

Standards and Tools
for Model Exchange and Analysis
in Systems Biology

Ralph Gauges

Dissertation
submitted to the
Combined Faculties for the Natural Sciences and for Mathematics
of the Ruperto-Carola University of Heidelberg, Germany
for the degree of
Doctor of Natural Sciences

presented by

Diplom-Biochemiker Ralph Gauges

born in: Sigmaringen, Germany

Oral-examination: 07/11/2011

**Standards and Tools
for Model Exchange and Analysis
in Systems Biology**

Referees: Prof. Dr. Ursula Kummer

Dr. Rebecca Wade

Contents

Zusammenfassung	vii
Summary	x
Abbreviations	xvii
1 Introduction	1
2 Materials & Methods	19
2.1 Operating Systems	19
2.2 Programming Languages	20
2.3 Unit Testing	24
2.4 Debugging & Profiling Tools	26
2.5 Libraries & Standards	29
3 SBML Layout & Render Extension	46
3.1 SBML & Diagrams	46
3.2 Alternative Diagram Formats	47
3.3 Design & History	49
3.4 The SBML Layout Extension Specification	51
3.5 Implementation Of The Layout Extension	57
3.6 The SBML Render Extension	61
3.7 Third Party Implementations	86
3.8 The SBML Layout And Render Extension In NF- κ B Modeling	87
4 Standards In COPASI	93
4.1 SBML Support In COPASI	93
4.2 Layout And Render Information In COPASI	112
4.3 Graphical Display Of Time Course Simulation Data	115
4.4 Graphical Display Of Elementary Modes	116
4.5 COPASI Language Bindings	117
4.6 NF- κ B Modeling with COPASI	120

4.7	Work Contributions	128
5	Expression Normalization	130
5.1	Normal Form Classes	131
5.2	Expression Tree Classes	137
5.3	Normalization Algorithm	142
5.4	Testing	150
5.5	Normalizing BioModels Expressions	150
5.6	Finding Kinetic Laws For The NF- κ B Model	153
6	Discussion	157
6.1	Layout And Render Information In SBML	157
6.2	Standards In COPASI	160
6.3	Expression Normalization	164
	Acknowledgments	171

List of Figures

1.1	example of a simple reaction network	6
1.2	alternative model specifications	8
1.3	reaction display and differential equation display in COPASI .	10
1.4	electrical circuit diagram vs. SBGN diagram	14
2.1	XML structure example	33
2.2	Resource Description Framework example	35
2.3	SBML Document structure	39
2.4	Scalable Vector Graphics example	40
2.5	SVG rendering example	41
3.1	diagram examples	49
3.2	SBML XML structure example	52
3.3	model structure vs. layout structure	53
3.4	example of role attribute usage	54
3.5	bounding box example	55
3.6	line segment example	55
3.7	cubic bézier example	55
3.8	curve definition example	56
3.9	layout rendered with different styles	57
3.10	SBML layout classes	59
3.11	embedding render information in SBML documents	63
3.12	RGB and RGBA value specification	64
3.13	color definition example	65
3.14	linear gradient example	65
3.15	radial gradient example	66
3.16	example for association by type	67
3.17	example for association by role	68
3.18	example for association by id	69
3.19	render primitives	71
3.20	render curve example	71

3.21	stroke example	72
3.22	line ending example	73
3.23	application of line ending example	74
3.24	example for <i>enableRotationalMapping</i> attribute	75
3.25	polygon example	76
3.26	fill style example	76
3.27	rectangle example	77
3.28	rectangle with rounded corners	77
3.29	ellipse example	77
3.30	circle example	78
3.31	text element example	78
3.32	bitmap example	79
3.33	complex style example	80
3.34	coordinate schema	81
3.35	relative vs. absolute coordinates	81
3.36	transformation matrix example	82
3.37	render extension classes	83
3.38	render extension implementations	85
3.39	CellDesigner diagram for NF- κ B activation model	89
3.40	display of imported CellDesigner layout information	91
4.1	SBML data structure vs. COPASI data structure	98
4.2	SBML test suite results	106
4.3	SBML online test suite results	107
4.4	simulator comparison website	109
4.5	stochastic test suite results	110
4.6	COPASI screen shot for MIRIAM annotation	111
4.7	layout structure in COPASI files	113
4.8	render classes in COPASI	113
4.9	COPASI layout rendering	114
4.10	time course animation	116
4.11	graphical elementary mode display	117
4.12	Results of elementary mode analysis of initial NF- κ B model	123
4.13	diagram of extended NF- κ B model	124
4.14	Results of elementary mode analysis of extended NF- κ B model	125
4.15	time course data plot from COPASI	126
4.16	several animation frames for a time series of the NF- κ B model	127
5.1	structure of normalized expression	132
5.2	normal form data structure classes	133
5.3	<i>CNormalFraction</i> structure	133

5.4	<i>CNormalSum</i> structure	134
5.5	<i>CNormalProduct</i> structure	134
5.6	<i>CNormalItemPower</i> structure	135
5.7	<i>CNormalGeneralPower</i> structure	135
5.8	<i>CNormalChoice</i> structure	136
5.9	<i>CNormalLogical</i> structure	137
5.10	<i>CNormalChoiceLogical</i> structure	137
5.11	expression tree node classes	138
5.12	expression tree example	139
5.13	choice node example	141
5.14	function expansion example	143
5.15	normalization process	144
5.16	operations on numbers	146
5.17	example for the canceling operation	146
5.18	conversion of <i>CNormalGeneralPower</i>	148
5.19	result of BioModels analysis	152
6.1	ambiguous rate law example	166
6.2	comparison result output	167
6.3	SBO term SBO:0000054 (screen shot)	169

List of Tables

1.1	Definition of "Omics"-terms	4
2.1	SWIG language support	44
3.1	role attribute values	54
3.2	predefined glyph types	67
4.1	supported SBML versions	95
4.2	language binding versions	119
5.1	types for constant nodes	139
5.2	functions represented by function nodes	140
5.3	logical node types	141

Zusammenfassung

Die Systembiologie hat sich in den letzten Jahren zu einem wichtigen Werkzeug der biochemischen Forschung entwickelt. Systembiologie beschäftigt sich mit der Simulation und Analyse biologischer und biochemischer Vorgänge mit Hilfe von Computern. Diese Vorgänge werden in rechnerischen Modellen beschrieben, wobei verschiedene mathematische Formalismen zum Einsatz kommen. Durch die Simulation solcher Modelle ist es meist möglich, Vorhersagen über das Verhalten der repräsentierten Systeme zu machen und so gezielt Experimente zu planen, um diese Vorhersagen zu bestätigen. Dadurch besteht die Möglichkeit, ein besseres Verständnis über das zu untersuchende System zu erlangen bzw. existierende Hypothesen bezüglich des Systems zu überprüfen. Oft kann auch aufgrund der gezielteren Planung von Experimenten durch den Einsatz systembiologischer Methoden kostbare Zeit im Labor eingespart werden.

Für die Analyse und Simulation solcher Modelle wird geeignete Software benötigt und viele Forschungsgruppen auf der ganzen Welt sowie eine Reihe von Firmen arbeiten an der Entwicklung solcher Programme.

Die meisten dieser Programme widmen sich gezielt einem bestimmten Aspekt der systembiologischen Forschung. So ermöglichen manche Programme die Simulation von Modellen, während andere Programme Forscher bei der Erstellung der mathematischen Repräsentation unterstützen und wieder andere Programme sich auf die Visualisierung der verschiedenen Analyseergebnisse spezialisieren.

Da es vermutlich kein Programm gibt, welches sämtliche Arbeitsschritte unterstützt, die zur Erstellung, Simulation und umfassenden Analyse dieser Modelle nötig sind, sind Systembiologen bei ihrer Arbeit oft auf eine ganze Reihe von Programmen angewiesen. Um sinnvoll mit dieser Vielzahl an Programmen arbeiten zu können ist es notwendig, dass die einzelnen Programme den Austausch von Daten und insbesondere Modellen unterstützen. Dies war zu Beginn dieser Arbeit keinesfalls selbstverständlich, sondern eher die Aus-

nahme als die Regel.

Erst die Entwicklung bestimmter Standards auf dem Gebiet der Systembiologie änderte die Situation grundlegend. Mit Hilfe dieser Standards ist es nun möglich, Daten und Modelle einmal zu erstellen, um sie anschließend in verschiedenen Programmen zu simulieren, zu analysieren oder zur Visualisierung zu verwenden.

Ein grosser Teil dieser Arbeit beschäftigt sich dementsprechend auch mit der Entwicklung, der Erweiterung und der Implementierung solcher Standards in Form verschiedener Computerprogramme. Ein solches Programm, welches Forscher bei der Erstellung und Analyse von Modellen biochemischer Reaktionsnetzwerke unterstützt, ist das von unserer Gruppe mitentwickelte Programm COPASI. Teile dieser Arbeit beschreiben somit auch die Implementierung einiger wichtiger Standards im Rahmen der Entwicklung von COPASI. Besonders die von mir entwickelte Erweiterung der System Biology Markup Language (SBML) um die Möglichkeit grafische Darstellungen von Modellen zusammen mit der mathematischen Beschreibung abzuspeichern bzw. auszutauschen nimmt hierbei eine zentrale Rolle ein. Die vorliegende Arbeit beschreibt sowohl die Entwicklung dieser grafischen Erweiterung als auch die Implementierung des zugrundeliegenden SBML Formats sowie der Erweiterung in Form verschiedener Werkzeuge für die systembiologische Forschung. Eine weitere wichtige Rolle spielt die Verwendung der grafischen Erweiterung zur Visualisierung von Simulations- und Analyseergebnissen und die Erstellung entsprechender Visualisierungswerkzeuge in COPASI.

Der dritte Teil dieser Arbeit beschäftigt sich ferner mit der Analyse mathematischer Ausdrücke wie sie oft im Zusammenhang mit Modellen biochemischer Reaktionsnetzwerke verwendet werden. Die beschriebene Analysemethode dient dazu, mathematische Ausdrücke umzuformen (normalisieren), um sie anschliessend vergleichen und unter Umständen identifizieren zu können. Dies dient unter Anderem dem automatisierten Vergleich von Modellen, um Übereinstimmungen in bzw. Unterschiede zwischen diesen zu finden.

Die Verwendung und der Nutzen der entwickelten Formate, Methoden und Werkzeuge wird dabei anhand eines aktuellen Forschungsprojektes dargelegt. Dieses Projekt ist Teil der "Virtual Liver Network" Initiative und wird in Zusammenarbeit mit einer experimentell arbeitenden Forschungsgruppe hier an der Universität Heidelberg durchgeführt.

Das Ziel der "Virtual Liver Network" Initiative ist es, das Wissen um die verschiedenen Prozesse, welche in der Leber ablaufen, voranzubringen. Dies erfolgt insbesondere im Hinblick auf medizinische Belange und Anwendungen. Ein wichtiger Aspekt dieser Forschung ist die Erstellung von mathematischen Modellen verschiedener Signaltransduktionswege, die eine essentielle Rolle z.B. bei der Regeneration der Leber spielen.

Zwei Signalkaskaden, von denen bereits bekannt ist, dass sie großen Einfluss auf die Prozesse haben, die bei der Leberregeneration ablaufen, sind einerseits der relativ gut untersuchte NF- κ B Signaltransduktionsweg und andererseits der erst vor relativ kurzer Zeit entdeckte Hippo Signaltransduktionsweg. Es gibt ebenfalls starke Hinweise auf Interaktionen zwischen diesen beiden Signalkaskaden, die Details bezüglich dieser Interaktionen sind jedoch bisher nicht aufgeklärt. Ziel dieses Forschungsprojektes ist es, die Mechanismen für diese Interaktionen zu finden und die resultierenden Effekte aufzuklären.

Diese Zusammenarbeit steht erst am Anfang und wir sind momentan dabei, initiale Modelle der beiden Signaltransduktionswege zu erstellen. Dabei haben sich die in dieser Arbeit beschriebenen Methoden und Werkzeuge bereits als sehr hilfreich erwiesen.

Summary

Over the last few years, systems biology has become an important tool in biochemical research. In systems biology biological and biochemical processes are simulated and analyzed with the help of computers and computer software. These processes are described by computational models employing a number of different mathematical formalisms.

By simulating these models, predictions about the represented processes can be made which allow for a more target-oriented planing of experiments. These experiments can in turn be used to confirm those predictions. Due to this, a deeper understanding of the examined processes can be gained and unconfirmed hypothesis concerning the systems may be affirmed.

Often, the possibility for more target-oriented planing of experiments also helps in cutting down on costly experiments in favor of more theoretical approaches.

For the analysis and simulation of those computational models, appropriate computer software is needed and many research groups around the world as well as a number of commercial companies are working on the development of such software.

Most of these programs target only certain aspects of systems biological research. Some programs allow for the simulation of models while others are more focused towards building models and still others are specialized in the analysis and the visualization of data created by conduction simulations.

Although some may come close, there is probably no single program that covers all steps and all methods necessary for creating and simulating models as well as analyzing the ensuing results in an all-encompassing way. Therefore, system biologists usually depend on a number of different programs for their work. To really be able to make proper use of these programs, they have to support the exchange of data and especially of models descriptions. When this work started this was not commonplace, but rather the exception

than the rule.

Only with the development of certain standards in systems biology did this situation change fundamentally.

Using these standards it is now possible to create models and data once and use them for simulation, analysis and visualization in many different computer programs.

A major part of this work describes the development, extension and implementation of such standards in the form of various computer software projects. Our group is co-developing a software tool called COPASI which enables scientists to intuitively create simulate and analyze computational reaction network models. Consequently, part of this work details the implementation of standards related to systems biology in the context of the development of this tool.

Especially the graphical extension to the Systems Biology Markup Language (SBML), one of the major standards in systems biology today, plays a central role. This extension was mainly developed by me and it allows the storage and exchange of diagram information together with the mathematical description of a model within SBML documents. This thesis details the development of this extension as well as its implementation and the implementation of the underlying SBML standard in a number of different computer programs for systems biology.

Since the graphical extension is also important for the visualization of simulation results as well as results of other analysis methods, the development of several such visualization methods in COPASI is described.

The last part of this thesis covers a framework for the analysis of mathematical expressions as they are commonly found in biochemical reaction network models. The purpose of this framework is the normalization of arbitrary mathematical terms in order to be able to compare and identify them. This can for example be used to automatically find similarities as well as differences between models which is an important aspect in model merging or data mining.

The use and the usefulness of the described formats, methods and tools is demonstrated in the context of a research project we are currently involved in. This project is part of the Virtual Liver Network initiative and is carried out in collaboration with an experimental research group here at the University of Heidelberg.

The goal of the Virtual Liver Network initiative is to enlarge knowledge

on the different processes taking place in the liver, especially with respect to medical needs and applications. One important aspect of the research conducted in this initiative is the creation of computational models describing certain signaling pathways which are known to play an essential role in e.g. liver regeneration.

