adapted from [106])

peripheral terminals of dorsal root ganglion (DRG) neurones. Nociceptors are responsible for the detection of noxious stimuli and their transduction into electrical energy. An action potential is induced if the electrical energy reaches a threshold value, which depends on the sensitization of the nociceptors themselves. The case study on nociceptive signalling describes a subset of pain-relevant (analgesia) components involved in the excitation and inhibition of DRG neurons and has been published previously in [93, 94, 95, 106, 117].

The latest version of the nociceptive signalling model comprises 38 protein modules, see Figure 5.4, including

- members of the G-protein-coupled receptor family (GPCRs) like opioid, cannabinoid, muscarinic, prostaglandin, and β-2-adrenergic receptors,
- adenylyl cyclases (Type VIII, V, I),
- numerous protein kinases, e.g. different subunits of PKA and their isoforms, PKC (α, ϵ, ζ), and CaMK (II, IV)
- protein phosphatase 2A and calcineurin
- ion-channels, e.g. voltage-dependent calcium channels (CaV1.2, CaV1.3, CaV3.3) and capsaicin receptor (TRPV1),
- Calmodulin, and
- phospholipase Cβ

The protein modules describe the interaction with second messenger like DAG, Ca^{2+} and cAMP, which play a major role in nociception. The composed model consists of 713 places and 775 transitions spread over 325 hierarchically nested pages, with a nesting depth of 4. The modules have been constructed based on clinical pain literature (given in [93]). Details of each module in Figure 5.4 can be looked up in [93].

5.3 *Phosphate Regulatory Network in Enterobacteria*

Inorganic phosphate (P_i) limits the biosynthesis of nucleic acids and other cellular components of enteric bacteria like *Escherichia coli* (compare Figure 5.5). Thus, P_i may turn into a growth-limiting factor if its availability becomes low in the environment, even if sufficient nutrients are offered [18].

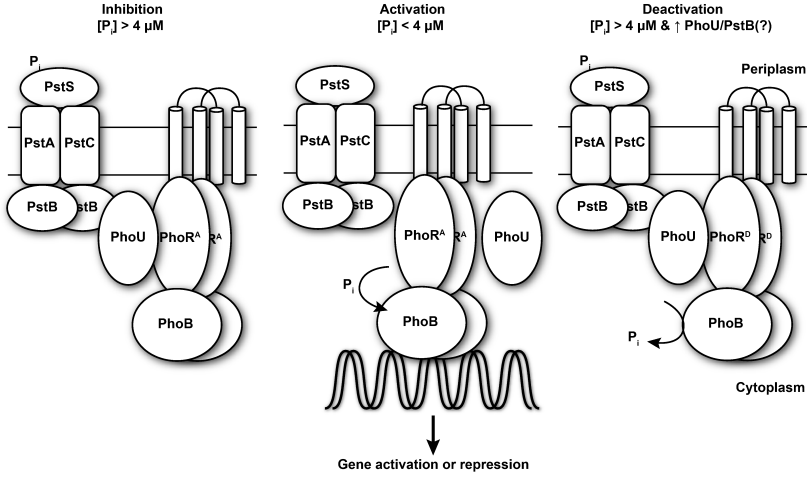


Figure 5.5: Biochemical model for sensing extracellular inorganic phosphate (P_i) and transduction of the signal to control gene expression. The PstSCAB transmembrane complex serves as an ABC transporter for the uptake of environmental P_i. At high extracellular P_i concentration, the binding protein PstS is fully saturated, and this signal is relayed to the cytoplasmic part of the receptor that forms an inhibitory complex with a second transmembrane protein, the PhoR kinase via the cytoplasmic protein PhoU. If P_i is low, the complex dissociates and the autophosphorylating kinase PhoR phosphorylates PhoB, which in its phosphorylated form binds DNA to induce gene expression. When P_i subsequently increases, the complex with PhoU has formed again, and PhoBP is dephosphorylated. (redrawn from [100])

An ABC transporter system, the PstSCAB transmembrane complex (Figure 5.5 [100]), takes up external P_i. The PstSCAB complex actively transports external P_i across the cell membrane into the cytoplasm if sufficient amounts of P_i are available. In this case, the pstSCAB transporter system forms a complex with the PhoU protein and the PhoR histidine kinase [100]. Due to the complex formation, the autophosphorylation of PhoR is inhibited by PhoU. PhoU is a chaperone-like PhoR/PhoB inhibitory protein. The PstSCAB complex becomes inactive if the external amount of P_i is too low. In consequence, PhoU dissociates from the complex and allows the autophosphorylation of PhoR. PhoR then phosphorylates and thereby activates the transcription factor PhoB. The phosphorylated form of PhoB, namely PhoBP, activates the transcription of at least 31 genes organised into 9 transcriptional units (*eda*, *phnCDEFGHIJKLMNOP*, *phoA*, *phoBR*, *phoE*, *phoH*, *psiE*, *pstSCAB-phoU*, and *ugpBAECQ*) [100]. One of the activated genes, *phoA*, encodes the PhoA protein. PhoA is a bacterial alkaline phosphatase that is exported across the membrane into the periplasm. There, PhoA converts organic phosphorous compounds to P_i. The produced P_i is then taken up into the cell to overcome the limitation.

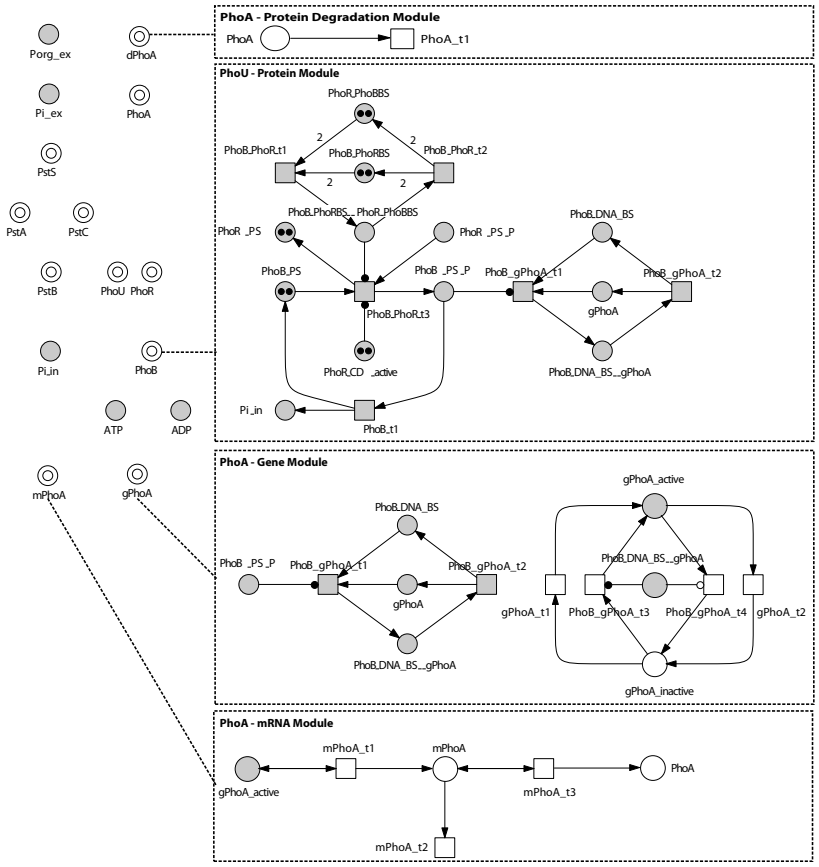


Figure 5.6: Modular phosphate regulatory network model. Each coarse place in the modular model representation contains a module. The grey shaded places represent the non-genetic components of the model e.g. ATP, ADP, inorganic phosphate P_*i*, organic phosphate source P_*org*. The modular phosphate regulatory network model consist of eight protein modules (PstS, PstC, PstA, PstB, PhoR, PhoU, PhoA, PhoB). In the case of PhoA, there exists also a gene module (gPhoA), a mRNA module (mPhoA), and a protein degradation module (dPhoA). The Petri net representation of the modules PhoB, gPhoA, mPhoA and dPhoA is given in more detail on the right side, see also text.(adapted from [116])

If sufficient amounts of P_i are available, the system is switched off again, and PhoBP is dephosphorylated.

In [114], we gave a monolithic Petri net of a simplified version of the phosphate regulatory circuitry. For simplicity reasons, the model considers only the synthesis of PhoA in response to the phosphorylated PhoB. The case study has been previously published in [116]. To cover the entire functionality, the modular Petri net model is composed of eight protein modules (PstS, PstC, PstA, PstB, PhoR, PhoU, PhoA, PhoB). In the case of PhoA, there exists a gene module (gPhoA), a mRNA module (mPhoA), and a protein degradation module (dPhoA), see Figure 5.6. The Petri net representation of the modules PhoB, gPhoA, mPhoA and dPhoA is given in Figure 5.6. The PhoB module models the interaction with PhoR including the complex formation and the transfer of the phosphate residue from PhoR to PhoB. It also represents the binding of PhoB to the regulatory site of the *phoA* gene if PhoB is phosphorylated. The gPhoA module represents the activation of the *phoA* gene in response to its interaction with the phosphorylated PhoB protein. The *phoA* gene also has a low basal activity state that can be triggered even if the phosphorylated PhoB protein is not present. The mPhoA module models the transcription of the active *phoA* gene into the respective mRNA and the translation of the mRNA into the PhoA protein. The dPhoA module describes the degradation of the PhoA protein.

6
Examples of Application

The applicability of the BMKFR relies on the generous reuse of modules in variable combination. The MIRIAM-compliant module annotation, see Section 3.2, provides information to understand and to execute the modelled molecular mechanisms described by a particular module, respectively a modularly composed model. The MIRIAM-compliance is a prerequisite to ease the reuse of models [54], and thus of modules defined in the BMKFR. Modules defined in theBMKFR can be recombined in any desired composition to create a modularly composed model. This allows the automatic generation of alternative models with variable module combinations, see Figure 6.1.

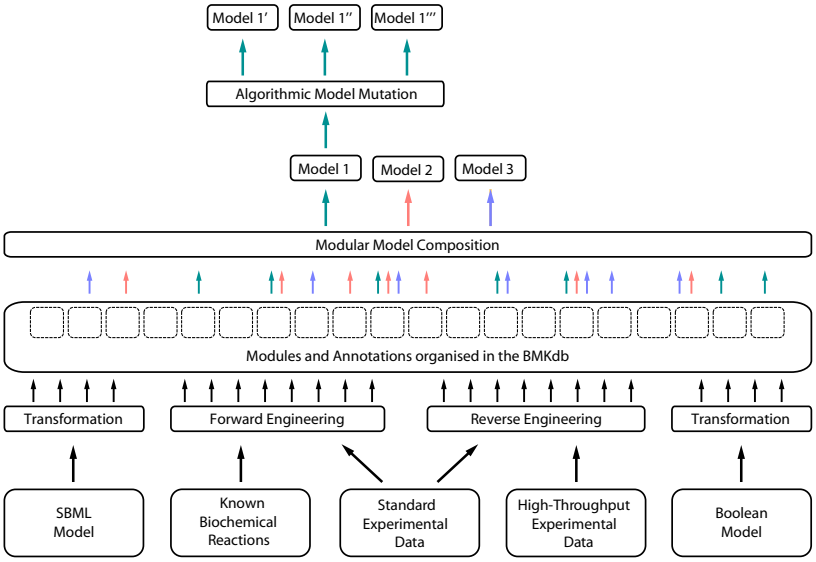


Figure 6.1: Generation of alternative models. Alternative models can be composed through the rigorous reuse and recombination of modules from different sources integrated into the BMKFR. Applying the algorithmic model mutation to modularly composed models, allows to generate even more version of alternative models with biochemically realistic modifications automatically. (adapted from [124])

Composed models may be executed as continuous, stochastic, or hybrid models or merely be used to simulate a causal sequence of discrete events qualitatively or subjected to individualised analysis workflows. Models of desired module combinations can be created by choosing a respective set of modules from the BMKDB, which holds a module repertoire of pre-existing, interchangeable, reusable, and curated (approved) modules. The different module types provide extensive flexibility in considering components of the proteome, the transcriptome, and the genome, but also of the metabolome.

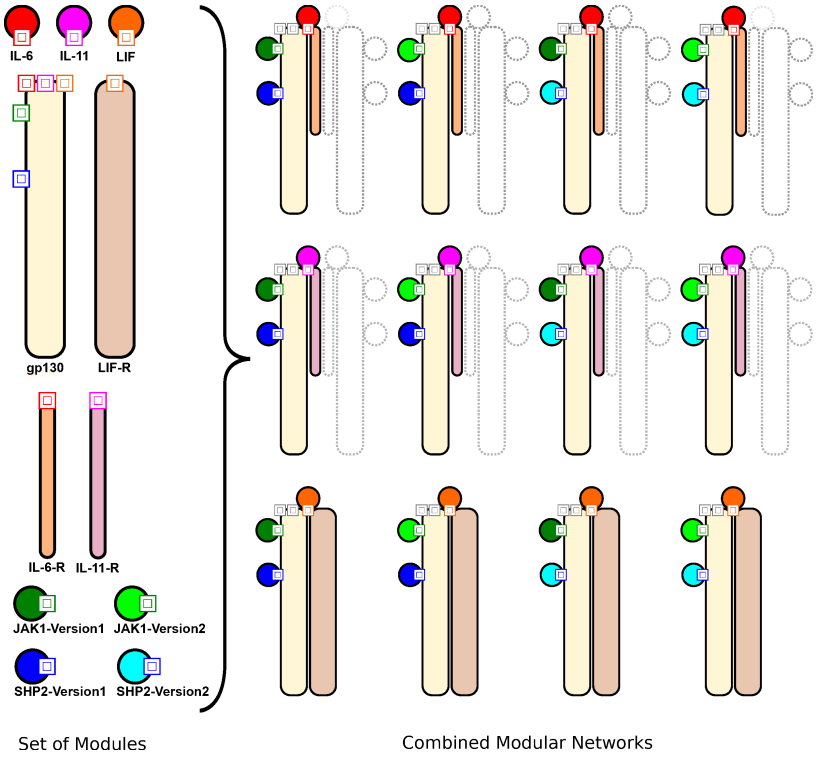


Figure 6.2: Alternative models of the JAK-STAT pathway. As already mentioned in Section 5.1, JAK-STAT signalling can be activated by several cytokines, interferons and growth factors and thus, also interacts with other receptors. The figure shows how protein modules of different cytokines, transmembrane receptors, and modules representing alternative reaction mechanisms of JAK1 and SHP2 may (left panel) be freely combined yielding alternative modularly composed models of reaction networks involving IL-6, Il-11, or LIF signal transduction or any combination thereof (right panel). The modular model composition is mediated through the interface networks, indicated by coarse transitions (boxed squares). Nodes in the interface networks are defined as logical nodes to automatically couple matching interface networks. (taken from [126])

Thus, the BMKFR can be applied to any molecular network, ranging from metabolic to signalling to genetic networks. It even supports the integration of the different network types due to interface networks among different module types. Using prototype modules and causal/allelic influence modules in combination with reverse engineering approaches, as explained in Section 3.4.2 helps to create new modules easily and to integrate experimentally obtained data directly. Also, modules can be generated from existing models, in particular Boolean models of genetic networks and models defined in SBML [48], see Section 3.4.3 and 3.4.4. The transformation of such models will tremendously increase the number of modules available for the model composition in the BMKDB. But on the other hand, the transformation also augments the reuse of the source models in the form of modules and their integration with other modules.

In composing a model from modules, one may choose interactively and from case to case which components to include, whether or not to consider protein degradation, RNA stability, or the regulation of gene expression. During the interactive modular composition of models one may also choose whether or not to consider alternative regulatory mechanisms that are suggested by the database in the form of alternative module versions, see Figure 6.1 and Figure 6.2 for an example of JAK-STAT signalling as introduced in Section 5.1.

6.1 Integration of Omic Data

The BMKFR opens exciting prospects regarding algorithmic generation of new modules. Places in a Petri net may in turn be initialised by importing high-throughput *omic* data sets. As in the case of gene expression data, thousands of genes can be considered in a modularly composed model by creating new modules from gene and mRNA prototype modules with conserved graph structure, see Section 3.4.2. Or as already explained in Section 3.4.2, high-throughput *omic* data sets can be used to construct allelic or causal influence modules, which allow integrating non-mechanistic correlations into modularly composed models. Employing gene expression data to generate modularly composed models allows to specifically consider the protein composition of different cell types under given physiological condition. The presence or absence of certain proteins described by specific gene expression patterns has a direct impact on the structure and function of regulatory networks. The automatic generation of models may provide a formal framework to evaluate correlations between transcript and protein abundance systematically on a global scale [115].

6.2 Synthetic Biology

The BMKFR might also apply to the generation of synthetically created networks with desired properties. The combination of modules of different organisms from a vast repertoire may provide an effective and efficient way to search systematically for synthetically created networks with desired properties.

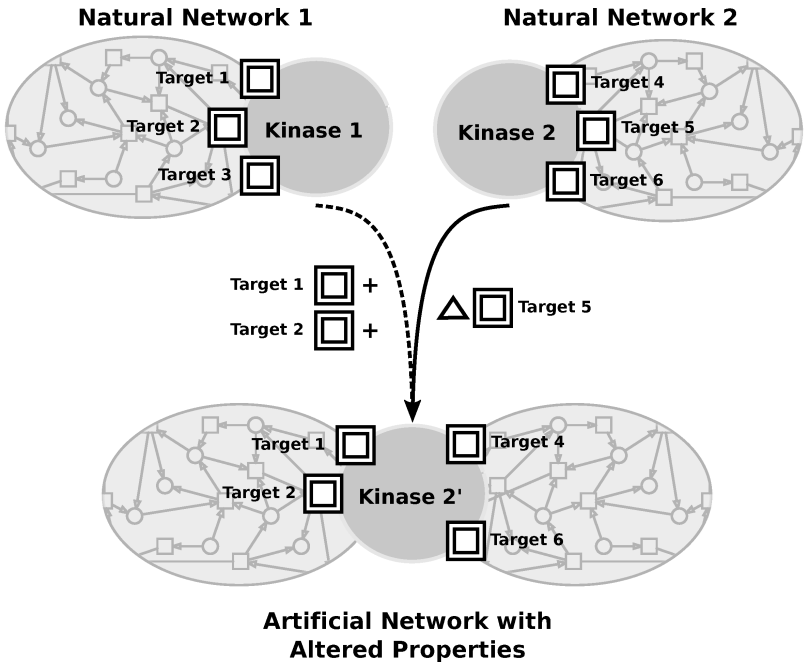


Figure 6.3: Synthetic and synthetically rewired by combing modules with altered interface networks. Synthetic and synthetically rewired networks can be generated by re-engineering modules through altering, adding, or deleting the interface networks of a module representing interaction with other components. *In silico* rewired networks can easily be queried for pre-defined properties. Based on the BMKDB supporting the BMKFR, such approaches could be automatized by integrating the module annotation. The use of the module annotation avoids combinatorial explosion by only generating biochemically realistic scenarios. (taken from [126])