Two signaling pathways that are known to have a strong influence on the complex processes taking place during liver regeneration are the relatively well studied NF- κ B signaling pathway as well as the recently discovered Hippo signaling pathway. There is also strong evidence that these two pathways interact thereby influencing each other. The details of these interactions and the points of interaction between the two pathways are still unknown. It is the goal of this collaboration to find these interaction points as well as determine the effects of the interaction.

Work on this project has just recently begun and we are currently in the process of setting up initial models for the two signal transduction pathways. However, the methods and tools described in this thesis have already been very helpful for this work.

List of Papers

published papers

Major parts of this work have been covered by the following papers:

COPASI—a COmplex PAtchway SIMulator.

Bioinformatics. 2006 Dec 15;22(24):3067-74.

Hoops S, Sahle S, Gauges R, Lee C, Pahle J, Simus N, Singhal M, Xu L, Mendes P, Kummer U.

Virginia Bioinformatics Institute, Virginia Tech, Washington St. 0477, Blacksburg, VA 24061, USA.

abstract

Here, we present COPASI, a platform-independent and user-friendly biochemical simulator that offers several unique features. We discuss numerical issues with these features; in particular, the criteria to switch between stochastic and deterministic simulation methods, hybrid deterministic-stochastic methods, and the importance of random number generator numerical resolution in stochastic simulation.

Computational modeling of biochemical networks using COPASI.

Methods Mol Biol. 2009;500:17-59.

Mendes P, Hoops S, Sahle S, Gauges R, Dada J, Kummer U.

Manchester Centre for Integrative Systems Biology, University of Manchester, UK.

abstract

Computational modeling and simulation of biochemical networks is at the core of systems biology and this includes many types of analyses that can aid understanding of how these systems work. COPASI is a generic software package for modeling and simulation of biochemical networks which provides many of these analyses in convenient ways that do not require the user to program or to have deep knowledge of the numerical algorithms. Here we provide a description of how these modeling techniques can be applied to biochemical models using COPASI. The focus is both on practical aspects of software usage as well as on the utility of these analyses in aiding biological understanding. Practical examples are described for steady-state and time-course simulations, stoichiometric analyses, parameter scanning, sensitivity analysis (including metabolic control analysis), global optimization, parameter estimation, and stochastic simulation. The examples used are all published models that are available in the BioModels database in SBML format

A model diagram layout extension for SBML.

Bioinformatics. 2006 Aug 1;22(15):1879-85.

Gauges R, Rost U, Sahle S, Wegner K.

Bioinformatics and Computational Biochemistry, EML Research Schloss-Wolfsbrunnen Weg 33, D-69118 Heidelberg, Germany.

`Ralph.Gauges@eml-r.villa-bosch.de`

abstract

Since the knowledge about processes in living cells is increasing, modelling and simulation techniques are used to get new insights into these complex processes. During the last few years, the SBML file format has gained in popularity and support as a means of exchanging model data between the different modelling and simulation tools. In addition to specifying the model as a set of equations, many modern modelling tools allow the user to create and to interact with the model in the form of a reaction graph. Unfortunately, the SBML file format does not provide for the storage of this graph data along with the mathematical description of the model. Therefore, we developed an extension to the SBML file format that makes it possible to store such layout information which describes position and size of objects in the graphical representation.

SYCAMORE—a systems biology computational analysis and modeling research environment.

Bioinformatics. 2008 Jun 15;24(12):1463-4.

Weidemann A, Richter S, Stein M, Sahle S, Gauges R, Gabdoulline R, Surovtsova I, Semmelrock N, Besson B, Rojas I, Wade R, Kummer U.

Scientific Databases and Visualization Group, EML Research, Schloss-Wolfsbrunnengasse 33, 69118 Heidelberg, Germany.

`ursula.kummer@bioquant.uni-heidelberg.de`

abstract

SYCAMORE is a browser-based application that facilitates construction, simulation and analysis of kinetic models in systems biology. Thus, it allows e.g. database supported modelling, basic model checking and the estimation of unknown kinetic parameters based on protein structures. In addition, it offers some guidance in order to allow non-expert users to perform basic computational modelling tasks. SYCAMORE is freely available for academic use at <http://sycamore.eml.org>. Commercial users may acquire a license.

The Systems Biology Graphical Notation

Nat Biotechnol. 2009 Aug;27(8):735-41.

Le Novère N, Hucka M, Mi H, Moodie S, Schreiber F, Sorokin A, Demir E, Wegner K, Aladjem MI, Wimalaratne SM, Bergman FT, Gauges R, Ghazal P, Kawaji H, Li L, Matsuoka Y, Villéger A, Boyd SE, Calzone L, Courtot M, Dogrusoz U, Freeman TC, Funahashi A, Ghosh S, Jouraku A, Kim S, Kolpakov F, Luna A, Sahle S, Schmidt E, Watterson S, Wu G, Goryanin I, Kell DB, Sander C, Sauro H, Snoep JL, Kohn K, Kitano H.

EMBL European Bioinformatics Institute, Hinxton, UK.

`lenov@ebi.ac.uk`

abstract

Circuit diagrams and Unified Modeling Language diagrams are just two examples of standard visual languages that help accelerate work by promoting regularity, removing ambiguity and enabling software tool support for communication of complex information. Ironically, despite having one of the highest ratios of graphical to textual information, biology still lacks standard graphical notations. The recent deluge of biological knowledge makes addressing this deficit a pressing concern. Toward this goal, we present the

Systems Biology Graphical Notation (SBGN), a visual language developed by a community of biochemists, modelers and computer scientists. SBGN consists of three complementary languages: process diagram, entity relationship diagram and activity flow diagram. Together they enable scientists to represent networks of biochemical interactions in a standard, unambiguous way. We believe that SBGN will foster efficient and accurate representation, visualization, storage, exchange and reuse of information on all kinds of biological knowledge, from gene regulation, to metabolism, to cellular signaling.

Abbreviations

AMD Advanced Micro Devices

AMD64 abbreviation for especially Linux operating systems for 64Bit Intel based CPUs, sometimes also refereed to as x86_64

API Application Programming Interface

ATI ATI Technologies Inc.

ATP adenosine triphosphate

awk text processing computer program, the name is made up of the initials of the authors last names: Alfred V. Aho, Peter J. Weinberger und Brian W. Kernighan

BSD Berkeley Software Distribution

CellML Cell Markup Language

ChEBI Chemical Entities of Biological Interest

CLAPACK C version of Linear Algebra PACKage

COPASI Complex Pathway Simulator

CPU Central Processing Unit

CSS Cascading Style Sheet

DBX source level debugger provided by the Sun Developer Tools

DirectX shorthand for a number of game development libraries from Microsoft (Direct3D, DirectDraw, DirectMusic, DirectPlay, DirectSound, etc.)

DKFZ Deutsches Krebsforschungs Zentrum (German Cancer Research Center)

DNA Desoxyribonucleic Acid

DNF disjunct normal form

DOM Document Object Model

FEBS (Journal) Federation of European Biochemical Societies (Journal)

GDB GNU Debugger

Gepasi General Pathway Simulator

Gimp GNU Image Manipulation Program

GNU GNU is a Unix

GOLD Genomes Online Database

GUI Graphical User Interface

HTML Hypertext Markup Language

IDB Intel Debugger

JDB Java Debugger

JNI Java Native Interface

KEGG Kyoto Encyclopedia of Genes and Genomes

JIT Just in time, as in JIT Compiler.

JPEG Joint Photographic Experts Group

JVM Java Virtual Machine

LAPACK Linear Algebra PACKage

LGPL Lesser GNU Public License

Linux Linus' Unix. Operating system kernel developed by Linus Torvalds.

MIASE Minimum Information About a Simulation Experiment

MIB Manchester Interdisciplinary Biocentre. Part of the University of Manchester, U.K.

MIRIAM Minimum Information Required in the Annotation of Models

mRNA messenger ribonucleic acid

ODE ordinary differential equation

OpenGL Open Graphics Library

OSG Open Scene Graph

PDB Python Debugger

PDE partial differential equation

PDF Portable Document Format

PNG Portable Network Graphics

PPC PowerPC, a CPU type developed by IBM

Qt read 'cute'. Cross platform library for the creation of graphical user interfaces.

Qwt Qt Widgets for Technical Applications. Library for 2D data visualisation in Qt.

Qwtplot3D Library modeled after Qwt for 3D data visualisation in Qt.

RDF Resource Description Framework

RELAX NG REgular LAnguage for XML Next Generation

RGB abbreviation for Red-Green-Blue-value

RGBA abbreviation for Red-Green-Blue-Alpha-value

RNA Ribonucleic Acid

rRNA ribosomal ribonucleic acid

SAX Simple API for XML

SBGN Systems Biology Graphical Notation

SBML Systems Biology Markup Language

SBRML Systems Biology Results Markup Language

SBW Systems Biology Workbench

SCons Software construction toolkit

sed stream editor

SED-ML Simulation Experiment Description Markup Language

SVG Scalable Vector Graphics

SWIG Simplified Wrapper and Interface Generator

SYCAMORE systems biology computational analysis and modeling research environment

tRNA transfer ribonucleic acid

VBI Virginia Biotech Institute

VTk Visualization Toolkit

W3 World Wide Web

W3C World Wide Web Consortium

WinDbg Windows Debugger, source level debugger from Microsoft with graphical user interface

x86 abbreviation for a 32Bit CPU family from Intel

XML Extensible Markup Language

XSL Extensible Stylesheet Language Family

XSLT XSL Transformation

Chapter 1

Introduction

Biochemistry – From Wöhler to Venter

Biochemistry and especially molecular biology are relatively new fields of science compared to for example mathematics, physics or even chemistry. The term biochemistry might first have been used as early as 1882 but there is no clear evidence. Today the invention of the term biochemistry is attributed to Carl Neuberg who was the first editor of the journal "Biochemische Zeitschrift" which is known today as FEBS journal[1]. He started using that term in the year 1903.

But this was not the beginning of biochemistry. Before that time biochemistry was known under the term physiological chemistry. Depending on the way one defines biochemistry, the beginnings of this field date back to the breakthrough discoveries of Friedrich Wöhler and Anselme Payen.

In the year 1828 Wöhler discovered by accident that the organic compound urea can be synthesized from inorganic compounds[2, 3, 4]. Something that was believed to be impossible at that time. Back then the general scientific hypothesis was that only living organisms had the potential to create organic compounds. This hypothesis is known under the term vitalism[5, 6] and it was called into question by Wöhler's discovery.

The fact that the reaction discovered by Wöhler is not the reaction that creates urea in living organisms, as discovered by Hans Krebs and coworkers in the early 1930s[7], does not diminish the importance of this breakthrough.

Only a short time later in 1832 Anselme Payen discovered the first enzyme – diastase – in extract from malt. The enzyme is also known under the name amylase and there exist different isoforms; all of these break down long sugar chains into smaller, soluble sugar compounds. This enzyme is also important for humans where it occurs for example in saliva. By breaking down the long

sugar chains from e.g. starch it makes the sugar molecules available for uptake by the cells in the digestive tract.

The birth of the field of molecular biology was brought about quite some time later by the discovery of the structure of desoxyribonucleic acid (DNA) and the role it plays in living organisms. The existence of DNA has been known since 1869, when it was discovered by Friedrich Miescher who called it nuclein[8]. In 1919 Phoebus Levene discovered that DNA consists of chains of linked adenine, guanine, cytosine and thymine molecules[9]. Levene believed that DNA consists of equal amounts of the four nucleotides (tetranucleotide hypothesis). This hypothesis was later refined by Erwin Chargaff who found first hints of the base paired structure of DNA and that DNA from different species had a slightly different composition (Chargaff rules)[10]. It wasn't until the discovery of the DNA structure by James Watson and Francis Crick in 1953 and a subsequent presentation by Francis Crick in 1957, where he postulated the connection between DNA, RNA and proteins[11, 12], that the actual role DNA plays in living organisms was brought to light. Crick created the term "central dogma" for his idea that the flow of genetic information was from DNA to RNA to protein and that this process was irreversible, i.e. DNA can not be produced from proteins.

These findings allowed Har Gobind Khorana, Marshall W. Nirenberg and Robert W. Holley to decipher the genetic code in 1961[13]. The genetic code determines how the sequence of base pairs in the DNA/RNA is translated into the amino acid sequence of the proteins encoded by that DNA. For these ground breaking discoveries Watson and Crick were awarded with the Nobel Price in Medicine in the year 1962. Har Gobind, Nirenberg and Holley also received the Nobel Price in Medicine only a short time later in 1968.

Until that time biochemical research was mostly done in the laboratory where researchers measured properties of individual cell components. Since at that time it was general believe that the proteins were the carriers of the genetic information, much emphasis was put on those. With the discovery of the role of DNA research interest shifted and much work was put into acquiring and deciphering the genetic information of the individual organisms. In the beginning, sequencing of genes was very laborious, manual work but sequencing speed picked up due to the development of new sequencing methods. The first complete genome of an organism, the bacteriophage Φ X 174, was sequenced in 1977[14]. Progress in the area of gene sequencing and gene sequencing methods continued at an astonishing rate and in the years following the publication of the first complete bacteriophage genome, the genomes of several other viruses and bacteria have been sequenced.

In the year 1988 the human genome project was started. This project to sequence the complete human genome ran for more than 13 years and

involved research groups from all over the world. With an estimated cost of 3 billion dollars, the project was able to present a first draft of the human genome in the year 2000 and was considered complete three years later in 2003[15, 16, 17].

In 1998, nine years after the human genome project had started, a private company called Celera Genomics lead by Craig Venter also began to sequence the human genome. Although the human genome project had a head start of many years Celera Genomics managed to finish their sequencing project roughly at the same time as the human genome project. This increased sequencing speed was due to advances in sequencing technologies which also allowed Celera Genomics to not only complete the sequencing in a fraction of the time, but also at roughly 10% of the costs of the human genome project[18].

During the ten years following the publication of the first draft of the human genome by the human genome project gene sequencing technology progressed fast. Today commercial companies provide full genome sequencing as a service at a fraction of the costs of even Celeras efforts. Price goals as low as 1000 or even 100 dollars per complete human genome sequence have recently been announced by various companies and hundreds or even thousands of human genomes have been sequenced since 2001. The exact number is hard to tell since most of the genomes have never been published and can only be estimated based on announcements made by these companies[19, 20].

The race between the human genome project and Celera for the sequence of the human genome and the commercialization of gene sequencing had lead to the development of a number of so called high throughput techniques that enable researchers to create massiv amounts of genomic and other data in a very short time and at moderate costs. This in turn has lead to several new research fields dealing with these large scale data sets and to new associated technical terms (see table 1.1).

Today biochemical research does not exclusively take place in laboratories any more, but more and more research in this field depends on the use of computers. One reason for this trend certainly lies in the fact that the amount of data that has to be handled has grown enormously due to the new methods mentioned above and the data generated by the corresponding fields of research.

Another cause for the increased use of computers might be the fact that research in this area today focuses on understanding complete systems or at least larger parts of these systems instead of single elements like individual reactions or proteins. The reason behind this new trend is the comprehension that the behavior of a complex system like an organism can not be explained by the sum of the properties of its constituents[21]. Complex systems tend

Genome/Genomics The complete genetic information of an organism. The genetic information of an organism normally does not change over time.