In this context, one visionary approach of the BMKFR is the synthetic rewiring of modularly composed modules by altering the interactions described by the interface networks shared among the modules [126]. As shown in Figure 6.3, let us assume two naturally occurring networks with hardly any crosstalk inheriting two different kinases, Kinase 1 and Kinase 2, each phosphorylating three distinct target proteins. From these two naturally occurring networks, we want to design a new artificial network. In the artificial network, the substrate specificity of Kinase 2 is altered to functionally couple the natural networks. The Kinase 2 module now connects the two natural networks by adopting the interface networks of Target 1 and Target 2. We also eliminate the interface networks of Target 5 in Kinase 2 for demonstration purposes. The structural modifications of Kinase 2 module alter the systems behaviours. In principle, re-engineering modules by deleting existing and introducing new interface networks specific to other modules could be performed entirely automatically and systematically, while rigorously incorporating the module annotation. Here, the module annotation is essential for recombining modules in the form of biochemically realistic scenarios.

The systematic exploration of the functional potential of re-engineered (mutated) networks, could be an ongoing effort in combining the steadily increasing biological knowledge with the steadily increasing (distributed) computing power. Presumably, corresponding systematic experimental approaches will remain out of reach for the simple reason that such experiments imply tremendous amounts of work.

The evolution of molecular networks *in silico* can be performed by incorporating known mechanisms of molecular evolution (including gene shuffling, etc.) and setting selection criteria.

6.3 *Mutation Studies*

Automatically generated models from modules may be used for *in silico* mutant screens. Here, possible types of mutations could be deletions or hyperactivations. By employing the module annotation, *in silico* mutations might even consider changes in the specificity of molecular interactions. The experimental random mutagenesis screens might be complemented by *in silico* mutagenesis, see also Section 6.2, to reveal and explain the occurrence of complex phenotypes. Furthermore, in contrast to *in silico* mutagenesis, e.g. algorithmic model mutation (see Section 3.3.3) or automatic rewiring/reengineering of modules (see Section 6.2), experimental mutagenesis will be restricted to model organisms for ethical reasons. In the case of multicellular organisms, *in silico* mutagenesis might be employed to understand the consequences of somatic mutations or to overcome the restrictions of embryonic lethality. Hence, the exhaustive rewiring of networks *in silico* seems to be a promising approach to understand the functionality of networks, see also Figure 6.4.

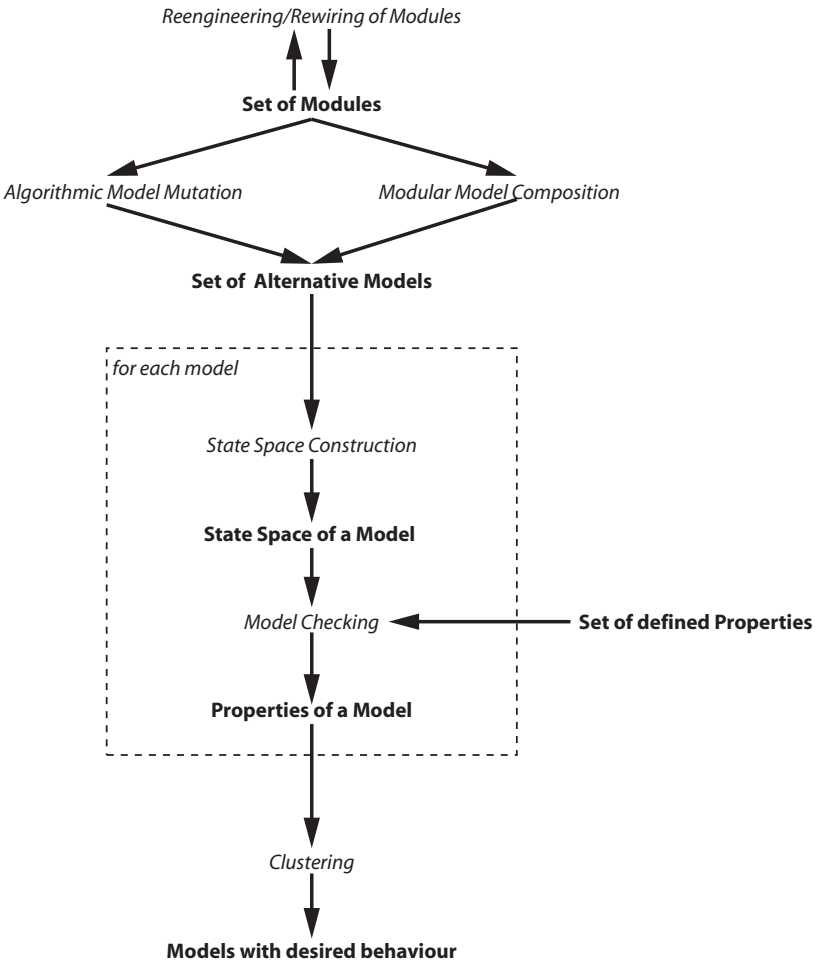


Figure 6.4: *In silico* mutagenesis. Alternative models for the *in silico* mutagenesis can be constructed by the algorithmic model mutation and by integrating (automatically) rewired/re-engineered modules in the modularly composed models. Each alternative model can be tested for defined properties using model checking methods. Therefore, it is necessary to construct the state space of each model. Models can be clustered according to their properties, which might allow identifying models with desired or interesting behaviour.

6.4 *Personalised Medicine*

As a long-term prospect, such approaches may also have relevance in the field of personalised medicine, e.g. by evaluating the consequences of altered gene expression patterns. As described in the Section 6.1, the integration of the gene expression data of a patient might affect the selection of modules subjected to the modular model composition and the initialisation of the model according to the expression level of genes, see also Figure 6.5 for illustration. Modules can be modified to represent individual mutations of a patient, e.g. the dysfunction of a particular protein due to germinal mutations (Huntington's disease, cystic fibrosis, Marfan syndrome, etc.) or somatic mutations (retinoblastoma, cancer, etc.). The reengineered modules can be reused as alternative module version in the modular model composition process. Again, approaches explained in the sections before about the synthetic rewiring of modules can be incorporated. Later on, the consequences of altered protein functions in a modularly composed model can be evaluated in a suitable analysis workflow, which might explain the patient's symptoms and predict possible targets for therapeutic mechanism-based interventions strategies.

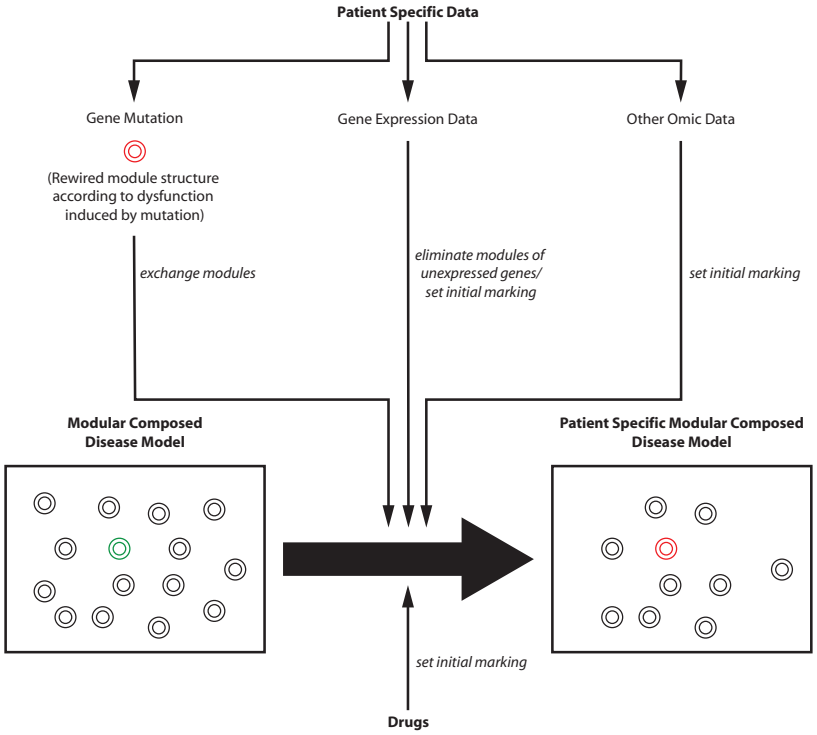


Figure 6.5: Design of patient-specific models. Patient data can be used to (1) rewire/re-engineer the structure module structure according to the effect of a mutation, (2) select modules according to gene expression data or (3) to set the marking according to gene expression data, other *omic* data, or drugs to test in a model. *Note:* Modules are represented by coarse places.

7
Conclusion & Outlook

7.1 Conclusion

The thesis aims to establish a versatile and unifying modular modelling framework for multi-scale biomodel engineering in systems biology called BioModelKit framework (BMKFR), see Chapter 3. The BMKFR integrates the ideas of modularization, redundant interface networks for the model composition, and database management. The overall benefits and advantages of the BMKFR will be summarised below. In this context, we chose Petri nets, see Chapter 2, as a modelling language due to its operational semantics and outstanding ability to model and analyse biological systems.