Transcriptome/Transcriptomics The complete set of molecules that are transcribed from the genome (mRNA, tRNA, rRNA and other non-coding RNA molecules) of an organisms. The transcriptome of an organism is usually more complex than its genome due to processes like alternative splicing. Since not all genes of an organism are expressed at all times, the content of a transcriptome of an organism depends on many factors like stress, availability and type of nutrition etc. This means that the transcriptome of an organism can change over time.

Proteome/Proteomics The complete set of proteins of an organism and their interactions. Due to post-translational modifications, the proteome of an organisms is usually more complex than the corresponding transcriptome. Since the proteome is translated from the transcriptome, it also changes over time.

Metabolome/Metabolomics The complete of "small" molecules (metabolites) participating in the reactions in an organism. The contents of the metabolome depends on the one hand on molecules acquired from outside the organism (e.g. food) and on the other hand on the reactions that take place and therefore on the proteome. This means that the metabolome of an organism usually also changes over time.

Table 1.1: Short definitions of new technical terms associated with different sets of biological data. The definition item specifies the name of the data set followed by the associated field of research that deals with this data.

to show so called emergent behavior that arises from the interactions of the individual elements. A good example for this principle are colony forming insects like for example ants. The behavior of a single ant is very simple, but the complete colony shows a complexity beyond what would be expected considering the behavior of each single individual in that colony. It is the interaction of thousands to millions of individuals that determines the properties of this system.

Likewise, in order to understand the complex behavior of a living organism, one has to consider a sufficiently large subset of processes and components to account for the complexity caused by the interactions between them[22, 23, 24].

Systems Biology

Systems Biology is the biology
of systems.

undisclosed author

Systems Biology is the name of yet another relatively new field of biochemical research that emerged due to the desire to understand the complexity of living organisms and how this complexity is created from the individual components[25, 26, 27, 28].

The German website for system biology, <http://www.systembiologie.de> defines systems biology as follows:

"Systems biology is devoted to the study of biological processes on a systems level. It focuses on network behaviour, particularly dynamics, through the use of mathematical modeling coupled to experiment. To accomplish this ambitious task, systems biology combines quantitative methods used in molecular biology with approaches from the fields of mathematics, computer sciences, and systems science."

Or short, systems biology tries to simulate the processes that occur in living organisms with the help of a computer. Similar concepts exist in many different fields in research, industry and even entertainment. An example from the automobile industry would be virtual crash tests. There the engineers create a mathematical representation of the mechanical properties of a car and use that mathematical description to compute the potential consequences of a crash using appropriate computer software. Another example

that everyone knows is the weather forecast. There, the different influences on parameters like temperature, wind or air pressure are cast into a set of formulas that is then used to compute the weather forecast that we see each evening in the news.

The set of mathematical expressions that are used for the simulation of some concept, in our case the reactions and interactions taking place in an organism, are often referred to as the model and the process of creating this model is called modeling. In systems biology, there are different types of models as well as different mathematical formalism to describe them. Models are usually divided into two classes. One class are the so called topological models. Topological models describe only the interactions between the elements that are relevant to a model and the mathematical concept graphs is often used to describe such a topological model. In such a graph, the interacting elements, e.g. metabolites or proteins are represented by the nodes and the interactions between them, e.g. reactions or binding processes, are represented by the edges. An example of a simple topological model is given in figure 1.1. Topological models don't have to be associated with numerical values which limits the types of analysis that can be conducted on those models.

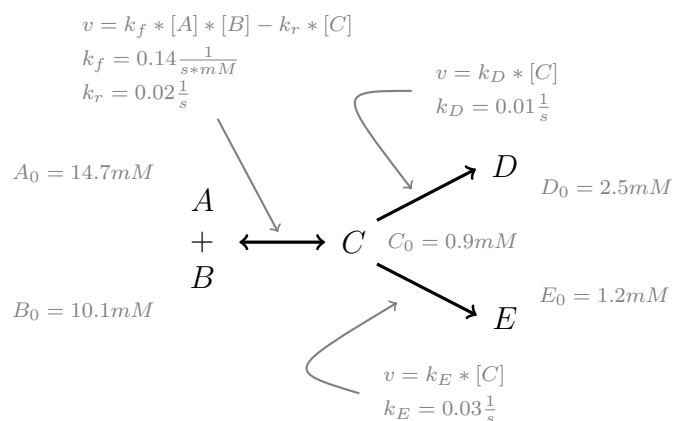


Figure 1.1: Example of a simple reaction network. The dark parts represent a topological model, the lighter parts add the necessary mathematical expression to transform it into a kinetic model.

The other type of models are the so called kinetic models. Kinetic models consists of the same information as topological models, i.e. the interactions and the interacting elements often in the form of a graph, but they supplement this information with numerical values associated with the nodes and

mathematical expressions that describe the interactions. In a deterministic model of a metabolic reaction network this means that the metabolites represented by the nodes are assigned concentration values and the edges which represent the reactions are assigned mathematical formulas that describe the speed of change for the concentrations of the associated metabolites. With the help of these additional elements, the change in concentration over time of all the metabolites of the model can be calculated with the help of a computer and appropriate computer software. The result of such a simulation experiment is called a time course and it is probably the most common type of analysis done on kinetic models.

The approach described above will lead to a set of coupled differential equations, but there are also other approaches to formulate kinetic models for simulation on a computer. In so called probabilistic/stochastic models all the nodes in the graph are also associated with e.g. concentration values, but in this case the edges are associated with probabilities. These probabilities determine the likelihood that the given reaction will occur within a certain time period. The higher the probability associated with a certain reaction, the more often it will probably occur within a given time period. Since this process is governed by random events, the exact time at which a reaction event occurs or how many reaction events occur within a certain time period is not predetermined.

A notorious example for such a probabilistic process is the decay of radioactive elements. For example caesium-137 has a half life of roughly 30 years. This means that if one looks at a large lump of caesium-137 half of the caesium-137 molecules will have decayed to barium within the course of 30 years. If one only has a single atom of caesium-137 however, it is not possible to predict at what point in time this atom will decay, it could be seconds, but it could also be years. There is only a certain probability for the event to occur within a certain time period.

Calculating time courses for probabilistic experiments requires the use of random numbers and usually involves many millions of calculations, so the use of computers for this type of analysis is inevitable. Although certain types of differential equations can be solved analytically, especially linear ordinary differential equations, the types of differential equations encountered in the analysis of biochemical reaction networks normally can't. For most types of differential equations of higher order, no method for solving them analytically is known and the use of special approximation algorithms is required to calculate an approximate solution. Depending on the number of steps calculated during the simulation and on the size of the differential equation system, this also involves millions of calculations and computer hard- and software is necessary for this task as well.

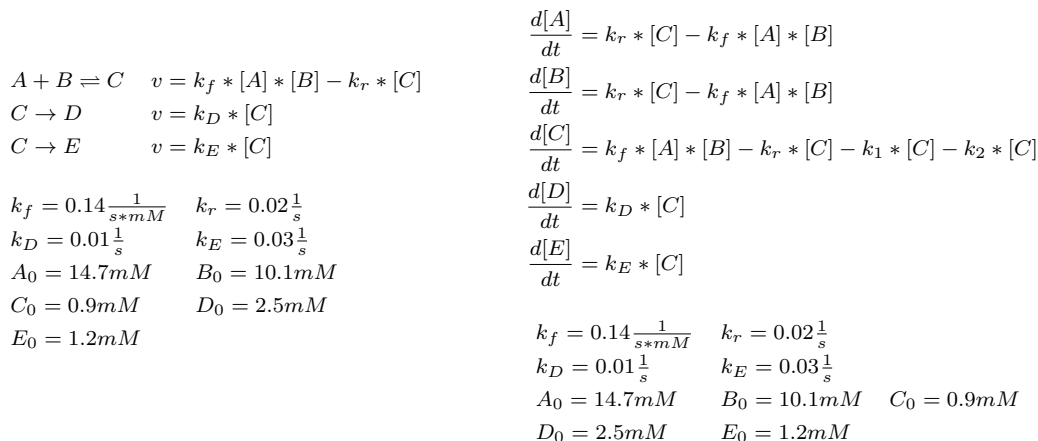


Figure 1.2: Two alternative ways of specifying the reaction network from figure 1.1. On the left the kinetic model is specified as a list of reactions with rate laws and kinetic parameters and on the right it is described as a set of coupled ordinary differential equations (ODEs).

Tools & Standards In Systems Biology

As described above, work in the field of systems biology requires the use of computers and of specific computer software that helps in the analysis of topological as well as kinetic biochemical reaction networks.

So far computer software was only mentioned in the context of calculating time course data from probabilistic and deterministic models, but the use of computer software can be beneficial to researchers during the complete workflow including model creation, simulation and other types of analysis as well as the visualization of the generated data.

Researchers in systems biology come from many different scientific backgrounds and besides biochemists there are biologists, mathematician, physicist, computer scientists and many more. While mathematician and physicists generally are familiar with the concept of differential equations, scientists from other fields usually aren't or are to a lesser extend. This means that while researchers used to working with differential equations will have no problems using any of the generally available mathematical frameworks, researchers with a more biological background can profit from software more geared towards their area of expertise.

Especially researchers with a strong background in life sciences are the users targeted by the COPASI program. COPASI is the successor of the well known and widely used Gepasi program[29, 30]. Gepasi was one of the first and at that time probably the only tool that allowed users with a

background in life sciences to create reaction network models in an intuitive way by specifying reaction equations. Most other tools at that time required users to specify the differential equations which requires a certain expertise and can be prone to errors.

The COPASI software is developed in our group in collaboration with groups from the University of Manchester in the U.K. as well as from the Virginia Biotech Institute in Blacksburg, Virginia, in the U.S.A. COPASI provides an intuitive user interface that supports the user in the creation of biochemical reaction networks by allowing them to specify reaction equations and kinetic laws instead of differential equation systems. The differential equations needed to calculate time courses of these models are afterwards generated from the information specified by the user but without requiring in depth mathematical knowledge about the process. In addition to delivering the user from having to write large sets of differential equations, COPASI provides users with various methods for the simulation and analysis of biochemical reaction networks as well as versatile and intuitive ways of visualizing the results of these analysis. COPASI is free software and so far it has been downloaded several thousand times from our web servers. It is used by scientists all over the world for research as well as in teaching.

Major parts of this work cover the implementation of analysis and visualization methods for reaction networks in COPASI.

The large number of sophisticated and well tested methods for the simulation and analysis of reaction network models in COPASI have also picked the interest of developers of other systems biology tools. Some of them have voiced their interest in being able to use parts of the functionality of COPASI from within their programs. Since COPASI has been written as a standalone program this is not directly possible. This is complicated by the fact that COPASI was written using the C++ programming language while many other software tools use other programming languages like Java, Perl or C# just to name a few. In software development it is not unusual to interface program code written in C or C++ to higher level languages like Java, Perl or Python. Most higher level languages provide some kind of communication interface that allows programmers to write extra code that lets the program code written in the higher level language communicate with the code from the low level language and vice versa. Since the communication interface is different for every high level language this communication code has to be provided for each high level language separately.

With the help of a tool called "Simplified Wrapper and Interface Generator" (SWIG) that partially automates the process of creating the communication code for a large number of high level languages, it is now possible to access COPASI functionality from a number of different programming lan-

The top screenshot shows the 'Reactions' list in COPASI. The reactions are as follows:

#	Name	Equation
1	p65 activation	p65{cytosol} -> p65_2; IKKa_2
2	phosphorylation of IkBa	IkBa_3 -> IkBa{cytosol}; IKKa_2 IKKb_2
3	phosphorylation of IKKa	IKKa -> IKKa_2; IKKb_2
4	phosphorylation of IKKb	IKKb -> IKKb_2; A20{cytosol}
5	degradation of IkBa	IkBa{cytosol} -> IkBa_2
6	p65 transport	p65_2 -> p65{nucleus}
7	A20 trascription	p65{nucleus} -> A20{nucleus}
8	IkBa transcription	p65{nucleus} -> IkBa{nucleus}
9	A20 translation	A20{nucleus} -> A20{cytosol}
10	IkBa translation	IkBa{nucleus} -> IkBa_3
11	p65 deactivation	p65_2 -> p65{cytosol}; IkBa_3

The bottom screenshot shows the 'Differential Equations' view for the cytosol compartment. The equations are:

$$\frac{d([p65]_{cytosol})}{dt} = -V_{cytosol}(0.1[p65]_{cytosol}) + V_{cytosol}(0.1[p65_2])$$

$$\frac{d([p65_2]_{cytosol})}{dt} = +V_{cytosol}(0.1[p65]_{cytosol}) - V_{cytosol}(0.1[p65_2]) - (0.1[p65_2])$$

$$\frac{d([IkBa]_{cytosol})_{cytosol}}{dt} = +V_{cytosol}(0.1[IkBa_3]) - V_{cytosol}(0.1[IkBa]_{cytosol})$$

$$\frac{d([IKKa]_{cytosol})}{dt} = -V_{cytosol}(0.1[IKKa])$$

$$\frac{d([IKKa_2]_{cytosol})}{dt} = -V_{cytosol}(0.1[IKKa_2])$$

Figure 1.3: COPASI supports users with biological background by allowing users to specify reaction equations (top) instead of differential equations, nevertheless, the set of differential equations (bottom) is also displayed to the user if requested. Alternatively if a file contains graphical information, the reaction network can be displayed as a diagram (see figure 3.40).

guages. Currently access from C++, Python as well as Java is possible and work on providing access from Perl, Octave and R are under way. Together with the substantial documentation and examples provided for each of the target languages, this work of mine provides a useful tool to developers and scientists from all over the world as well as several Ph.D. students from our group. The most visible use of COPASI, at least in terms of users, is the integration of COPASI's simulation capabilities in the CellDesigner[31] software. CellDesigner is a tool for the graphical creation of reaction networks and it does not contain much functionality beyond that. A few years ago they approached us and asked if they could use methods from COPASI within their tool. They were the first to use the language bindings for COPASI and this collaboration actually started the whole project of creating language bindings for COPASI.

At the time this work started, most research software tools were created to solve a very specific problem, e.g. there were tools for the stochastic and the deterministic simulation of reaction networks, tools for the calculation of elementary flux modes, tools for parameter fitting etc. and each of these tools required its input to be in a specific format which was incompatible with the file formats for most of the other tools. If a scientist in the course of his/her work had to use several different tools for different types of analysis or to verify the results of one tool with another, this often required the model to be recreated in the format specific to each of these tools. Needless to say that this created a lot of extra work and made the whole process somewhat cumbersome.

To address this situation, a number of scientist got together in order to come up with a standard for the exchange of reaction networks. Unfortunately not all of these scientist got together at the same time and at the same place and therefore this idea of a common exchange format was conceived several times around the world at roughly the same time, but in different systems biology communities. This is the reason that today we don't have one common standard for storing reaction network models but (at least) three standards¹ which are supported by several groups. Today these standards are competing with each other to some extent, but at the same time members of the three communities are trying to collaborate on new developments in that area.