Modularization is one general key concept in engineering, which splits up a system into smaller parts called modules. Thus, a modularly composed system can be characterised by functionally partitioned modules with well-defined interfaces. In general, modules can be independently created, as well as reused and recombined in different systems. Even more, modules as smaller parts of a system are easier to handle and to maintain. Modularity mainly increases the flexibility in systems design. It also eases the augmentation and diminution of a system by adding or excluding modules. The application of modularization to biomodel engineering reduces the complexity of monolithic models and as a consequence thereof increases the chance of reusing models, respectively submodels in the form of modules.

The modularization concept suggested in this thesis aims at an *a priori* construction of modules and the modular composition of models. As opposed to other modularisation approaches, in our approach modules are specifically designed for the purpose of automatic model composition in variable combination. Furthermore, modules in our approach are defined on a macromolecular level, each molecular component, in particular, components with genetic information (gene, mRNAs, and proteins), are interpreted as a functional module with interfaces to represent their interactions. The module describes the functionality of a component and its interactions with other components.

Based on the different types of genetic components four distinct modules types have been defined to meet their specific requirements:

protein modules, gene modules, mRNA modules and protein degradation modules. The discrimination of these four different modules types allows to separate different types of genetic components and processes they are involved in, e.g. gene regulation, translation, transcription, protein interactions, protein degradation, etc. The module types above might exist in different versions to capture different levels of abstraction or competitive hypothesis on molecular mechanisms. So far, these four module types rely on detailed mechanistic information about the respective components. We introduced two more module types, causal and allelic influence modules, which can be applied if the mechanistic information is missing. Causal and allelic influence modules constructed by reverse engineering approaches are especially valuable for directly integrating large datasets obtained by high-throughput technologies into modularly composed models and to test for the effect on the behaviour of the model.

In engineering, the redundancy of critical components or functions of a system has the intention to increase the reliability of a system. The interface networks shared among the modules introduce redundant information. But the modular model composition as defined in the BMKfr relies on redundant interface networks. In a modularly composed model, only components specified by modules with matching interface networks can interact *in silico*. Here, redundant interface networks ensure the correct functioning of modularly composed models. Futhermore, interface networks can be employed to incorporate tissue specificity into modularly composed models.

Modules can be obtained by direct and reverse engineering approaches, as well as by transforming existing models into modules. The direct engineering approach is liable to a deep curation process from literature, bio databases, experimentally obtained data and other suitable resources. Modules constructed by this approach provide a detailed representation of the structural organisation of the modelled component (binding sites, functional sites, domains, sites for post-translational modifications) in context with its function. Reverse engineered modules rely on data-driven approaches. Mainly gene, mRNA, causal and allelic influence modules can be constructed by such approaches from large data sets as provided by the plethora of high-throughput techniques in systems biology. In the case of SBML models, models are decomposed into modules of different types according to the nature of the involved components. Boolean models are translated into an equivalent set of gene modules. The transformation of models into modules increases the universal applicability of the BMKfr, but also enhances the reusability of the source models and facilitates their integration with other network types. Modules that have been constructed by any of the approaches above have to pass a validation test, where modules have to adhere certain structural properties depending on the module type. The validation ensures a uniform quality of the module structure.

As discussed by Le Novère et al. in [54] most published biological models are lost due missing access or insufficient characterisation.

Therefore, proposed a standard for curating and encoding models called MIRIAM (Minimum Information Requested In the Annotation of Models) [54]. We integrate the ideas outlined by MIRIAM [54] into the BMKfr by defining an XML-based module annotation language called BioModelKit mark-up language (BMKml), which allows providing a rich-annotation of each module to document the underlying Petri net model and information sources incorporated during the construction process. The rather compact form a module compared to a monolithic biomodel combined with detailed annotations provided by the definition of the BMKml eases the understanding of the modelled components and molecular mechanism defined in the module. By providing access to the modules through the BMKdb, see below, and a MIRIAM compliant [54] documentation of the modules, we accomplished the fundamental prerequisite to reuse models, here in the form of defined modules, as suggested by Le Novère et al. in [54].

Based on the key principles of the modular model composition from an *ad hoc* chosen set of modules, we introduced two possible variations of the modular model composition. The first one integrates the idea of algorithmic model mutation. The algorithmic model mutation allows mimicking single/double gene knock-outs by systematically deleting modules from the modularly composed model. Another approach simulates structural mutations of a component by modifying the underlying model structure of a module. This method can also be guided by annotations mapped to the module. The application of the algorithmic model mutation generates sets of alternative models, which can be submitted to individual analysis workflows to identify models with desired behaviours. Such models might be beneficial to increase the understanding of the modelled systems and thus, to make predictions about their behaviour.

The second variation extends the modularly composed model with spatial information. Each component in the modularly composed model is equipped with spatial attributes, to store its current location. The spatial attributes of each component allow keeping track of its movement depended on of its state of interaction. Based on the spatial extension of modularly composed models important spatial aspects of cells and tissues can be integrated into a model without rebuilding it. Spatial aspects of cells and tissues might include the cell size and geometric shape, the cellular compartmentalization, and distribution of molecular components. Those spatial aspects might be of importance for the behaviour of a system and can be investigated *in silico* based on the proposed spatial extension of modularly composed models.

Our *de novo* modularization approach facilitates the handling of models. Since modules of biomolecular systems are easier to maintain by steady curation processes and to update. The achieved modular representation of models facilitates the integration of new experimental insights and hypotheses. Modules defined in theBMKfr can be recombined in any desired composition to create a modularly composed model, which allows generating alternative models with

variable module combinations automatically. Composed models may be executed as continuous, stochastic, or hybrid models or merely be used to simulate a causal sequence of discrete events qualitatively or subjected to individualised analysis workflows. Through the coupling of different module types in a composed model metabolic, signalling and gene regulatory networks can be easily integrated with each other. The different module types provide extensive flexibility in considering components of the proteome, the transcriptome, and the genome. Thus, the user can decide, which processes to include, e.g. protein degradation, RNA stability, or the regulation of gene expression, in the composed model by interactively choosing a set of respective modules. Alternative models can also be obtained by applying the model mutation algorithms. Using prototype modules or allelic and causal influence modules in combination with reverse engineering approaches helps to create new modules easily and to integrate experimentally obtained data directly. During the interactive modular composition of models, one may also choose whether or not to consider alternative regulatory mechanisms that are suggested by the database in the form of alternative module versions. Analysing the set of alternative models due to the rigorous reuse of modules might be extremely helpful to identify models with desired or interesting behaviour.

Databases are vital for the handling of large datasets as they occur in all fields, as well as in systems biology due to the high-throughput technologies. In consequence, organising the growing number of models in databases is inevitable. Therefore, we decided to *ab initio* store modules defined in the BMKfr in a database, called biomodel kit database (BMKdb), which is a relational MySQL database, see Chapter 4. The BMKfr offers not only a repository for the model and annotation files of a module. The BMKdb explicitly stores and links the content of the model and annotation files. This allows the strict organisation of modules. Modules can be retrieved by any matching criteria from the BMKdb. The database structure also supports the module versioning. Distinct modules of one and the same component, due to different abstraction levels or competing hypothesis on molecular mechanisms, can be integrated into the BMKdb. Modules of different versions can be easily exchanged in the modularly composed model.

A web-interface provides access to BMKdb, called BioModelKit web-interface (BMKwi, `www.biomodelkit.com`), and facilitates other interested users to work with the modules stored in the BMKdb. Through the BMKwi the user can browse and search for modules. Modules can be organised in user-defined collections. The BMKdb supports the automatization of the modular model composition process. From a module collection, the user can call the model composition, which comprises three different features: (1) modular composition of the unmodified model, (2) modular composition of algorithmically mutated models or (3) modular composition of the unmodified model and its spatial extension. The user is also allowed to submit

modules, which are released after a manual curation process by the administrator. Through the BMKwi the user gets also support to prepare the model annotation of the submitted modules. Users, and particularly experts on specific molecular components are asked to provide comments to the released modules. This interaction hopefully guarantees a high quality of the content offered by the BMKfr.

We proved the applicability of our proposed modular modelling concept with three case studies investigating the interleukin-6 induced JAK-STAT signalling, nociceptive mechanisms involved in the perception of pain, and the phosphate regulatory network in enterobacteria, see Chapter 5. This case studies demonstrate how to handle large networks with the proposed modularization concept of the BMKfr; its power on integrating information on the metabolomic, proteomic, transcriptomic, and genomic level; and its applicability to prokaryotic and eukaryotic systems.

The BMKfr has great potentials within the cope of systems biology based on the reusability of modules and their variable recombination, see Chapter 6. The ideas introduced in this thesis support the integration different kinds of *omic* data into modularly composed models. Approaches in synthetic biology can be supported by the *in silico* generation of synthetic and synthetically rewired networks based on the proposed modular modelling concept. The concept of synthetic and synthetically rewired networks, as well as the algorithmic model mutation, might also complement experimental mutagenesis studies. Also, the BMKfr might have prospects in personalised medicine by integrating patient specific data into modularly composed models.

Finally, the BMKfr with all its features introduced in this thesis is a versatile and unifying framework for multi-scale biomodel engineering with excellent prospects in several examples of application.