The work in this thesis is mostly concerned with the Systems Biology Markup Language (SBML) standard, its development and extension as well

¹Since none of the three formats have actually undergone a standardization process yet, the word standard is used rather liberally in this context.

as its implementation in the form of computer software. Nevertheless the similarities as well as the differences between the different standards shall be highlighted briefly:

CellML CellML is a standard for exchanging mathematical models of complex systems. It has been developed by the Auckland Bioengineering Institute at the University of Auckland in New Zealand[32, 33, 34]. CellML is a very general file format allowing the user to encode all kinds of models not just reaction network models. The emphasis for CellML has been put on generating a generic file format and on modularity, allowing the user to combine different models to form larger models from these smaller components. The disadvantage of having such a general file format is increased complexity. Because the file format cover such broad range of applications, there are not a lot of tools that can support all CellML features. CellML has been developed in New Zealand and according to the website of the developers, most groups that use CellML for their research seem to come from the Asian/Pacific region as well as from Oxford in the U.K.[35]. The first publications related to CellML appeared in the year 2000 and 2001 in the Proceedings of the Physiological Society of New Zealand[36, 37]. It is pure speculation, but maybe the journal does not have a high visibility outside New Zealand and therefore not a lot of people noticed these publications.

BioPAX Another standard that is used to exchange data between a number of research groups and software packages is BioPAX[38, 39]. The center of activity related to BioPAX lies in the U.S.A., but several groups from Europa are also participating in the development of this format. BioPAX is not as general as CellML. It is more geared towards describing models of biochemical reaction networks and emphasis has been put on providing detailed meta information about the biological meaning of the constituents of the models. Although the first publication for the BioPax specification appeared only very recently in 2010[38], the effort behind it is several years old and the paper only covers the latest release called Level 3. The first version of the specification that is available on the internet is termed Level 1 Version 1.4 and has been released in 2005.

SBML Is the Systems Biology Markup Language which also is a format for the exchange of mathematical descriptions of reaction networks. Here the emphasis lies on the exchange of the mathematical description mainly for use in time course simulations[40, 41]. Although the

first publication of the SBML standard appeared in the year 2003, the corresponding final version of the specification was released in 2001. The SBML web page is listing more than 100 software tools that are supposedly support this standard which probably makes it the most widely used of the three formats. This however does not imply that all the tools support all versions of SBML or even all features of a single version of SBML.

There is also another standardization initiative called Systems Biology Graphical Notation (SBGN)[42]. This initiative overlaps somewhat with the three formats mentioned above as well as with the work described in this thesis. The SBGN standard is not a standard for the storage of the mathematical description of biochemical reaction networks but it defines a set of rules of how such reaction networks can be represented graphically in an unambiguous way. The intention behind this is to have a set of symbols that can be combined in certain, well defined ways to represent diagrams of biochemical processes. This is the same concept that is used when drawing circuit diagrams in electrical engineering (see figure 1.4). In this respect it complements the other three standards and members from the different communities are collaborating on the SBGN effort.

I am participating in the development of SBGN and parts of this work aim at providing support for creating and displaying SBGN diagrams within the COPASI software.

With the amount of data usually involved in the creation of biochemical reaction networks as well as the large amounts of data that are created by some of the analysis methods, it is very important to visualize this data in an intuitive way or to be able to visualize the same data in different ways in order to highlight different aspects of a problem.

In our group, software for the visualization of such data has been created very early on and several people in our group were working on different visualization tools in parallel. This lead to the situation that we eventually had several tools that displayed analysis results on a reaction network diagram, but each of the programs was using a different way (or no way at all) to store the diagrams that had been created in the process. Since this was sub-optimal, we started to look for a common format that would let us store this diagram information. This was around the same time the first SBML paper has been published in 2003. Eventually we found that none of the existing solutions we considered could provide us with a reasonable solution that on the one hand was versatile enough to be used in each of our applications and

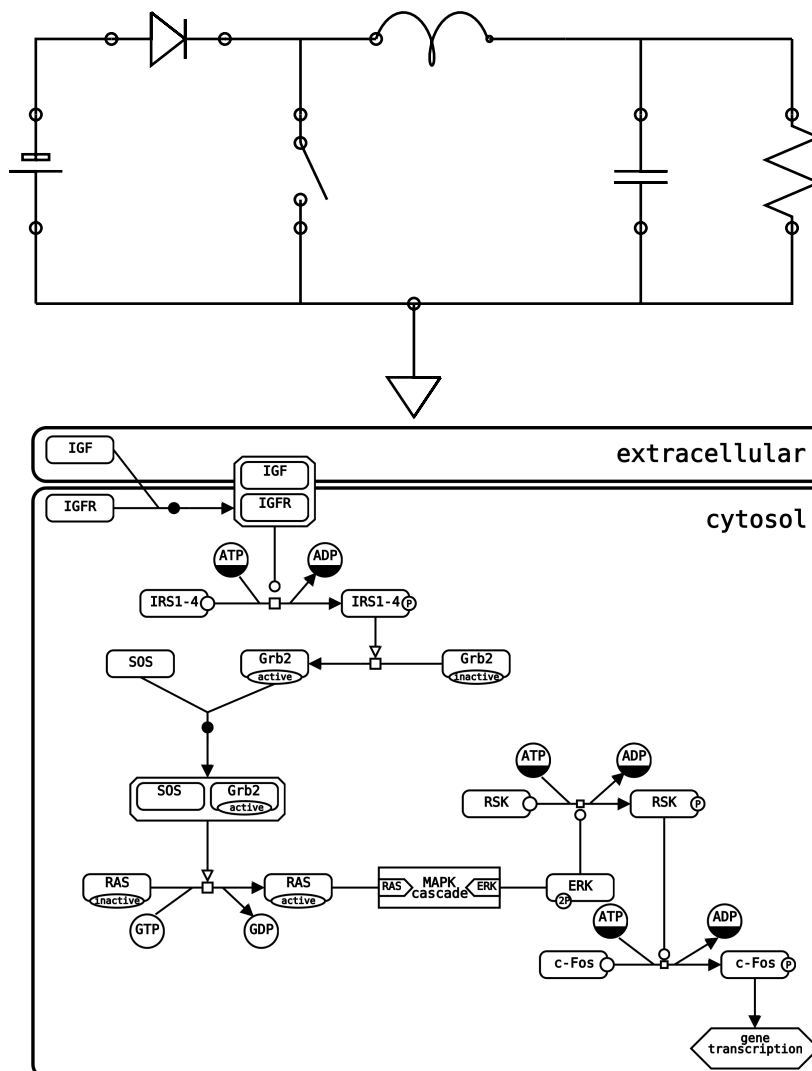


Figure 1.4: Example of an electrical circuit diagram (top) and an SBGN diagram (bottom). The electrical circuit diagram is an example provided with a Latex package for writing such diagram and the example as well as the package have been written by Umesh Rai[43]. The SBGN diagram is the reproduction of a diagram from the SBML specification using the SBML layout and render extension.

at the same time simple enough so that an implementation was feasible. All we found was the SBML format which allowed the exchange of mathematical descriptions of reaction networks, but not graphical information. However, it provided a way with which it could easily be extended with arbitrary additional information.

In the following years, an official specification for extending the SBML format with graphical information has been worked out by me with considerable help from several members of our group. Other developers of systems biology software also became interested in the extension and useful feedback by the community was provided leading to a very stable and useful extension to the SBML format. This has been shown by several proof of concept implementation for the extension as well as implementations that are now used in different software tools. Besides the implementations written by myself, there are also some implementations of the specification by other developers from the SBML community.

Although the specification was meant to be used in conjunction with SBML files, we wrote it in a way that would keep it mostly agnostic of the underlying file format. This independence from the underlying file format allowed me to later use the same format for storing graphical information on top of COPASI's native file format. Due to this reuse of the existing format, I could also reuse most of the code written for the SBML specific implementation. Additionally writing the program code for the conversion between graphical information stored in SBML files and graphical information stored in COPASI files and vice versa was straight forward.

As this graphical extension to SBML and its derivatives are the basis for many of the visualization methods described in this thesis, a large fragment of this text details the extension itself, its implementations and its use for the visualization of reaction networks and analysis results.

The last topic of this thesis describes my work on creating a framework for the normalization of mathematical expressions.

When dealing with kinetic models, a lot of mathematical expressions have to be handled by computer programs and in certain situation it would be beneficial if these expression could be compared and identified. One simple example is the comparison of complete models. If we have two separate models and we want to find out how much these two models overlap, we need to be able to compare the elements in the two models as well as the mathematical expressions in these models. Another situation where it would be useful to be able to identify a mathematical expression is for validating the annotations of a model. Some models contain additional information

that associates elements of the model with biological meaning. For example a metabolite in a model could provide extra information which specifies that this metabolite represents ATP. This association is created via references to unique identifiers from certain biological databases. For the example above, the identifier `CHEBI:15422` could be associated with the metabolite in the model. This is the unique identifier for ATP from the ChEBI database[44, 45] and this association establishes that the metabolite represents the molecule ATP. There are similar ways of assigning biological meaning to the kinetic laws in a model and this additional information could for example specify that a certain expression in the model represents a mass action kinetic formula. Unfortunately it is possible for the annotation to specify that a mathematical expression represents e.g. a mass action rate law, while the expression itself is for a totally different rate law. In order to identify such discrepancies I have implemented a mathematical framework that normalizes arbitrary mathematical expressions so that they can be compared more easily. A very simple example of why this normalization is necessary in order to be able to compare mathematical expressions is represented by the two equations below:

$$\begin{aligned}v &= V_{max} * (S / (K_m + S)) \\v &= (S * V_{max}) / (S + K_m)\end{aligned}$$

A scientist looking at these two equations would probably recognize immediately that both formulas might represent a Michaelis-Menten type rate law and that the elements of the expression have only been rearranged. A computer program on the other hand can't determine if these two equations are equivalent without rearranging the elements of both equations in a defined way prior to making the comparison. So what the framework I have implemented does is to rearrange the two expressions according to a set of rules and in a way that does not change the semantics of the expression. For this example the result of the normalization might look like this:

$$\begin{aligned}v &= (S * V_{max}) / (K_m + S) \\v &= (S * V_{max}) / (K_m + S)\end{aligned}$$

Now the computer program can compare the two equations and will hopefully determine that they are identical.

Application Of The Tools And Methods

Besides developing computer programs for systems biology, we are also participating in different research projects where we collaborate with different theoretical as well as experimental groups. Our experiences in these projects as well as the feedback from our collaboration partners is used to improve our programs and methods to further facilitate the complex process of model building and analysis.

Currently we are part of the "Virtual Liver" network[46]. Parts of this initiative deal with the create models of different processes in liver cells and their integration into larger models to simulate certain aspects of human liver functionality. The models, methods and tools created by this effort are supposed to bring new insights on the mechanisms occurring in the liver as well as promote medical research in this area.

One special area of interest in this project is liver regeneration or more specifically the mechanisms and processes that lead to the total recovery of the organ even if large parts have been damaged or removed[47].

Several different signaling pathways in liver cells are known or suspected to play a major role in these regeneration processes[48, 49, 50], but some of the details and especially the ways in which the different signaling pathways interact are not fully understood. In order to shed some light on these processes some of the sub-projects in the Virtual Liver Network are going to create models for the individual signaling pathways and combine those models in order to be able to understand how they are connected.

One of these sub-projects we are involved in is the creation of a model for the so called Hippo (or Hpo) signaling pathway. The Hippo pathway has first been discovered in drosophila where it regulates cell proliferation and apoptosis[51, 52]. This pathway is highly conserved in mammals where it is involved in the regulation of cell density dependent cell growth, which is an important aspect in liver regeneration.

Even so the fact that the Hippo pathway plays an important role in the regulation of liver size and regeneration has been established, it remains unclear how this pathway works in detail[53, 54]. Especially specifics concerning the crosstalk with the relatively well studied NF- κ B signaling pathway are unresolved.

In this project we are working together with the group of Dr. Kai Breuhahn from the German Cancer Research Center in Heidelberg (DKFZ). This work started only very recently and we are currently in the stage of creating initial models for the two signaling pathways. With the help of the experimental data provided by Dr. Breuhahns group as well as their knowledge about those signaling pathways involved, we hope to eventually be able

to create a reaction network model that can describe the combined effects of the NF- κ B and Hippo signaling pathways on liver regeneration.

During this collaboration many of the methods and principles described in this thesis have been applied to systems biology research. This will be discussed in more detail together with the corresponding topics in the following chapters.

Chapter 2

Materials & Methods

2.1 Operating Systems

During this work several operating systems were used to write, test and run the different software tools. The following sections will give a short overview over these operating systems.

Unix & GNU/Linux

The Unix operating system describes a whole family of operating systems. The Unix standard and the first implementation was developed in the early 1970s in the Bell Laboratories. Today there are many implementations adhering to the "Single Unix Specification"[55] standard.

Examples of Unix compliant operating systems are Oracle Solaris[56], formerly known as Sun Solaris, HP-UX[57] by Hewlett Packard, AIX[58] by IBM.

Operating systems adhering to the standard but that have not been officially certified are often called Unix-like operating systems. These include the different BSD[59] versions or the different GNU/Linux[60] versions.

Unix operating systems are multitasking, multi-user operating systems that have been used in the scientific field for a long time. Especially the free Unix-like operating systems like BSD and GNU/Linux that can run on commodity hardware have led to the spread of Unix.

The graphical user interface for the Unix and Unix-like operating systems is predominantly provided by the X Windows system[61].

During this work the following Unix and Unix-like operating systems have been used: Oracle Solaris & GNU/Linux (Debian[62] for AMD64, Ubuntu[63] for x86 and AMD64, Debian for PPC)

Mac OS X

Mac OS X is the operating system shipped with all current computer from Apple Inc.[64].

The Mac OS X kernel is based on parts of FreeBSD[65] and NetBSD[66] which makes it one of the Unix-like operating systems.

The difference to the BSD clones and other Unix operating systems is that Mac OS X uses a custom, proprietary graphical user interface not available for the other operating systems.

During this work the following versions of Mac OS X have been used: Mac OS X 10.2 for PPC, Mac OS X 10.3 for PPC, Mac OS X 10.4 for PPC and x86, Mac OS X 10.5 for PPC and x86 as well as Mac OS X 10.6 for x86.

Microsoft Windows

Microsoft Windows[67] is the operating system developed by Microsoft and which is shipped with almost all new x86 based computers.

The operating system is based on a proprietary kernel as well as a proprietary graphical user interface and it is usually restricted to x86 CPU based computer hardware.

Due to the fact that most computers come pre-installed with some version of Microsoft Windows it is the most widely used operating system today.

During this work the following versions of Microsoft Windows were used: Windows XP (32 Bit), Windows Vista (32Bit & 64Bit) and Windows 7 (32Bit & 64Bit)

2.2 Programming Languages

This work deals extensively with the development of methods and standards and their implementation on computer hardware. For this purpose different programming languages have been used.

C++

The C++ programming language was developed as an enhancement to the C programming language called "C with classes" in 1979 by Bjarne Stroustrup at the Bell Laboratories[68]. In 1983 it was renamed to C++.

C++ is a general purpose compiled language which means that it can be used to write arbitrary computer programs and that the program code

written as text (source code) by the developer is compiled into a form (executable binary) that can be processed by the computers central processing unit (CPU).

For the compilation of C++ programs, another computer program, a so called compiler, is needed.

The C++ programming language has been standardized in 1998 and since the standard is open, everyone that is interested can implement a standard compliant C++ compiler.

As the language as well as the standard have existed for many years now they are very mature and many high quality compilers, commercial as well as non-commercial, are available for the different operating systems.

Its maturity and the availability of a number of high quality C++ compilers on almost all operating systems, was the reason that C++ was chosen as the main programming language for the implementation of COPASI[69].

The first implementations of libsbml[70], a library for reading SBML documents, were based on the C programming language, but recent versions are also implemented using the C++ programming language.

Not all compilers exist for all operating systems and even if some compiler is available for a number of operating systems, it is not guaranteed that the code generated on all operating systems has the same quality. Due to these reasons several different compiler from different compiler manufacturers have been used for this work.

For the Windows operating system, C++ developers have a choice of several free and commercial compilers. Free compiler used to compile code under Windows were the GNU Compiler Collection (GCC)[71] as well as different versions (2008, 2010) of Microsoft Visual C++ Studio Express[72]. As a commercial compiler, the Intel C++ compiler[73] was used because it can sometimes provide executable binaries that run faster than the binaries build with GCC or Visual C++.

On Mac OS X, GCC is pre-installed and was used to compile most of the C++ code on that platform. Occasionally the commercial Intel C++ compiler was used to get executable binaries with improved speed as well as to cross check for potential errors in the code.

Under Linux and Solaris, the GCC compilers were used as a default. Occasionally the free C++ compiler from Sun[74] or the commercial C++ compiler from Intel was used. Especially when cross checking for errors in the code.

Java

The Java programming language is an interpreted general programming language developed in 1995 by James Gosling at Sun Microsystems[75]. The syntax of the language is based on C++, but several language elements have been dropped or modified to make the language less complex.

In contrast to C++ source code, Java source code is usually not compiled to a binary executable that can be directly interpreted by the CPU, but to an intermediate byte code that needs to be interpreted by a so called Java virtual machine (JavaVM or JVM). This is done by a Java compiler. The JVM finally translates the intermediate byte code into code that can be executed by the CPU. This translation is usually done by a Just-in-time (JIT) compiler which for some source code can provide almost the same speed as a program written in e.g. C++.

Java is not a standardized language and Sun Microsystems (now owned by Oracle) determines the direction of the development of the language.

There are several implementations of Java compilers and Java virtual machines.

For the most common operating systems like Windows and Linux these are provided by Sun/Oracle. Since support for other operating systems is not directly provided by Sun/Oracle, support on these operating systems usually is several versions behind the latest stable version released by Sun/Oracle and sometimes doesn't provide the same level of quality and compliance to the Java specification.

Apple provides a version of the Java compiler and the virtual machine with Mac OS X, but it usually lags behind the current implementation by Sun/Oracle.

Today Java is widely used for writing programs in the scientific community[31, 76, 77], especially with respect to client/server programming.

The Java programming language has been used in this work to write the Java language bindings for the implementation of the SBML Layout and Render Extension in the different versions of libsbml as well as to implement the Java bindings for the COPASI API.

Python

Python is a general purpose, high level scripting language developed by Guido van Rossum in the late 1980s, early 1990s[78].

Scripting languages are similar to Java in so far as the source code is not compiled to a binary that is interpretable by the CPU, but a so called

interpreter takes the source code and translates it to code that the CPU can process. This translation is done each time the program is run. Sometimes an intermediate binary file that is easier to interpret is created in the process.

The main advantage of scripting languages like Python is their ease of use compared to compiled languages like C++. The program does not have to be compiled, but the source code is directly executed in the interpreter. Since scripting languages are usually high level languages, they provide the user with sophisticated constructs that allow the development of programs with less code in less time.

Due to the fact that the program code has to be interpreted each time the program is run, the increase in flexibility and ease of use comes at the price of slower running times of the programs. So scripting languages like Python are mostly used for programs where execution time is not an important factor.

Due to its relatively simple syntax, Python has gained a lot of support over time and is widely used in the scientific community[79, 80, 81, 82].

The Python programming language has been used in this work to write the Python language bindings for the implementation of the SBML Layout and Render Extension in the different versions of libsbml as well as to implement the Python bindings for the COPASI API.

Perl

Perl[83] is another general purpose, high level scripting language developed by Larry Wall in 1987. It is very similar in scope to Python and shares the same advantages and disadvantages. It is also widely used in the scientific community, e.g. BioPerl[84].

The Perl programming language has been used in this work to write the Perl language bindings for the COPASI API.

GNU Octave

GNU Octave[85] is a free clone of the Matlab[86] numerical computing environment. It includes a general purpose, high level scripting language with a strong emphasis on mathematical methods and concepts and it comes with a user interface for the display of graphical output and numerical results.

Octave in its current form was started in 1992 by John W. Eaton and is now developed by a number of volunteers under the GNU aegis.

Although it is not 100% compatible with Matlab, it can replace the commercial program for many tasks.

The Octave programming language has been used in this work to write the Octave language bindings for the COPASI API.

the R programming language

The R programming language[87] is also a general purpose, high level scripting language but with a strong emphasis on statistical computing and graphics.

R is an implementation of the S programming language originally developed by John Chambers at the Bell Laboratories[88].

Ross Ihaka and Robert Gentleman implemented R in 1993. Today it is developed by a team of developers under the GNU aegis.

The R programming language has been used in this work to write the R language bindings for the COPASI API.

2.3 Unit Testing

With computer programs becoming ever more complex, manually testing all the functionality after making changes is usually no longer possible and automatic testing methods need to be used.

In unit testing the programmer writes a number of tests that systematically check different aspects and parts of a computer program. These tests can be run automatically and can assure the quality of existing source code.

Some programming paradigms[89] propose to write tests for new functionality before implementing the actual functionality. This way the tests replace the formal design and the final implementation can then be checked as to how well it implements the design provided by the test cases.

During this work, a lot of source code in different programming languages has been written and a number of unit testing frameworks have been used to ensure the quality of the software.

libcheck

libcheck[90] is a simple unit testing framework for the C (and C++) programming language(s). It is modeled after other similar frameworks like, e.g. JUnit (see below).

The library is released under the GNU Lesser General Public License (LGPL)[91] and binary packages for libcheck are available for many of the operating systems used during the work described in this thesis.

libcheck was chosen by the developers of libsbml as unit testing framework to test the C and C++ based implementation of libsbml.

For this reason, libcheck was used in this work for writing unit tests for the C++ based implementation of the SBML Layout and Render Extension for the different versions of libsbml.

CppUnit

CppUnit[92] is also a unit testing framework for C++ programs and it is similar in scope to libcheck. It is modeled after the JUnit framework (see below). In contrast to libcheck, CppUnit is a pure C++ based framework providing a number of features not provided by libcheck.

CppUnit is released under the LGPL license and pre-compiled binary packages are available for most of the operating systems used in this work.

CppUnit was used in this work to test the software implementations in COPASI, especially with respect to implementing support for the SBML standard including the SBML Layout and Render Extension.

JUnit

JUnit[93] is the most popular unit testing framework for the Java programming language. The framework is released under the Common Public License aka Eclipse Public License[94] and pre-compiled packages are available for most of the platforms used in this work.

JUnit is the unit testing framework used in libsbml to test the implementation of the libsbml Java bindings.

In this work, the JUnit framework was used to test the Java bindings of the SBML Layout and Render Extension in the different versions of libsbml as well as the Java bindings for the COPASI API.

PyUnit

PyUnit is the name under which the `unittest` package of the Python language is referred to. The unit testing package comes with all versions of the

Python language and is therefore pre-installed on many operating systems, e.g. Mac OS X or most Linux distributions.

The framework is modeled after the JUnit framework (see above) and is released under the Python License[78] which is the license of the Python programming language.

PyUnit is the unit testing framework used in libsbml to test the implementation of the libsbml Python bindings.

In this work, the PyUnit framework was used to test the Python bindings of the SBML Layout and Render Extension in the different versions of libsbml as well as the Python bindings for the COPASI API.

2.4 Debugging & Profiling Tools

A large part of every software implementation consists of finding and eliminating errors or performance bottlenecks in the programs source code. This is a very complex task and support by good software tools is paramount to this work.

Each operating system as well as each programming language comes with its own set of profiling and debugging tools. Therefore in the course of this work a large set of different debugging and profiling tools has been used.

the GNU debugger (GDB)

The GNU Debugger[95] is the debugging tool that comes with almost every flavor of Linux and/or Unix and is usually installed together with the compilers from the GNU Compiler Collection (see above). On Mac OS X, GDB is installed together with the rest of the software development tools.

A debugger like GDB allows the developer to single step through the instructions of a running program in order to find the place in the source code where an error occurs.

In addition to this, it allows the user to examine the state and the data of a program as well as monitor data for modifications during runtime. The program can for example be used to notify the developer about unexpected changes in the programs data which might point to errors.

DBX

DBX[96] is the debugger that comes with the compiler from Sun and is available on Linux as well as Solaris. In theory it provides the same kind

of functionality as GDB, but especially when used on the Solaris operating system, it provides some additional features not found in other debuggers. In certain situations this provides additional help in finding errors.

IDB

IDB[97] is the debugger from Intel and it too provides the same functionality as GDB or DBX. If an executable binary has been compiled with the compiler from Intel, idb sometimes provides advantages over GDB when debugging the program.

WinDbg

WinDbg[98] is the debugger from Microsoft and has been used during this work when debugging programs compiled with Visual C++ from Microsoft under the Windows operating system.

Valgrind

Valgrind[99] is an invaluable debugging tool when it comes to analyzing memory access problems or so called memory leaks in a program.

Valgrind simulates the main memory of a program and monitors all memory access, notifying the developer about access to memory that either does not belong to the program or about memory that has not been properly released once it is no longer needed.

Valgrind has been available for the Linux operating system for several years and recently it has been ported to Mac OS X.

the Java debugger (JDB)

The Java debugger provides the same functionality for Java that for example GDB provides for C++ programs.

The user can single step through the running Java program and can display the state of the program while it is executed by the Java virtual machine.

JDB was used for finding errors in the Java bindings for the SBML Layout and Render Extension in libsbml as well as for finding errors in the Java language bindings for the COPASI API.

the Python debugger (PDB)

The Python debugger (PDB) provides the same functionality as GDB, but for programs written in Python.

PDB was used for finding errors in the Java bindings for the SBML Layout and Render Extension in libsbml as well as for finding errors in the Java bindings for the COPASI API.

gprof

gprof is a profiler that allows developers to find performance bottlenecks in compiled programs and it works in conjunction with the GNU Compiler Collection.

The program to be profiled has to be compiled with profiling information and linked against the gprof library. This allows the program to write a detailed log file while running. This log file contains information on how much time is spent in the individual parts of the program or how often a certain part is executed. This allows the developer to identify the regions of the programs that consume most of the time allowing him to maybe improve the code for those regions. This often leads to programs with significantly faster running times.

Mac OS X OpenGL profiler

The development tools for Mac OS X provide a so called OpenGL profiler program. The OpenGL profiler provides a graphical user interface for profiling and debugging OpenGL (see below) based functionality in programs.

The functionality is similar to the combined functionality of a debugger and a profiler, but in this case the developer can display the resources and timing data used for OpenGL based drawing code, e.g. the contents of render buffers or the memory allocated for textures.

This tool was used extensively for finding and removing errors while implementing the rendering library for SBML Layout and Render information as well as for finding and removing errors in the implementation of all the graphical display routines in COPASI.

2.5 Libraries & Standards

A few years ago, programs written in the scientific community usually interacted with the user via a textual interface where the user had to answer a few simple questions and the program then calculated some result which was presented in the form of textual output.

Today most programs provide the user with a more or less sophisticated graphical user interface and the user interacts with the application by clicking on so called graphical user interface widgets with the mouse.

Writing such a library for a graphical user interface is a major task taking many years of development time. Due to the complexity of this task, not every software project can invest the time needed to create their own user interface elements.

Today software developers use existing libraries of graphical user interface elements with which they build up the user interface for their program. This allows them to write sophisticated user interfaces in only a fraction of the time.

This concept of reusing existing libraries for certain tasks also extends to other concepts like reading and writing files in a certain format or plotting results in two or three dimensional graphs.

Today there are many libraries for developers to choose from to handle certain tasks like displaying a user interface or reading XML files. This makes the task of writing complex user friendly programs easier and allows developers to concentrate on the core functionality of a program.

While many of these libraries have been developed by the open source community, which means that everybody can use them free of charge, other libraries and tools are being developed by companies and developers have to pay in order to be able to use them.

Qt

Qt is a library mainly for writing graphical user interfaces (GUIs)[100]. Newer versions of Qt also contain elements for different tasks like reading XML files or accessing databases, but only very little of this additional functionality has been used during this work.

The main advantage of Qt over other similar GUI toolkits is its cross platform portability. Code written for one platform can be compiled on any other platform supported by Qt, even if the underlying graphical system is different. The application will look and behave the same on all platforms while at the same time providing the user with the specific look and feel of programs that is platform or operating system specific. For example if a

program is compiled for Microsoft windows, the menu bar will be at the top of the main window of the application while on Mac OS X, the menu bar for the same program will be at the top of the screen. The core functionality of the program however will be the same on Windows and on Mac OS X.

This is a big advantage from the developers perspective as well as from the perspective of the users.

Qt provides support for all major and even minor platforms since it supports the graphical framework that is used on Microsoft Windows, the X Windows System that is used on all major Unix and Linux versions as well as the graphical front end of the Mac OS X operating system.

During the course of this work, several different major and minor versions of Qt have been used. Starting with Qt version 2 over Qt version 3 to Qt version 4 which is the version that is currently used in our projects.

Until Qt version 4, Qt was licensed under a dual license where developers could choose between the GNU General Public License and a Qt specific commercial license.

Since not all of our projects fulfilled the requirements of the GPL, we used commercial licenses for Qt until Qt 4.

Starting with Qt 4, the license under which Qt is released has changed to the GNU Lesser General Public License which all our projects fulfill. This allows us to use a non-commercial license for Qt.

In addition to the base functionality of Qt, other libraries based on Qt have been used in programming the user interface for COPASI, especially for the graphical display of results.

One additional library used to graphically display analysis results is Qwt[101]. Qwt is a free library for plotting of two dimensional data, e.g. time course data.

Qwt is released under the GNU Lesser General Public License.

Another library based on Qt that is used in our projects is Qwtplot3D[102]. As the name suggests, this library is similar in scope to Qwt, but while Qwt provides methods for drawing in two dimensions, Qwtplot3D provides methods and widgets for the display of data in three dimensions.

An important feature provided by Qt that is not directly related to creating the graphical user interfaces but that was used extensively in this work is the qmake tool.