7.2 Outlook

The BMKfr introduced in this thesis offers a number of potential areas for future research:

- We will continue to improve the implementation of the BMKfr, which can be accessed through the BMKwi, by including more features beneficial for the handling of modules and the modular composition of models.
- We will insistently work on the release of more modules in the BMKfr to capture a plethora of metabolic, gene regulatory and signalling pathways in prokaryotes and eukaryotes.
- We will work on additional algorithms to support the reverse engineering of modules and integration of those modules by the use of the BMKwi.
- The naming of nodes in the Petri net graphs of modules is of significant importance for the correct matching of interface network,

and thus of the modular composition of models. We will work on a model ontology to provide a strict naming convention for nodes to ensure the correct composition of models. This might also include the development of a plug-in for the modelling tool Snoopy [120] to reuse content stored in the BMKDB while the data-driven construction of modules.

- We will also work on the integration of patient specific data in the modular model composition process to offer an *in silico* solution for personalised medicine. Patient data will be integrated to individualise modularly composed models by either specifying the set of modules to include in a composed model, to initialize the marking of the composed model, or to design module versions of altered components or molecular processes affected in a patient.

Bibliography

[1] A. Fick. "On liquid diffusion". In: *Philosophical Magazine* 10.63 (1855), pp. 30–39.

[2] W. V. Quine. "The Problem of Simplifying Truth Functions". In: *The American Mathematical Monthly* 59.8 (1952), p. 521.

[3] E. J. McCluskey. "Minimization of Boolean Functions". In: *Bell System Technical Journal* 35.6 (1956), pp. 1417–1444.

[4] C. A. Petri. "Kommunikation mit Automaten". PhD Thesis. Darmstadt University of Technology, Germany, 1962.

[5] S. A. Kauffman. "Metabolic stability and epigenesis in randomly constructed genetic nets". In: *Journal of Theoretical Biology* 22.3 (1969), pp. 437–467.

[6] E. F. Codd. "A Relational Model of Data for Large Shared Data Banks." In: *Communications of ACM* 13.6 (1970), pp. 377–387.

[7] M. J. Flynn and T. Agerwala. "Comments on Capabilities, Limitations and Correctness of Petri Nets." In: *Proc. 1st Annual Symposium on Computer Architecture (ISCA 1973)* 2.4 (1973), pp. 81–86.

[8] R. Thomas. "Boolean formalization of genetic control circuits". In: *Journal of Theoretical Biology* 42.3 (1973), pp. 563–585.

[9] T. Araki and T. Kasami. "Some Decision Problems Related to the Reachability Problem for Petri Nets." In: *Theoretical Computer Science* 3.1 (1976), pp. 85–104.

[10] A. W. Shogan. "A decomposition algorithm for network reliability analysis." In: *Networks* 8.3 (1978), pp. 231–251.

[11] H. J. Genrich and K. Lautenbach. "The analysis of distributed systems by means of predicate/ transition-nets". In: *The Temporal Semantics of Concurrent Programs.* Berlin/Heidelberg: Springer Berlin Heidelberg, 1979, pp. 123–146.

[12] H. J. Genrich and K. Lautenbach. "System Modelling with High-Level Petri Nets." In: *Theoretical Computer Science* 13.1 (1981), pp. 109–136.

[13] K. Jensen. "Coloured Petri Nets and the Invariant-Method." In: *Theoretical Computer Science* 14.3 (1981), pp. 317–336.

[14] A. Pnueli. "The temporal semantics of concurrent programs". In: *Theoretical Computer Science* 13.1 (Jan. 1981), pp. 45–60.

[15] L. Cardelli and P. Wegner. "On Understanding Types, Data Abstraction, and Polymorphism." In: *ACM Computing Surveys* 17.4 (1985), pp. 471–522.

[16] E. M. Clarke, E. A. Emerson, and A. P. Sistla. "Automatic verification of finite-state concurrent systems using temporal logic specifications". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 8.2 (1986), pp. 244–263.

[17] J Otomo, W Marwan, D Oesterhelt, et al. "Biosynthesis of the two halobacterial light sensors P480 and sensory rhodopsin and variation in gain of their signal transduction chains." In: *Journal of Bacteriology* (1989).

[18] F. C. Neidhardt, J. L. Ingraham, and M Schaechter. *Physiology of the Bacterial Cell. A Molecular Approach*. Sunderland, Massachusetts: Sinauer Associates, 1990.

[19] V. N. Reddy, M. L. Mavrovouniotis, and M. N. Liebman. "Petri net representations in metabolic pathways." In: *Proc. International Conference on Intelligent Systems for Molecular Biology (ISMB 1993)* 1 (1993), pp. 328–336.

[20] M. AB. *MySQL*. 1994. URL: `http://www.mysql.com`.

[21] U. Montanari and F. Rossi. "Contextual nets". In: *Acta Informatica* 32.6 (1995), pp. 545–596.

[22] W3C. *XML*. 1996. URL: `http://www.w3.org/TR/REC-xml/`.

[23] T. Akutsu, S. Miyano, and S. Kuhara. "Identification of Genetic Networks from a Small Number of Gene Expression Patterns Under the Boolean Network Model." In: *Pacific Symposium on Biocomputing* (1999), pp. 17–28.

[24] P. Buchholz and P. Kemper. "A Toolbox for the Analysis of Discrete Event Dynamic Systems." In: *CAV* (1999), pp. 483–486.

[25] D. E. Levy. "Physiological significance of STAT proteins: Investigations through gene disruption in vivo." In: *Cellular and Molecular Life Sciences* 55.12 (1999), pp. 1559–1567.

[26] A. Aziz, K. Sanwal, V. Singhal, et al. "Model-checking continuous-time Markov chains". In: *ACM Transactions on Computational Logic (TOCL)* 1.1 (2000), pp. 162–170.

[27] M Kanehisa and S Goto. "KEGG: Kyoto encyclopedia of genes and genomes." In: *Nucleic Acids Research* 28.1 (2000), pp. 27–30.

[28] S Roch and P. H. Starke. *INA Integrated Net Analyzer Version 2.2 Manual*. Humboldt-Universitat zu Berlin, 2000.

[29] J. C. Rowlingson. "Textbook of Pain". In: *Anesthesia & Analgesia* 91.5 (2000), p. 1315.

[30] J Gough, K Karplus, R Hughey, et al. "Assignment of homology to genome sequences using a library of hidden Markov models that represent all proteins of known structure." In: *Journal of Molecular Biology* 313.4 (2001), pp. 903–919.

[31] H. Kitano, ed. *Foundations of systems biology*. Cambridge (Mass.), London: MIT Press, 2001.

[32] R. J. Roberts. "PubMed Central: The GenBank of the published literature." In: *Proc. National Academy of Sciences* 98.2 (2001), pp. 381–382.

[33] C. Gershenson. "Classification of Random Boolean Networks". In: *Proc. 8th International Conference on Artificial life.* (2002).

[34] T. K. Attwood, P Bradley, D. R. Flower, et al. "PRINTS and its automatic supplement, prePRINTS." In: *Nucleic Acids Research* 31.1 (2003), pp. 400–402.

[35] G Ciardo, R. L. Jones, A. S. Miner, et al. "Logical and Stochastic Modeling with Smart". In: *Computer Performance Evaluation. Modelling Techniques and Tools*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 78–97.

[36] C. A. Cooper, H. J. Joshi, M. J. Harrison, et al. "GlycoSuiteDB: A curated relational database of glycoprotein glycan structures and their biological sources. 2003 update." In: *Nucleic Acids Research* 31.1 (2003), pp. 511–513.

[37] A. A. Cuellar, C. M. Lloyd, P. M. F. Nielsen, et al. "An Overview of CellML 1.1, a Biological Model Description Language." In: *SIMULATION* 79.12 (2003), pp. 740–747.

[38] A. Gattiker, K. Michoud, C. Rivoire, et al. "Automated annotation of microbial proteomes in SWISS-PROT." In: *Computational Biology and Chemistry* 27.1 (2003), pp. 49–58.

[39] P. C. Heinrich, I. Behrmann, S. Haan, et al. "Principles of interleukin (IL)-6-type cytokine signalling and its regulation." In: *The Biochemical Journal* 374.Pt 1 (2003), pp. 1–20.

[40] M Hucka, A Finney, H. M. Sauro, et al. "The Systems Biology Markup Language (SBML): A medium for representation and exchange of biochemical network models". In: *Bioinformatics (Oxford, England)* (2003).

[41] D Kamimura, K Ishihara, and T Hirano. "IL-6 signal transduction and its physiological roles: The signal orchestration model." In: *Reviews of Physiology, Biochemistry and Pharmacology* 149.Chapter 1 (2003), pp. 1–38.

[42] M. Migliore, T. M. Morse, A. P. Davison, et al. "ModelDB - Making models publicly accessible to support computational neuroscience." In: *Neuroinformatics* 1.1 (2003), pp. 135–139.

[43] A. V. Ratzer, L. Wells, H. M. Lassen, et al. "CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets." In: *ICATPN* 2679.Chapter 28 (2003), pp. 450–462.

[44] S. Sivakumaran, S. Hariharaputran, J. Mishra, et al. "The Database of Quantitative Cellular Signaling: Management and analysis of chemical kinetic models of signaling networks." In: *Bioinformatics (Oxford, England)* 19.3 (2003), pp. 408–415.