How an executable binary is build from the source code usually differs

from operating system to operating system and developers writing programs that are intended to run on several platforms have to use a different build system on each platform.

qmake eliminates this problem by providing an operating system independent build tool where developers only have to specify which source files belong to a certain project plus some additional information about the project. From this information qmake creates all the files necessary to build the project on the given operating system.

The qmake based build process is used to build COPASI as well as some of the other programs, e.g. the stand alone demo rendering application.

OpenGL

OpenGL[103] is a cross platform programming interface specification for the development of 2D and 3D computer graphics. The first version of OpenGL, OpenGL 1.0, was released in 1992 and it has since been continuously developed by a consortium of members from the soft- and hardware industry including big companies like Microsoft, Apple Inc., Silicon Graphics Inc., NVIDIA Corporation, AMD, ATI, etc.

The API is supported by most graphics card manufacturers which means that hardware support for OpenGL is usually found in the drivers for all graphics cards. This makes drawing with OpenGL very efficient and fast.

Other advantages of OpenGL are that it is very stable and mature and that it is independent of the programming language used for writing the program. Thus OpenGL code can be written in C++ as well as in Python, Perl, Java or most other programming languages.

With the advances in hardware, new features were added to OpenGL with each new version. The latest version of OpenGL is OpenGL 4.1 released in July of 2010.

While in general the OpenGL specification has cross platform support and is independent of the programming language, new versions of OpenGL are not supported on all platforms yet.

Proprietary drivers from NVIDIA and AMD provide full support for OpenGL under Windows and Linux, but only for relatively new hardware. The open source drivers that are used in many Linux distributions, currently only support OpenGL up to version 2.1. OpenGL on Mac OS X is also only just now making the transition from OpenGL 2 to OpenGL 3.

In order to support as many operating systems and hardware types as possible, the programs described in this work only use OpenGL features up to and including OpenGL 1.5. This version is supported on most graphics

hardware and most operating systems, providing a very high level of compatibility.

Using an older version of OpenGL doesn't have any major disadvantages, safe for the fact that we can not use some of the newer features on new hardware and that the performance on new hardware might be sub-optimal, but still fast enough for our purpose.

Extensible Markup Language (XML)

The Extensible Markup Language (XML)[104] is a standard for storing textual data in a hierarchical structured way.

The XML standard has been developed by the W3 consortium and the first version of the XML 1.0 specification has been released in 1998.

In XML files, data is arranged in a hierarchical, tree like structure with elements and attributes.

Elements can either be tags which are enclosed in a pair of angled brackets (<>) or text elements.

Tag elements can be supplemented by attributes and child elements and each tag element has to be ended by a corresponding closing tag element which has the same name, but starts with a / (see figure 2.1).

There are many different file formats that are based on the principles of the XML specification, e.g. the XHTML format that is used to serve web pages from web servers to web browsers.

There exist several libraries for reading and writing XML files and there are two different methods how these libraries present the structures read from an XML file to the program.

Some libraries implement the "Simple API for XML" (SAX) where the XML file is split into individual tokens and these tokens are passed on to the program using the library. The program only sees the token that is currently parsed by the library.

Other libraries implement the so called "Document Object Model" (DOM) where the complete XML file is read into a tree like structure called the Document Object Model, and the program that uses the library works with this tree data structure.

There are also libraries that implement both of these programming interfaces.

The advantage of the "Simple API for XML" approach is that it only needs to read small parts of an XML file to hand them on to the program. This makes it very resource friendly because it does not need a lot of memory.

The DOM approach always needs to keep the complete XML tree in memory which might be problematic if the file is very large. On the other

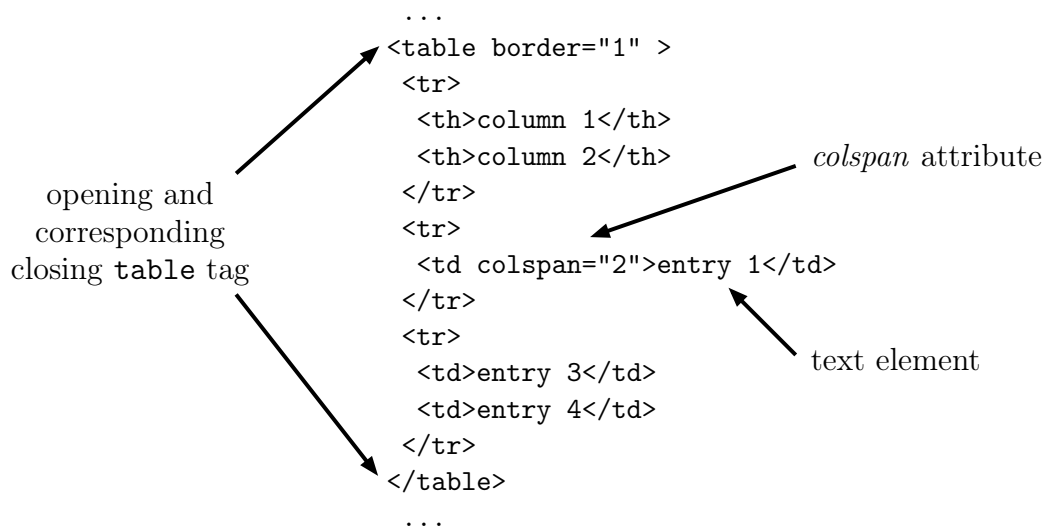


Figure 2.1: Definition of a table in XHTML notation as an example of an XML structure. The table definition in XHTML starts with the `table` tag. This tag has an attribute called `border` which is assigned a value of 1. The table tag has child tag elements (`tr`) which define the rows of the table and those child tags again have children to define the cells in each row. All elements are ordered hierarchically and each opening tag has to be ended by a corresponding closing tag with the same name and an additional `/` at the beginning.

hand, the program gets to work with the complete tree which makes certain things easier to implement.

In this work, the expat[105] library which implements the Simple API for XML was used for reading and writing XML files in COPASI.

libsbml can be configured to use expat, xerces[106] or libxml2[107] for reading and writing XML files. Therefore all of these libraries were used for testing the correctness of the implementation of the SBML Layout and Render extension implemented in libsbml.

An important tool for working with XML files is xmllint. xmllint is an XML checking tool that is part of the libxml2 distribution. It can test if an XML document is well formed according to the XML standard and it can reformat well formed documents to make them more human readable.

In conjunction with an XML schema[108] document or a RELAX NG[109] document (see below), xmllint can check if a given XML document conforms to a specific XML dialect, e.g. if a given XML document is a valid SBML Level 1 Version 1 file.

Resource Description Framework (RDF)

The Resource Description Framework[110] is a standard defined by the W3 consortium for the storage of metadata.

The format describes entity-relationship elements consisting of a subject, a predicate and an object.

All data is stored in these triplets and the set of triplets can form directed graphs (see figure 2.2) with the subjects and objects representing the nodes of the graph and the predicates representing the edges.

The RDF specification provides several ways to store these triplets, but the format that is of interest in the context of this work is the XML data format.

Metadata as suggested by the MIRIAM standard is stored in SBML and COPASI files using the RDF format.

The predicates in SBML and COPASI files are taken from a list of allowed predicates, as e.g. specified in the section called "A standard format for the annotation element" in the SBML specification for SBML Level 2 Version 4[113]. The nodes (subjects and objects) are URIs pointing to elements in biological databases (see figure 2.2).

COPASI uses the Raptor RDF Syntax Library[114] to read and write RDF data from SBML files and from COPASI files.

```

...
<rdf:Description rdf:about="#_180340">
  <bqbiol:isVersionOf>
    <rdf:Bag>
      <rdf:li rdf:resource="urn:miriam:kegg.pathway:hsa04110"/>
      <rdf:li rdf:resource="urn:miriam:reactome:REACT_152"/>
    </rdf:Bag>
  </bqbiol:isVersionOf>
</rdf:Description>
...

```

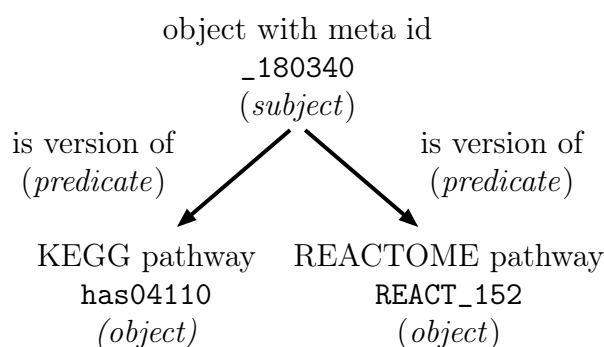


Figure 2.2: Example of RDF information in XML format and the resulting graph. The RDF text describes two triplets. The subject is an SBML model which has the meta id `_180340`. The predicate is the tag `bqbiol:isVersionOf` which is followed by two objects which point to two entries in two different biological databases. The RDF information specifies that the model described by the SBML document is a version of the pathway `hsa04110` in KEGG[111] as well as a version of pathway `REACT_152` from the REACTOME database[112]. This example is part of an example from the SBML Level 2 Version 4 specification.

XML schema documents

XML was described above as a way to encode textual information in a hierarchical manner with XHTML being one possible implementation of an XML file format.

It is relatively simple to test if a given file is a valid XML file because there is only a small set of rules that an XML file has to follow in order to be valid, e.g. each opening tag must have a corresponding closing tag and tags must be closed in the reverse order in which they were opened.

However, it is much harder to check if a given XML file is a valid instance of an XHTML file because the XHTML format poses a lot of additional restrictions on the format e.g. XHTML specifies a fixed set of tags that are allowed in an XHTML file and those tags are usually restricted in the order in which they may appear in the document.

In order to test if a given XML file is a valid instance of some given file format, e.g. XHTML, definitions for the additional restrictions of that file format are needed.

One way to specify such a set of restrictions is via an XML Schema document (XSD)[108]. An XML schema document is a standard released by the W3 consortium for defining XML document structures. An XML schema document is itself an XML file.

An XML schema for the XHTML format would for example define which tags are allowed in an XHTML file, in which order these tags may appear within an XHTML file, what attributes certain tags may contain and what values are allowed for these attributes etc.

Once such an XML schema document for a certain format has been created, it can be used to validate documents to find out if they adhere to the structure specified in the schema document.

Many XML reading libraries as well as the xmllint tool mentioned above can use XML schema documents to validate instances of XML based file formats.

Several XML schema documents are provided by the SBML community to validate SBML documents in different versions.

During this work XML Schema documents have been created to validate SBML documents with Layout and Render Extension information as well as to validate layout and render information in COPASI files.

Document Type Definition (DTD)

Document type definitions are the predecessor of the XML schema documents described above. DTDs have similar scope and capabilities as XML schema documents, but they differ in the details.

Since schema documents are easier to work with and provide some additional features over DTD documents, DTDs are not used very often today. Mostly structure definitions for older XML based file formats is still available as DTDs.

In this work, DTDs were initially used to validate SVG images created by the XSLT stylesheet. Later checking was done with RELAX NG (see below).

RELAX NG

RELAX NG[109] is similar in scope to the XML Schema documents mentioned above. With RELAX NG the structure of an XML document can be defined and used to validate instances of this document format.

RELAX NG has several advantages over XML Schema documents. For example, in RELAX NG certain things are easier to define than with XML schema documents and some things are possible with RELAX NG are not possible in XML schema documents at all.

Many XML document specifications provide RELAX NG documents for the validation of document instances.

The SBML community is currently working on a RELAX NG schema for SBML Level 3 Version 1 and COPASI also uses a RELAX NG document to validate COPASI's CPS file format.

Tools used to do the validation of XML documents against RELAX NG schemas were again `xmllint` as well as `jing`[115].

In this work, the XSLT style sheets written for the Layout and Render document specification have been validated against RELAX NG documents. Also SVG documents created with these style sheets have been validated against RELAX NG documents.

Rules for the validation of layout and render information in COPASI files have been added to the RELAX NG document for the COPASI file format.

Systems Biology Markup Language (SBML)

The Systems Biology Markup Language[40, 116] is an XML based file format for the storage and exchange of mathematical representations of biochemical

reaction pathways for time course simulations and other analysis methods in the field of systems biology.

The first version of SBML, termed SBML Level 1 Version 1 was created between 1999 and 2000 by a small group of researchers and the first official version was released in 2001[41]. Until then, each software tool in the field of systems biology used its own file format to store models and it was very hard for the users of these tools to exchange models between the different tools.

The format was received very well by developers in the systems biology community and support for reading and writing SBML files was soon incorporated into a number of tools. With the success of the file format, its shortcomings soon became evident and new improved versions of the standard have been created in a community effort. The latest version of the SBML specification is SBML Level 3 Version 1 from 2010. Today many tools in the field of systems biology contain support for reading and writing SBML files[117].

The core SBML structure consists of a single model definition which contains compartments, chemical species, parameters, rules and reactions. Rules and reactions can define mathematical expressions of how values of model elements, e.g. the concentration of a chemical species, change over time. This information can be used to build systems of differential equations to create time course simulations of those models. This core structure has more or less stayed the same for all versions of SBML with new versions adding new feature, elements or just clarifications concerning existing features to the specification. Newer versions of SBML for example allow the declaration of arbitrary units, function definitions as well as events to name just a few of the features added over time (see figure 2.3).

Most of the work described here is concerned directly or indirectly with SBML, its development, its extension and as well as its implementation in different software tools.

SBML is a big success in the systems biology community and part of this success is due to the excellent software support the SBML community has build around the SBML standard. An important first step in this direction was the implementation of a library called libsbml[70] for reading and writing of SBML documents and providing it for free to developers of systems biology software. libsbml relieves developers from writing their own routines for reading, checking and writing SBML documents and allows developers to concentrate on writing code that simulates or analyzes or displays these models. Having support for many of the popular programming languages like C, C++, Python, Perl, Java, etc, certainly helped as well.

```

Model
  Function Definitions
  Unit Definitions
  Compartments
  Species
  Parameters
  Initial Assignments
  Rules
  Constraints
  Reactions
  Events

```

Figure 2.3: Structure of an SBML document. The core features that existed since SBML Level 1 are marked in black, features added in later versions of SBML are marked in gray.

Scalable Vector Graphics (SVG)

The Scalable Vector Graphics format is an XML based standard for two dimensional vector graphics. It has been developed by the W3 consortium since 1999.

The file format allows the definition of complex drawings by combining elements from a large set of shape primitives like circles, rectangles, ellipses. In addition to defining primitives, the user can define colors and color gradients and apply those to the primitives (see figure 2.4).

The feature set of SVG is very large, making the format rather complex to implement. Out of this reason it took a very long time until there were software libraries and tools that could render SVG documents in a consistent way.

Today there are several implementations of libraries and programs able to render SVG images.

While most implementations widely differed in quality when this work started, most implementations today provide similar or identical results when rendering SVG images (see figure 2.5).

Programs used for the rendering of SVG images during this work include several web browsers like Mozilla Firefox[118], Apples Safari[119] or Opera[120] as well as special SVG rendering libraries and programs like batik[121], rsvg[122] or inkscape[123].

There are a number of reasons why several programs were used rather than a single one. For one, we needed to find out what the level of support for the SVG format was available in the different software tools. This allowed


```

<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg"
      width="410.0"
      height="410.0"
      version="1.1">
  <g>
    <defs>
      <radialGradient id="bw_gradient"
                    cx="50%" cy="50%"
                    r="50%">
        <stop offset="0%" stop-color="white" />
        <stop offset="100%" stop-color="black" />
      </radialGradient>
    </defs>
    <circle cx="205" cy="205" r="200"
           fill="url(#bw_gradient)" />
  </g>
</svg>

```

Figure 2.4: Example of a simple SVG file which defines a radial black and white gradient and applies this gradient as the fill style of a circle.

us to adjust the SVG generating XSLT style sheet to only produce output that could reproducibly be rendered by several different tools.

Also different tools handled errors in SVG files in different ways, so in order to find errors, the SVG images had to be rendered several times with different tools to check for differences which might point to errors in our XSLT style sheet.

GNU image manipulation program (Gimp)

Gimp[124] is a program for viewing and manipulating bitmap images. Newer versions also have limited support for reading SVG images.

Gimp was used in this work to view bitmap images created by the different software tools. This includes images created by rendering SVG images created with the XSLT stylesheet as well as images created with the different rendering implementations. This allowed us to test the correctness and quality of these implementations.

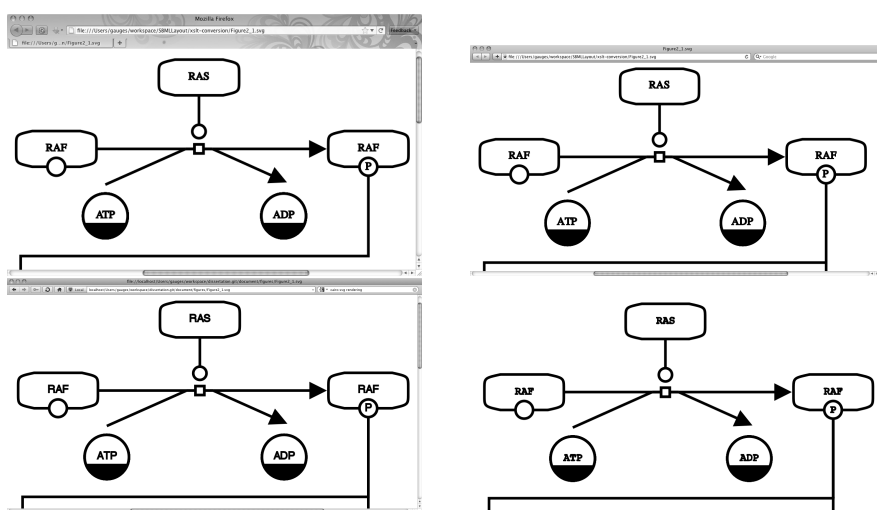


Figure 2.5: Example from figure 2.4 rendered in different programs (Top left: Mozilla Firefox 4.0, Top Right: Apple Safari Browser, Bottom Left: Opera Browser, Bottom right: Apache Batik toolkit). This is a very simple example, but nevertheless it demonstrates that the tools used to render it render it identically. Which means that SVG images can be used today to encode high quality vector images and get consistent rendering results from different tools.

Systems Biology Workbench (SBW)

The Systems Biology Workbench (SBW)[125] is a framework for the communication between systems biology tools.

In systems biology a large number of software tools has been implemented over the years and each of these tools has its field of expertise as well as its strength and weaknesses. Scientists therefore sometimes have to use a number of tools for the simulation and the analysis of biochemical reaction network models.

Normally this means that they have to create a model in one tool, save it to file and open it in another tool; something that has actually been made possible by the emerging standards in systems biology as for example the SBML file format (see above).

The Systems Biology Workbench tries to facilitate this exchange of models between tools by eliminating the file saving and loading steps, allowing programs to communicate directly, e.g. allowing one program to send a reaction network model to another program and receiving the results of an analysis or calculation back together with the potentially modified model.

To achieve this, SBW consists of a so called broker which handles the communication between the programs. This broker has to be installed by the user and software tools wanting to profit from SBWs exchange mechanism need to register with this broker. The broker keeps a list of registered programs and their capabilities and provides this list to other registered programs. This way each program registered with the broker knows what other programs are registered and can therefore send models to those programs if the user requests this. The models and messages are exchanged over a network based interface which means that it is theoretically possible to exchange models and data between different computers, e.g. a desktop and a powerful compute server. The internal format that is used to exchange models between the different tools is SBML.

Simplified Wrapper and Interface Generator (SWIG)

The Simplified Wrapper and Interface Generator (SWIG)[126] is a tool for (semi-)automatically providing access to program code written in low level languages like C or C++ to higher level languages like Java, Python or Perl.

As already mentioned in the sections on the different programming languages, high level scripting languages provide the users with powerful concepts and constructs which makes writing code often easier and allows programmers to get the same functionality with less lines of code. The disadvantage of this is that these scripting languages usually depend on some kind

of virtual machine or interpreter that has to convert the code into something the CPU understands. This conversion is costly and code written in high level languages is usually not as fast as the equivalent code written in a low level language.

Developers sometimes try to circumvent this problem by implementing the performance critical parts of their programs in some low level language that provides fast execution speed and implement the rest that is not critical for performance in the higher level language. This approach however makes it necessary that program parts written in the higher level language have access to the parts written in the lower level language. Since this is a very common task most higher level languages provide ways of creating this connection by writing some extra code. This extra code is usually written in the low level language using some API provided by the higher level language and the code to be written therefore usually depends on the type of higher level language used. This means that the extra code the developer has to write to connect a Java program to some low level C++ code is different from the code that is needed to connect a Python program to the same C++ code. Since it is often necessary to provide access to a library from not just on higher level language but from several a number of different wrapper implementations would have to be written. One for each of the higher level languages.

This is where the use of SWIG can be beneficial. SWIG provides a unified mechanism of generating the wrapper code for different high level programming languages. In the optimal case, the developer just points SWIG to the files that define the data structures and methods of the low level library and tells SWIG for which target language it should create wrapper code. SWIG will then try to automatically create all the code that is necessary to make the functionality of the low level library available to that target language. All the developer has to do is compile this auto-generated code and install it in the appropriate place.

Usually if the low level library is not very trivial, SWIG will not be able to generate the wrapper code fully automatically and correctly from the declarations provided by the low level library. A bit of extra information is usually needed to tell SWIG how to handle certain data structures for the individual target languages. This extra information is provided to SWIG in so called interface files.

Still, the work involved in generating wrapper code for any target language using SWIG is usually less than generating the wrapper code manually and it has the added benefit that wrapper code for many target languages can be generated from the information provided by a single set of interface files.

For a list of the target programming languages currently supported by

SWIG see table 2.1.

languages supported by SWIG					
AllegroCL	C#	CFFI	CHICKEN	CLISP	D
Go	Guile	Java	Lua	MzScheme	Ocaml
Octave	Perl	PHP	Python	R	Ruby
Tcl/Tk					

Table 2.1: List of target languages supported by SWIG. The target languages not used in this work are marked in gray, the ones that were used are marked black.

SWIG is used by libsbml to provide bindings to a large set of high level languages. The implementation of the Layout and Render Extension on top of libsbml therefore also implements some SWIG interface files to provide the functionality of the SBML Layout and Render Extensions to the Java and Python programming languages.

Also the COPASI language bindings for Java, Python, Perl, Octave and R use SWIG to generate the wrapper code from a single set of SWIG interface files.

SCons

SCons[127] is a Python based cross platform build system similar in scope to qmake (see above). The information about the elements of a project and additional information for compiling the elements is provided in the form of Python source code. The SCons program reads this information and runs the programs needed to create the project binaries from the provided source files.

With SCons building of projects on several platforms with one set of build information is possible.

The build system is very flexible, but compared to qmake, more information has to be provided in order to work for the different operating systems and development tools. The advantage of SCons lies in the fact that it doesn't depend on a full Qt installation.

SCons has been used as an alternative way for building some of the projects described in this work.

CMake

CMake[128] is another cross platform build system similar to qmake.

The build system is more flexible than qmake while providing a similar level of convenience. As an additional benefit, there are different graphical tools to use in conjunction with CMake to provide some of the information to the build system.

CMake has been used as an alternative way for building some of the projects described in this work.

Chapter 3

Systems Biology Markup Language Layout & Render Extension

3.1 SBML & Diagrams

With the Systems Biology Markup Language (SBML) there is a file format that allows users to store models and exchange those models with other users or transfer them from one application to another, provided that both applications support the SBML format. With over 100 application supposedly supporting SBML[129], this is quite likely.

The different modeling and simulation applications provide different means of creating and editing models. Some require the user to write models in the form of differential equations, others allow the user to directly specify chemical reactions together with kinetic equations and some tools even allow users to create models by graphically creating reaction graphs. In those reaction graphs, the chemical species are represented by the nodes and the reactions are specified as the edges of the graph.

Depending on the size of the model scientists have to invest a lot of effort to create a visually pleasing graphical representation of a model. The advantage of investing in this work is that the resulting graph can be used to get a comprehensive overview of the model as well as being able to use the graphical representation for displaying results of different analysis methods in an intuitive way.

Unfortunately the SBML file format is only suited to store a mathematical representation of a model. If a graphical modeling tool is used to build a model and the model is subsequently stored in the SBML format, all the

work invested in creating the graphical representation is lost or at least can't be transferred to other applications easily.

Not all applications dealing with reaction network models deal with them in the same way. Some applications merely allow the user to create or edit models, while other applications might only allow the user to do certain kinds of analysis on a model. Since the SBML format was supposed to serve as an exchange format between all these programs, the developers of SBML decided to limit the functionality of SBML to only allow the storage of the mathematical description of reaction network models.

Knowing that the core SBML specification would not be able to satisfy the needs of every piece of software, the developers of SBML added features to the format that would allow other developers to easily extend the format with new features.

The SBML format stores all information in the Extensible Markup Language (XML) format and the SBML specification defines an annotation tag that allows users to associate generic XML data with any element defined in the core SBML specification. This way the SBML format can be extended with arbitrary additional information.

The goal of the work described in this section was to use this extension mechanism of SBML to create a way to store graphical information together with the mathematical description of a model within an SBML file and link the graphical elements to their corresponding model elements. The result of this work is called the SBML layout and render extension.

3.2 Alternative Diagram Formats

At the time when this work started SBML was a very new format and it was only supported by a few programs. Back then most programs used their own proprietary format for storing models[29, 125, 31]. If those programs were storing graphical information with the model the format of that graphical information was specific to the needs of one program and therefore not useful for storing graphical information in a general way.

The other problem was that those file formats were hardly documented. Most of the time it was not even obvious what features and capabilities those formats provided. Popular examples for such programs using their own proprietary formats are JDesigner[125] by the group of Herbert Sauro or CellDesigner[31] by the group of Hiroaki Kitano. Both programs use their own format to store layout and render information and the formats are very specific to the rendering style of the two programs. Consequently there is no way to exchange the graphical information between the two programs. Since

the feature sets of the formats used by these two programs are different, it would be very hard if not impossible to write software that could convert the two formats.

In this context it is also important to mention the new Systems Biology Graphical Notation (SBGN)[42] standard. In contrast to SBML, SBGN is not a storage format for models, but rather a specification of how reaction network diagrams should be represented graphically (see figure 3.1). SBGN defines a set of graphical elements associated with a certain meaning and it specifies how these elements can be combined to create diagrams of reaction networks that are unambiguous. This approach can best be compared to electrical circuit diagrams in electrical engineering.

Although this is a related topic it does not pursue the same goal as the SBML layout and render extension. The SBML layout and render extension is about storing arbitrary kinds of diagrams within an SBML model file (see figure 3.1), potentially even diagrams not representing reaction networks, and it specifies a storage format with which this can be achieved.

SBGN on the other hand specifies how reaction network diagrams should be drawn, but it does not specify a storage format that would let the user store these diagrams to file. So SBGN is no replacement for the SBML layout and render extension, but the layout and render extension can be used to store diagrams as specified by SBGN (see figure 3.1).

There is some very recent work[130] that also specifies a file format for storing SBGN diagrams in files and even provides a library to help developers in supporting this format. But as mentioned above this is also no replacement for the SBML layout and render extension since it is limited in scope to just storing SBGN type diagrams.

Apart from diagram formats used by systems biology applications, there also exist some general graph and diagram storage formats that were considered for their suitability as a format to store diagrams in SBML files.

Two of the most popular formats that were considered as alternatives for storing diagrams were GraphML[132] and the Scalable Vector Graphics (SVG) format[133].

While GraphML is a very general graph storage format that also allows users to associate arbitrary data with nodes and edges, the format is tailored to storing graphs and not general diagrams and using it to store arbitrary diagrams would have lead to very abstract and hard to understand notations. Another issue with using GraphML would have been its somewhat restrictive license which would have hindered implementations in the non-academic field.

The SVG format on the other hand more or less contained all the features that were needed to store arbitrary diagrams combined with a more liberal license. With SVG the problem was rather that it was so complex and

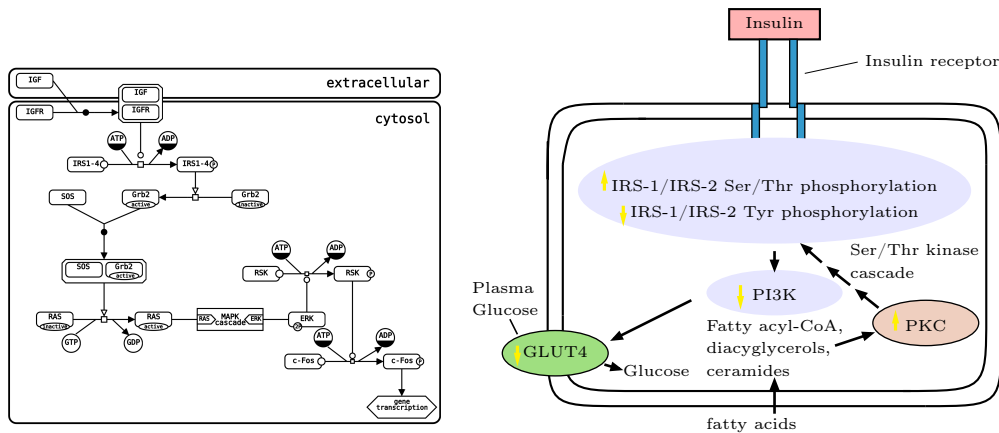


Figure 3.1: Example of two potential diagrams targeted by the SBML layout and render extension. (Left: reproduction of Figure A2 from the SBGN 1.0 specification, Right: reproduction of Figure 27-6 from Voet/Voet 3rd edition[131].)

powerful that implementations of it are very difficult.

SVG was developed as a standard to represent vector graphic diagrams on the web. Version 1.0 was released in 2001, but two years later, when work on the SBML layout and render extension was under way, there was still no implementation of SVG that would support most of its features. Depending on which implementation one chose, the results one got when rendering a SVG document were very different. This slow uptake of SVG was to a large part due to its complexity. Since there was no improvement of this situation foreseeable, using SVG as a way of representing diagrams in SBML was discarded.