[45] F. Campagne, S. Neves, C.-w. Chang, et al. "Quantitative information management for the biochemical computation of cellular networks." In: *Science's STKE : Signal Transduction Knowledge Environment* 2004.248 (2004), pl11.

[46] C. Chaouiya, E. Remy, P. Ruet, et al. "Applications and Theory of Petri Nets 2004: 25th International Conference, ICATPN 2004, Bologna, Italy, June 21–25, 2004. Proceedings". In: ed. by J. Cortadella and W. Reisig. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. Chap. Qualitative Modelling of Genetic Networks: From Logical Regulatory Graphs to Standard Petri Nets, pp. 137–156.

[47] P. V. Hornbeck, I. Chabra, J. M. Kornhauser, et al. "PhosphoSite: A bioinformatics resource dedicated to physiological protein phosphorylation." In: *Proteomics* 4.6 (2004), pp. 1551–1561.

[48] M. Hucka, A. Finney, B. Bornstein, et al. "Evolving a *lingua franca* and associated software infrastructure for computational systems biology: The Systems Biology Markup Language (SBML) project." In: *Systems Biology* 1.1 (2004), pp. 41–53.

[49] B. G. Olivier and J. L. Snoep. "Web-based kinetic modelling using JWS Online." In: *Bioinformatics (Oxford, England)* 20.13 (2004), pp. 2143–2144.

[50] C. H. Wu, A. Nikolskaya, H. Huang, et al. "PIRSF: Family classification system at the Protein Information Resource." In: *Nucleic Acids Research* 32.Database issue (2004), pp. D112–4.

[51] S. M. Wurgler-Murphy, D. M. King, and P. J. Kennelly. "The Phosphorylation Site Database: A guide to the serine-, threonine-, and/or tyrosine-phosphorylated proteins in prokaryotic organisms." In: *Proteomics* 4.6 (2004), pp. 1562–1570.

[52] C. Bru, E. Courcelle, S. Carrère, et al. "The ProDom database of protein domain families: More emphasis on 3D." In: *Nucleic Acids Research* 33.Database issue (2005), pp. D212–5.

[53] H. Kitano. "International alliances for quantitative modeling in systems biology." In: *Molecular Systems Biology* 1.1 (2005), 2005.0007–E2.

[54] N. Le Novère, A. Finney, M. Hucka, et al. "Minimum information requested in the annotation of biochemical models (MIRIAM)." In: *Nature Biotechnology* 23.12 (2005), pp. 1509–1515.

[55] J. Saez-Rodriguez, A. Kremling, and E. D. Gilles. "Dissecting the puzzle of life: Modularization of signal transduction networks". In: *Computers & Chemical Engineering* 29.3 (2005), pp. 619–629.

[56] M Scherf, A Epple, and T Werner. "The next generation of literature analysis: Integration of genomic analysis into text mining". In: *Briefings in Bioinformatics* 6 (2005), pp. 287–297.

[57] B Berthomieu and F Vernadat. *Time Petri Nets Analysis with TINA*. IEEE, 2006.

[58] L. Cloth and B. R. Haverkort. "Five performability algorithms. A comparison". In: (2006).

[59] R. D. Finn, J. Mistry, B. Schuster-Böckler, et al. "Pfam: Clans, web tools and services." In: *Nucleic Acids Research* 34.Database issue (2006), pp. D247–51.

[60] A. Hinton, M. Kwiatkowska, G. Norman, et al. "PRISM: A Tool for Automatic Verification of Probabilistic Systems". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 441–444.

[61] N. Hulo, A. Bairoch, V. Bulliard, et al. "The PROSITE database." In: *Nucleic Acids Research* 34.Database issue (2006), pp. D227–30.

[62] N. Le Novère, B. Bornstein, A. Broicher, et al. "BioModels Database: A free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems." In: *Nucleic Acids Research* 34.Database issue (2006), pp. D689–91.

[63] I. Letunic, R. R. Copley, B. Pils, et al. "SMART 5: Domains in the context of genomes and networks." In: *Nucleic Acids Research* 34.Database issue (2006), pp. D257–60.

[64] J. Moffat and D. M. Sabatini. "Building mammalian signalling pathways with RNAi screens." In: *Nature Reviews Molecular Cell Biology* 7.3 (2006), pp. 177–187.

[65] C. Yeats, M. Maibaum, R. Marsden, et al. "Gene3D: Modelling protein structure, function and evolution." In: *Nucleic Acids Research* 34.Database issue (2006), pp. D281–4.

[66] Y Arinobu, S Mizuno, Y Chong, et al. "Reciprocal activation of GATA-1 and PU.1 marks initial specification of hematopoietic stem cells into myeloerythroid and myelolymphoid lineages". In: *Cell - Stem Cell* 1 (2007), pp. 416–427.

[67] D. Gilbert, M. Heiner, and S. Lehrack. "A Unifying Framework for Modelling and Analysing Biochemical Pathways Using Petri Nets". In: *Computational Methods in Systems Biology*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 200–216.

[68] T. Hucho and J. D. Levine. "Signaling Pathways in Sensitization: Toward a Nociceptor Cell Biology". In: *Neuron* 55.3 (2007), pp. 365–376.

[69] H. Iwasaki and K. Akashi. "Myeloid lineage commitment from the hematopoietic stem cell." In: *Immunity* 26.6 (2007), pp. 726–740.

[70] K. Jensen, L. M. Kristensen, and L. Wells. "Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems." In: *International Journal on Software Tools for Technology Transfer* 9.3-4 (2007), pp. 213–254.

[71] H. Mi, N. Guo, A. Kejariwal, et al. "PANTHER version 6: Protein sequence and function evolution data with expanded representation of biological pathways." In: *Nucleic Acids Research* 35.Database issue (2007), pp. D247–52.

[72] N. J. Mulder, R. Apweiler, T. K. Attwood, et al. "New developments in the InterPro database." In: *Nucleic Acids Research* 35.Database issue (2007), pp. D224–8.

[73] J. D. Selengut, D. H. Haft, T. Davidsen, et al. "TIGRFAMs and Genome Properties: Tools for the assignment of molecular function and biological process in prokaryotic genomes." In: *Nucleic Acids Research* 35.Database issue (2007), pp. D260–4.

[74] L. J. Steggles, R. Banks, O. Shaw, et al. "Qualitatively modelling and analysing genetic regulatory networks: A Petri net approach." In: *Bioinformatics (Oxford, England)* 23.3 (2007), pp. 336–343.

[75] E. E. Bolton, Y. Wang, P. A. Thiessen, et al. "PubChem: Integrated Platform of Small Molecules and Biological Activities". In: Elsevier, 2008, pp. 217–241.

[76] C. Chaouiya, E. Remy, and D. Thieffry. "Petri net modelling of biological regulatory networks". In: *Journal of Discrete Algorithms* 6.2 (2008), pp. 165–177.

[77] R. Donaldson and D. Gilbert. *A Monte Carlo model checker for probabilistic LTL with numerical constraints*. Tech. rep. University of Glasgow, Department of Computer Science, 2008.

[78] P Flicek, B. L. Aken, K Beal, et al. "Ensembl 2008." In: *Nucleic Acids Research* 36.Database issue (2008), pp. D707–14.

[79] Gene Ontology Consortium. "The Gene Ontology project in 2008." In: *Nucleic Acids Research* 36.Database issue (2008), pp. D440–4.

[80] E. Grafahrend-Belau, F. Schreiber, M. Heiner, et al. "Modularization of biochemical networks based on classification of Petri net t-invariants." In: *BMC Bioinformatics* 9 (2008), p. 90.

[81] P Grossman. *Discrete mathematics for computing*. 2008.

[82] M Heiner, D Gilbert, and R Donaldson. "Petri Nets for Systems and Synthetic Biology". In: *SFM 2008*. Ed. by M Bernardo, P Degano, and G Zavattaro. Vol. 5016. LNCS. Springer, 2008, pp. 215–264.

[83] G. Karlebach and R. Shamir. "Modelling and analysis of gene regulatory networks". In: *Nature Reviews Molecular Cell Biology* 9.10 (2008), pp. 770–780.

[84] C. M. Lloyd, J. R. Lawson, P. J. Hunter, et al. "The CellML Model Repository." In: *Bioinformatics (Oxford, England)* 24.18 (2008), pp. 2122–2123.

[85] S. T. Chou, E Khandros, L. C. Bailey, et al. "Graded repression of PU.1/Sfpi1 gene transcription by GATA factors regulates hematopoietic cell fate". In: *Blood* 114 (2009), pp. 983–994.

[86] T Courtney, S Gaonkar, K Keefe, et al. "Möbius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models". In: *Networks (DSN)* (2009), pp. 353–358.

[87] A. P. Heath and L. E. Kavraki. "Computational challenges in systems biology." In: *Computer Science Review* 3.1 (2009), pp. 1–17.

[88] M. Hecker, S. Lambeck, S. Toepfer, et al. "Gene regulatory network inference: Data integration in dynamic models-a review." In: *Bio Systems* 96.1 (2009), pp. 86–103.

[89] M. Heiner, S. Lehrack, D. R. Gilbert, et al. "Extended Stochastic Petri Nets for Model-Based Design of Wetlab Experiments." In: *Transactions on Computational Systems Biology* 5750.Chapter 7 (2009), pp. 138–163.

[90] L. M. Hillah, E Kindler, F Kordon, et al. "A primer on the Petri Net Markup Language and ISO/IEC 15909–2". In: *Petri Net Newsletter* (2009).

[91] K. Jensen and L. M. Kristensen. *Coloured Petri nets: Modelling and validation of concurrent systems*. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2009.

[92] R. Randhawa, C. A. Shaffer, and J. J. Tyson. "Model aggregation: A building-block approach to creating large macromolecular regulatory networks". In: *Bioinformatics (Oxford, England)* 25.24 (2009), pp. 3289–3295.

[93] M. A. Blätke. "Petri-Netz Modellierung mittels eines modularen und hierarchischen Ansatzes mit Anwendung auf nozizeptive Signalkomponenten". Diploma Thesis. Otto–von–Guericke–University Magdeburg, 2010.

[94] M. A. Blätke and W. Marwan. "Modular and hierarchical modelling concept for large biological Petri nets applied to nociception". In: *Proc. 17th German Workshop on Algorithms and Tools for Petri Nets (AWPN 2010)*. Vol. 643. CEUR Workshop Proceedings. CEUR-WS.org, 2010, pp. 42–50.

[95] M. A. Blätke, S. Meyer, C. Stein, et al. “Petri net modeling via a modular and hierarchical approach applied to nociception”. In: *Proc. 1st Int. Workshop on Biological Processes & Petri Nets (BioPPN), satellite event of Petri Nets 2010*. Braga, Portugal, 2010, pp. 135–146.

[96] N.-Y. Chia, Y.-S. Chan, B. Feng, et al. “A genome-wide RNAi screen reveals determinants of human embryonic stem cell identity.” In: *Nature* 468.7321 (2010), pp. 316–320.

[97] M. T. Cooling, V Rouilly, G. Misirli, et al. “Standard virtual biological parts: A repository of modular modeling components for synthetic biology.” In: *Bioinformatics (Oxford, England)* 26.7 (2010), pp. 925–931.

[98] A. Cuomo and T. Bonaldi. “Systems Biology “On-the-Fly”: SILAC-Based Quantitative Proteomics and RNAi Approach in Drosophila melanogaster”. In: *Systems Biology in Drug Discovery and Development*. Totowa, NJ: Humana Press, 2010, pp. 59–78.

[99] M. Heiner, C. Mahulea, and M. Silva. “On the Importance of the Deadlock Trap Property for Monotonic Liveness.” In: *ACSD/Petri Nets Workshops* (2010), pp. 23–38.

[100] Y.-J. Hsieh and B. L. Wanner. “Global regulation by the seven-component Pi signaling system.” In: *Current Opinion in Microbiology* 13.2 (2010), pp. 198–203.

[101] C. Li, M. Donizelli, N. Rodriguez, et al. “BioModels Database: An enhanced, curated and annotated resource for published quantitative kinetic models”. In: *BMC Systems Biology* 4.1 (2010), p. 1.

[102] M. Nagasaki, A. Saito, E. Jeong, et al. “Cell Illustrator 4.0: A Computational Platform for Systems Biology”. In: *In Silico Biology* 10.1,2 (2010), pp. 5–26.

[103] R. Randhawa, C. A. Shaffer, and J. J. Tyson. “Model composition for macromolecular regulatory networks.” In: *IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM* 7.2 (2010), pp. 278–287.

[104] C. Rohr, W. Marwan, and M. Heiner. “Snoopy – A unifying Petri net framework to investigate biomolecular networks.” In: *Bioinformatics (Oxford, England)* 26.7 (2010), pp. 974–975.

[105] M. A. Blätke, M. Heiner, and W. Marwan. *Tutorial – Petri nets in systems biology*. Tech. rep. Otto–von–Guericke–University Magdeburg, 2011.

[106] M. A. Blätke, S. Meyer, and W. Marwan. “Pain signaling – A case study of the modular Petri net modeling concept with prospect to a protein-oriented modeling platform”. In: *Proc. 2nd International Workshop on Biological Processes & Petri Nets (BioPPN), satellite event of Petri NETS 2011*. Vol. 724. CEUR Workshop Proceedings. CEUR-WS.org, 2011, pp. 117–134.

[107] H. Bongartz. "Experimentelle Parameterisierung und Validierung eines Petri-Netz Modells der IL-6-induzierten JAK/STAT-Signaltransduktion". Masterthesis. Otto–von–Guericke–University Magdeburg, 2011.

[108] M Durzinsky, W Marwan, M. Ostrowski, et al. "Automatic network reconstruction using ASP". In: *Theory and Practice of Logic Programming* 11 (2011), pp. 749–766.

[109] M. Durzinsky, A. Wagler, and W. Marwan. "Reconstruction of extended Petri nets from time series data and its application to signal transduction and to gene regulatory networks". In: *BMC Systems Biology* 5.1 (2011), p. 113.

[110] S. Ghosh, Y. Matsuoka, Y. Asai, et al. "Software for systems biology: From tools to integrated platforms". In: *Nature Reviews. Genetics* (2011).

[111] M Heiner and D Gilbert. "How Might Petri Nets Enhance Your Systems Biology Toolkit". In: *Proc. PETRI NETS 2011*. Ed. by L. Kristensen and L Petrucci. Vol. 6709. Lecture Notes in Computer Science. Springer, 2011, pp. 17–37.

[112] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, et al. "The ins and outs of the probabilistic model checker MRMC". In: *Performance Evaluation* 68.2 (2011), pp. 90–104.

[113] J. Krumsiek, C. Marr, T. Schroeder, et al. "Hierarchical Differentiation of Myeloid Progenitors Is Encoded in the Transcription Factor Network". In: *PloS one* 6.8 (2011), e22649.

[114] W. Marwan, C. Rohr, and M. Heiner. "Petri Nets in Snoopy: A Unifying Framework for the Graphical Display, Computational Modelling, and Simulation of Bacterial Regulatory Networks". In: *Systems Biology in Drug Discovery and Development*. New York, NY: Springer New York, 2011, pp. 409–437.

[115] B. Schwanhäusser, D. Busse, N. Li, et al. "Global quantification of mammalian gene expression control." In: *Nature* 473.7347 (2011), pp. 337–342.

[116] M. A. Blätke, M. Heiner, and W. Marwan. "Predicting Phenotype from Genotype Through Automatically Composed Petri Nets". In: *Proc. 10th International Conference on Computational Methods in Systems Biology (CMSB 2012), London*. Vol. 7605. LNCS/LNBI. Springer, 2012, pp. 87–106.

[117] M. A. Blätke and W. Marwan. "A database-supported modular modelling platform for systems and synthetic biology". In: *Proc. 3rd International Workshop on Biological Processes & Petri Nets (BioPPN), satellite event of Petri NETS 2012*. Vol. 852. CEUR Workshop Proceedings. CEUR-WS.org, 2012, pp. 18–19.

[118] M. A. Blätke, A. Dittrich, C. Rohr, et al. "JAK–STAT Signalling as Example for a Database-Supported Modular Modelling Concept". In: *Proc. 10th International Conference on Computational Methods in Systems Biology (CMSB 2012), London*. Vol. 7605. LNCS/LNBI. Springer, 2012, pp. 362–365.

[119] A. Dittrich, T. Quaiser, C. Khouri, et al. "Model-driven experimental analysis of the function of SHP-2 in IL-6-induced JAK/-STAT signaling". In: *Molecular BioSystems* 8.8 (2012), pp. 2119–2134.

[120] M Heiner, M Herajy, F Liu, et al. "Snoopy – A unifying Petri net tool". In: *Proc. PETRI NETS 2012*. Vol. 7347. LNCS. Hamburg: Springer, 2012, pp. 398–407.

[121] F Liu, M Heiner, and C Rohr. *The Manual for Colored Petri Nets in Snoopy – QPN^C/SPN^C/CPN^C/$GHPN^C$*. Tech. rep. 02-12. Brandenburg University of Technology Cottbus, Department of Computer Science, 2012.

[122] F. Liu. "Colored Petri Nets for Systems Biology". PhD Thesis. Brandenburg University of Technology, 2012.

[123] D Maccagnola, Q Gao, E Messina, et al. "A Machine Learning Approach for Generating Temporal Logic Classifications of Complex Model Behaviours". In: *Proc. Winter Simulation Conference (WSC 2012), Berlin*. IEEE, 2012.

[124] W. Marwan and M. A. Blätke. "A module-based approach to biomodel engineering with Petri nets". In: *Proc. Winter Simulation Conference (WSC 2012), Berlin*. IEEE, 2012, pp. 3383–3394.

[125] C.-F. Tiger, F. Krause, G. Cedersund, et al. "A framework for mapping, visualisation and automatic model creation of signal-transduction networks." In: *Molecular Systems Biology* 8 (2012), p. 578.

[126] M. A. Blätke, A. Dittrich, C. Rohr, et al. "JAK/STAT signalling - an executable model assembled from molecule-centred modules demonstrating a module-oriented database concept for systems and synthetic biology". In: *Molecular BioSystem* 9.6 (2013), pp. 1290–1307.

[127] Q Gao, D Gilbert, M Heiner, et al. "Multiscale Modelling and Analysis of Planar Cell Polarity in the Drosophila Wing". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 10.2 (2013), pp. 337–351.

[128] D Gilbert, M Heiner, F Liu, et al. "Colouring Space - A Coloured Framework for Spatial Modelling in Systems Biology". In: *Proc. PETRI NETS 2013*. Ed. by J. Colom and J Desel. Vol. 7927. LNCS. Milano: Springer, 2013, pp. 230–249.