3.3 Design & History

Since none of the existing formats were suitable for what the SBML layout and render extension was supposed to achieve, a new format had to be created.

Considering all the lessons learned from looking at existing software and formats, several design decisions were taken in the development of the SBML layout and render extension.

On the one hand, the extension was supposed to allow the user to store arbitrary diagrams which meant that it had to be very versatile. On the other hand the format was to be simple so that everybody could implement

it with as little effort as possible. As could be seen by the SVG format, these two goals are mutually exclusive. A lot of versatility usually leads to large complexity and having a format that is very simple takes away much of the versatility. It was clear that the SBML layout and render extension would have to make some compromises in order to achieve both of these goals.

Another design decision was to create a format that would allow diagram entities to be associated with the corresponding entities from the SBML model but at the same time be as independent of the underlying SBML format as possible. Having a diagram format that is independent of the model format and only connect to the model in a generic way would allow the use of the format within other model description frameworks/formats without major changes (see chapter 4.2).

Apart from its complexity, the SVG format would have provided everything that is needed for storing arbitrary graphical representation within SBML files. Therefore, as a first approach, the SBML layout and render extension was created by picking essential bits from the SVG specification that would allow users to store more or less arbitrary diagrams while leaving out all the parts that would make the format too difficult to implement. The parts that were picked for inclusion in the layout and render extension were further modified and extended to better meet the needs of the systems biology context in general and those of the enclosing SBML format in particular.

The first draft of the SBML layout and render extension was presented by Dr. Sven Sahle at the 8th SBML Forum meeting[134] in November of 2003. It was well received, but some developers still had the impression that the complete specification was too difficult to implement and it was therefore decided to split the layout and render extension in two parts. One part to describe the layout of the elements, meaning the size and positions of the diagram entities and one part to describe the style of elements, meaning the colors, shapes, fonts etc.

Due to the discussion at the SBML Forum meeting the proposal was split into the part that is now called the SBML layout extension and the part that is now called the SBML render extension. While the layout extension is independent of the render extension, the render extension depends on the information of the layout extension.

In the following sections the two parts will be described separately beginning with the SBML layout extension.

3.4 The SBML Layout Extension Specification

The SBML layout extension allows users to store layout information for diagram entities in an SBML file.

One of the design decisions for the SBML layout and render extension was to be able to associate layout information with the corresponding entities from the SBML model. The extension mechanism for SBML provides two options as to how this goal can be achieved.

The first option would be to store the layout information for a specific model entity together with that model entity in the XML file, this has the advantage that it is immediately clear which layout information belongs to which model entity, thus creating a connection of a diagram entity to the model entity via their co-location in the file. On the other hand, this also means that the layout information for an SBML model would be distributed over the complete model specification, making it more difficult to find specific information.

The second option would be to store the complete layout information in one single place, making it easy to find. The associations with the model entities would be achieved via references to their unique identifiers from within the layout information.

The second option provided a cleaner approach and it was decided to store the complete layout information as an annotation to the model element of the SBML document.

Each model should be able to store an arbitrary number of graphical representations. This way, each program can store its own graphical representation or the user can create different graphical representations for one model and store them all in the same file.

When creating the layout extension, it was decided to keep the representation of the layout information as close in design to the SBML format specification as possible. In SBML if a number of elements of the same type are stored, they are stored in an enclosing list element. So for the layout extension, the top level element is an element named *listOfLayouts* which can contain zero or more layout elements (see figure 3.2).

Each layout element is completely independent of the other layout elements and can be identified through an id attribute that has to be unique within the enclosing SBML model as well as within the list of layout elements.

Each layout element contains information about the size of the layout, as well as a tree like structure of elements representing the nodes and the edges. The structure of the layout information mirrors the structure of the model representation in the SBML document (figure 3.3).

In the SBML model, there is a list of compartments that defines the

```

<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1">
  <model id="Model_1" name="New Model">
    <annotation>
      <listOfLayouts xmlns="http://projects.eml.org/bcb/sbml/level2"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <layout id="layout_1">
          ...
        </layout>
        <layout id="another_layout">
          ...
        </layout>
      </listOfLayouts>
    </annotation>
    ...
  </model>
</sbml>

```

Figure 3.2: Example of an SBML XML document structure with layout information. The layout specific information is highlighted in gray.

compartments of the model, a list of species that defines the chemical entities participating in the models reactions and a list of reactions that specifies the reactions of the model.

In the layout we have a list of compartment glyphs that contains the graphical representations of compartments, a list of species glyphs that contains the graphical representations of the species and a list of reaction glyphs that contains the graphical representations of the reactions.

In addition to those elements, the layout contains a list of graphical objects and a list of text glyphs.

The graphical objects can be used to represent any object in the diagram that is not a representation for a compartment, species or reaction.

The text glyphs can be used to display textual information within a diagram. A common use case for a text glyph would be to store the name of a species as a label to its graphical representation.

Each of these lists contains zero or more elements of the corresponding type, e.g. the list of compartment glyphs can contain zero or more compartment glyphs. Likewise the list of reaction glyphs contains zero or more reaction glyphs.

Most glyph elements contain a number of attributes that define the prop-

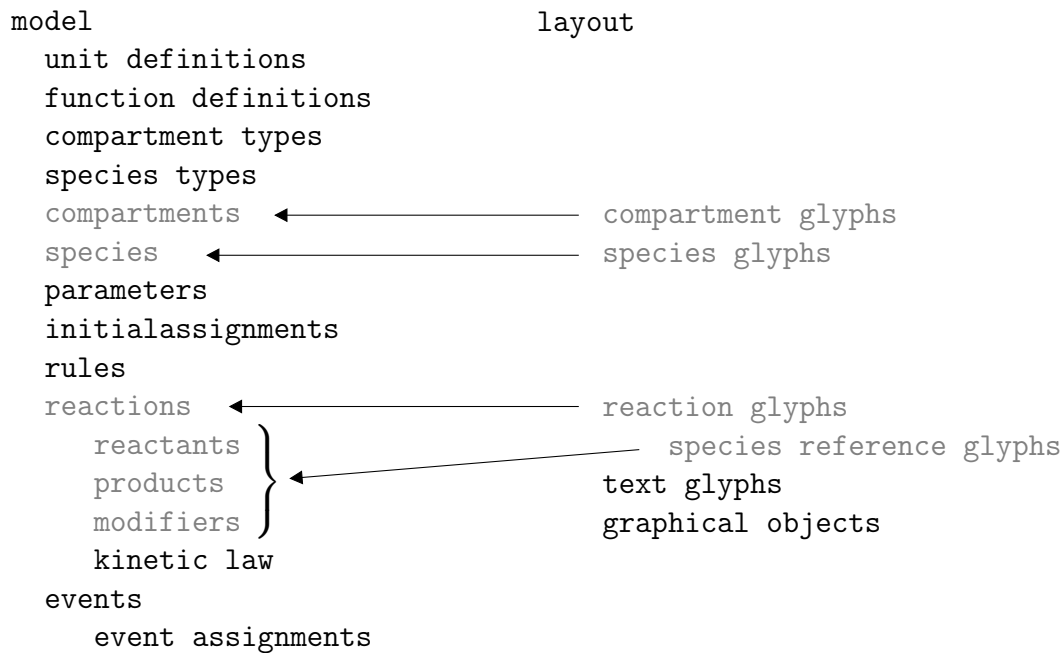


Figure 3.3: comparison of model and layout structure with model elements and corresponding layout elements highlighted in gray.

erties of the element like the size and the position. The reaction glyph element contains additional elements to define graphical representations which correspond to the species reference elements of the reaction.

The reaction element in SBML contains lists for the different metabolites. There is one list for those metabolites that are consumed (reactants), one for those that are produced (products) and one for those that influence the reaction in some other way (modifiers). Since a division into reactants, products and modifiers is not necessary with respect to layout, the reaction glyph in the layout extension only has a single list that contains all the graphical representations for reactant, product and modifier species references (see figure 3.3).

The way species reference glyphs are drawn in a reaction graph depends on the type of species reference that is represented by the glyph. For example most applications would draw the edge for a substrate different and the edge for an inhibitor in different ways. Some of this information can be deduced by looking at the species reference in the model and whether it was listed with the reactants, products or modifiers. But in the case of the modifiers it is not trivial to deduce if a modifier acts as e.g. an activator or an inhibitor. To make this type of information accessible to applications that render layout information, species reference glyphs can define a role attribute. The value

for the role of a species reference can be chosen from a list of predefined values (see table 3.1) and the application rendering the species reference glyph can determine the style of the glyph based on this role attribute.

undefined	substrate	product	sidesubstrate
sideproduct	modifier	activator	inhibitor

Table 3.1: possible values for the role attribute in species reference glyphs

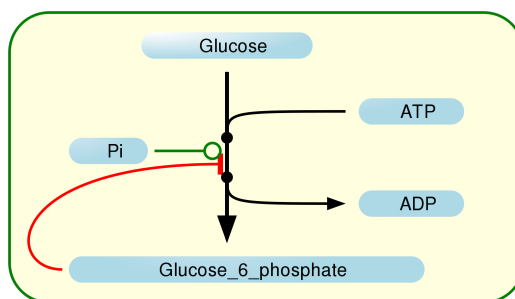


Figure 3.4: Example rendering of layout and render information using the role attribute on species reference glyphs.

Most model entities in an SBML model contain a unique identifier in the form of an *id* attribute. An notable exception to this is the species reference element which prior to SBML Level 2 Version 2 did not have an identifier. In order to be able to reference species reference elements from within layout information the layout extension augmented the SBML species reference element by an *id* attribute. This identifier has to be unique within the complete model. Because our work showed that this identifier on species reference elements was needed, it was added to the specification of SBML Level 2 Version 2.

The layout extension provides diagram information in the form of a graph consisting of nodes and edges. The information for the nodes is specified in the form of a bounding box (see figure 3.5) which provides information on its position and its size. Information for edges is usually provided in the form of curve objects.

Curve objects are made up of individual segments which can either be straight line segments (see figure 3.6) or cubic bézier segments (see figure 3.7). Straight line segments are defined by a start point and an end point, cubic bézier elements are defined by a start and an end point as well as two

base points that define the shape of the cubic bézier element. Each curve consists of one or more of those two element types and they can be mixed in arbitrary ways allowing for arbitrarily complex curve definitions (see figure 3.8).

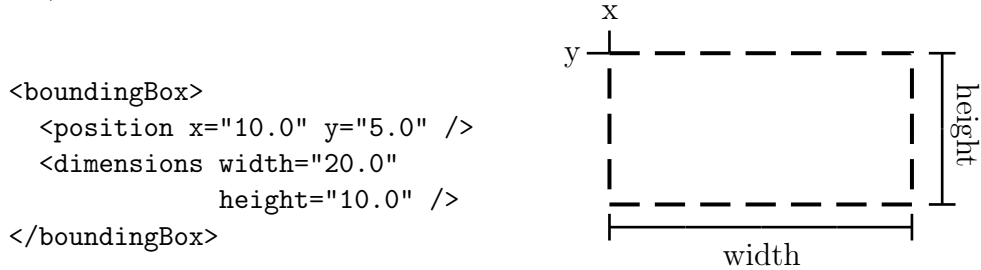


Figure 3.5: definition of a bounding box

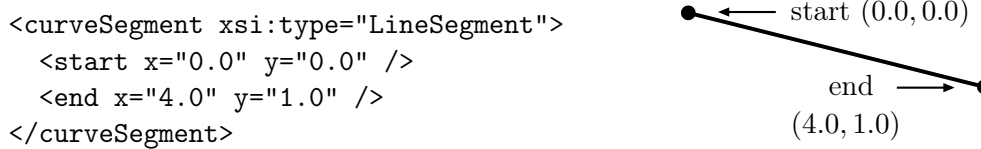


Figure 3.6: definition of a straight line segment

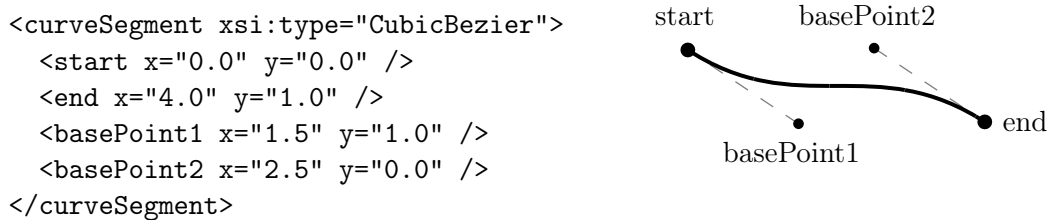


Figure 3.7: definition of a cubic bézier line segment

Currently most programs do diagram layout in two dimensions only, nevertheless all coordinate values for the layout elements can be specified as three-dimensional coordinates. E.g. a bounding box can take an optional *z* attribute and an optional *depth* attribute. Both attributes have default values of 0.0. If they are not specified, the bounding box describes a two dimensional element. This extra flexibility has two advantages:

- a) The specification is already prepared for three dimensional extensions should the need for such extensions arise


```

<curve>
  <listOfCurveSegments>
    <curveSegment xsi:type="LineSegment">
      <start x="0.0" y="0.0" />
      <end x="1.0" y="0.0" />
    </curveSegment>
    <curveSegment xsi:type="CubicBezier">
      <start x="1.0" y="0.0" />
      <end x="1.0" y="2.0" />
      <basePoint1 x="2.0" y="1.0" />
      <basePoint2 x="0.0" y="1.0" />
    </curveSegment>
    <curveSegment xsi:type="LineSegment">
      <start x="1.0" y="2.0" />
      <end x="2.0" y="2.0" />
    </curveSegment>
  </listOfCurveSegments>
</curve>

```

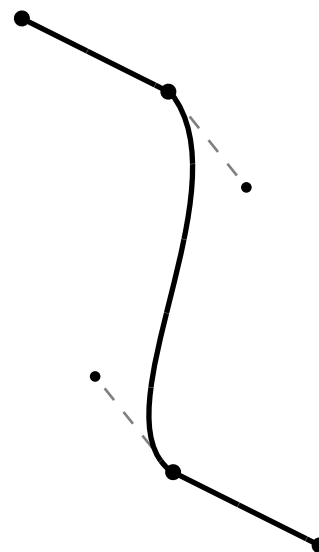


Figure 3.8: Definition of a curve with three segments.

- b) The depth information can be used to arrange the two dimensional layout into different layers to give programs a hint as to the order in which the layout elements should be rendered.

Since the SBML layout extension only stores information on the size and position of nodes as well as information on the curves making up the edges of a diagram, it is up to the program implementing the SBML layout extension what style it uses to draw the individual elements of the diagram. Back when this proposal was first presented this worked fairly well because the way in which the diagrams were drawn was more or less the same across most programs. Mostly the diagrams drawn by different programs would differ in colors or have slight variations in the shape of the nodes (see figure 3.9).

With the appearance of more complex types of diagrams, e.g. SBGN diagrams or diagrams commonly found in text books (see figure 3.1), the SBML layout extension alone was no longer able to provide all features necessary to store these diagrams. For this, the SBML render extension, which is described later in this work, was developed.