[129] M Heiner, C Rohr, and M Schwarick. "MARCIE – Model checking And Reachability analysis done effiCIEntly". In: *Proc. PETRI NETS 2013*. Ed. by J. Colom and J Desel. Vol. 7927. LNCS. Milano: Springer, 2013, pp. 389–399.

[130] M. Herajy. "Computational Steering of Multi-Scale Biochemical Networks". PhD Thesis. BTU Cottbus, Dep. of CS, 2013.

[131] C. F. Lopez, J. L. Muhlich, J. A. Bachman, et al. "Programming biological models in Python using PySB." In: *Molecular Systems Biology* 9.1 (2013), pp. 646–646.

[132] O Parvu, D Gilbert, M Heiner, et al. "Modelling and Analysis of Phase Variation in Bacterial Colony Growth". In: *Proc. 11th International Conference on Computational Methods in Systems Biology (CMSB 2013),Vienna*. Ed. by A. Gupta and T. Henzinger. Vol. 8130. LNCS/LNBI. Springer, 2013, pp. 78–91.

[133] M. A. Blätke, C. Rohr, M. Heiner, et al. "A Petri-Net-Based Framework for Biomodel Engineering". In: *Large-Scale Networks in Engineering and Life Sciences*. Modeling and Simulation in Science, Engineering and Technology. Springer International Publishing, 2014, pp. 317–366.

[134] L. Jehrke. "Modulare Modellierung und graphische Darstellung boolescher Netzwerke mit Hilfe automatisch erzeugter Petri-Netze und ihre Simulation am Beispiel eines genregulatorischen Netzwerkes". Masterthesis. Otto–von–Guericke–University Magdeburg, 2014.

[135] F. Liu, M. A. Blätke, M. Heiner, et al. "Modelling and simulating reaction–diffusion systems using coloured Petri nets". In: *Computers in Biology and Medicine* 53 (2014), pp. 297–308.

[136] M. Soldmann. "Transformation monolithischer SBML-Modelle biomolekularer Netzwerke in Petri Netz Module". Masterthesis. Otto–von–Guericke–University Magdeburg, 2014.

[137] M. Beccuti, M. A. Blätke, M. Falk, et al. "*Dictyostelium discoideum*: Aggregation and Synchronisation of Amoebas in Time and Space". In: vol. 4. 11. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015, pp. 138–226.

[138] M. A. Blätke, M. Heiner, and W. Marwan. "BioModel Engineering with Petri Nets". In: *Algebraic and Discrete Mathematical Methods for Modern Biology*. Elsevier Inc., 2015. Chap. 7, pp. 141–193.

[139] M. A. Blätke and C. Rohr. "A coloured Petri net approach for spatial Biomodel Engineering based on the modular model composition framework Biomodelkit". In: *Proc. 6th Int. Workshop on Biological Processes & Petri Nets (BioPPN 2015), satellite event of Petri Nets 2015*. Vol. 1373. CEUR Workshop Proceedings. CEUR-WS.org, 2015, pp. 37–54.

[140] K. A. Gray, B. Yates, R. L. Seal, et al. "Genenames.org: The HGNC resources in 2015". In: *Nucleic Acids Research* 43.D1 (2015), pp. D1079–D1085.

[141] M Heiner, M Schwarick, and J Wegener. "Charlie – An extensible Petri net analysis tool". In: *Proc. PETRI NETS 2015*. Ed. by R Devillers and A Valmari. Vol. 9115. LNCS. Brussels: Springer, 2015, pp. 200–211.

[142] M. Hucka, F. T. Bergmann, S. Hoops, et al. "The Systems Biology Markup Languagee (SBML): Language Specification for Level 3 Version 1 Core." In: *Journal of Integrative Bioinformatics* 12.2 (2015), p. 266.

[143] UniProt Consortium. "UniProt: A hub for protein information." In: *Nucleic Acids Research* 43.Database issue (2015), pp. D204–12.

[144] *GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets 2.0 (GreatSPN 2.0)*. URL: `http://http://www.di.unito.it/~greatspn/`.

Curriculum Vitae

Personal Information

Day of Birth	14.12.1985	
Place of Birth	Schönebeck/Elbe, Germany	
Nationality	German	
Personal Status	unmarried	
Adress	business: Otto-von-Guericke University Chair of Regulatory Biology (FNW/IBIO) Pfälzerstraße 5 (Building 25) 39106 Magdeburg/Germany	private: Karl-Marx-Straße 50 39221 Kleinmühlingen/Germany
Phone	+49 391 6758555	+49 160 96796909
Email	mary-ann.blaetke@ovgu.de	maryann.blaetke@gmail.com

Academic Career

seit 2011	Doctorate in Biology/Neurobiology, Otto-von-Guericke University, Magdeburg, Germany Member of the *International Max Planck Research School*, Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany
2005 - 2010	Study of biosystems engineering, Otto-von-Guericke University, Magdeburg
2009	Student resaerch thesis: "Petri Net Modules of Integrative Nociceptive Neurons" (Supervisor: Prof. Dr. Wolfgang Marwan)
2010	Diploma thesis: "Petri-Netz Modellierung mittels eines modularen und hierarchischen Ansatzes mit Anwendung auf nozizeptive Signalkomponenten" (Supervisor: Prof. Dr. Wolfgang Marwan) Graduate engineer for biosystems engineering
2005	Abitur, Friedrich Schiller Gymnasium, Calbe/Saale, Germany

Employment History

10/2010 - 02/2016	Research assistant at the chair of regulatory biology, Otto-von-Guericke University, Magdeburg, Germany
12/2009 - 09/2010	Student research assistant at the chair of regulatory biology, Otto-von-Guericke University, Magdeburg, Germany

10/2009 - 01/2010	Industrial internship, Bayer Technology Services GmbH, Leverkusen, Germany
08/2007 - 09/2007	Industrial internship, Lagotec GmbH, Magdeburg, Germany
02/2007 - 04/2007	Scientific internship, Helmholtz-Centre for Environmental Research, Magdeburg, Germany
07/2005 - 08/2005	Scientific internship, Otto-von-Guericke University, Magdeburg, Germany

Additional Qualifications and Certificates

2012	Presenting Professional, *International Max Planck Research School*, Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany
2012	Scientific Writing, *International Max Planck Research School*, Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany
2012	Teaching, *International Max Planck Research School*, Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany
2007	Basics of Laser Scanning Microscopy - Theory and Practice, Helmholtz-Centre for Environmental Research, Magdeburg, Germany
2005	Integration of women in scientific and engineering-oriented projects, Otto-von-Guericke University, Magdeburg, Germany

Teaching Experience

2012 – 2015 (SS)	Lecture and exercise "Biomodelltechnik mit Petri Netzen", Masterstudiengang Biosystemtechnik, Otto-von-Guericke University, Magdeburg, Germany
2010 – 2015	Practical Course "Modellierung des *Lac* Operons mithilfe von Petri Netzen", Otto-von-Guericke University, Magdeburg, Germany
2013 (SS)	Lecture and exercise "Biomodel Engineering with Petri Nets Applied to Systems Biology", *International Max Planck Research School*, Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany

Supervised Thesis

2014	M. Soldmann: Transformation monolithischer SBML-Modelle biomolekularer Netzwerke in Petri Netz Module, Otto-von-Guericke University, Magdeburg, Germany (master thesis)
2014	L. Jehrke: Modulare Modellierung und graphische Darstellung boolescher Netzwerke mit Hilfe automatisch erzeugter Petri-Netze und ihre Simulation am Beispiel eines genregulatorischen Netzwerkes, Otto-von-Guericke University, Magdeburg, Germany (master thesis)
2011	H. Bongartz: Experimentelle Parameterisierung und Validierung eines Petri-Netz Modells der IL-6-induzierten JAK/STAT- Signaltransduktion, Otto-von-Guericke University, Magdeburg, Germany (master thesis)

Academic Work

2012 – 2015	Consortium "Nociceptor Pain Model" (NoPain), (directed by Prof. Dr. Tim Hucho, e:Bio funding project of the Federal Ministry of Education and Research, Germany)
2009 – 2011	Consortium "Modelling of Pain Switches" (directed by Prof. Dr. Christoph Stein, MedSys funding project of the Federal Ministry of Education and Research, Germany)

Organisational work

I have been involved in the organisation and execution of the following workshops, tutorials and events:

2015 Presentation of the chair of regulatory biology at "Langen Nacht der Wissenschaft", Otto-von-Guericke University, Magdeburg, Germany

2014 Presentation of the chair of regulatory biology at "Langen Nacht der Wissenschaft", Otto-von-Guericke University, Magdeburg, Germany

2013 Tutorial "Petri Nets for Multiscale Systems Biology" at *Petri Nets 2013* (International Conference on Application and Theory of Petri Nets and Concurrency), Milano, Italy

2012 Workshop "Model Formulation", *International Max Planck Research School*, Max-Planck Institute für Dynamik Komplexer Technischer Systeme, Magdeburg, Germany

2011 Tutorial "Further Advanced Petri Net Techniques and Applications" auf der *ICSB* (International Conference on Systems Biology), Heidelberg, Germany

2011 Tutorial "Foundations of Advanced Petri Nets" auf der *ICSB* (International Conference on Systems Biology), Heidelberg, Germany

2010 Indian Summer School "Petri Net Modelling in Systems Biology", Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany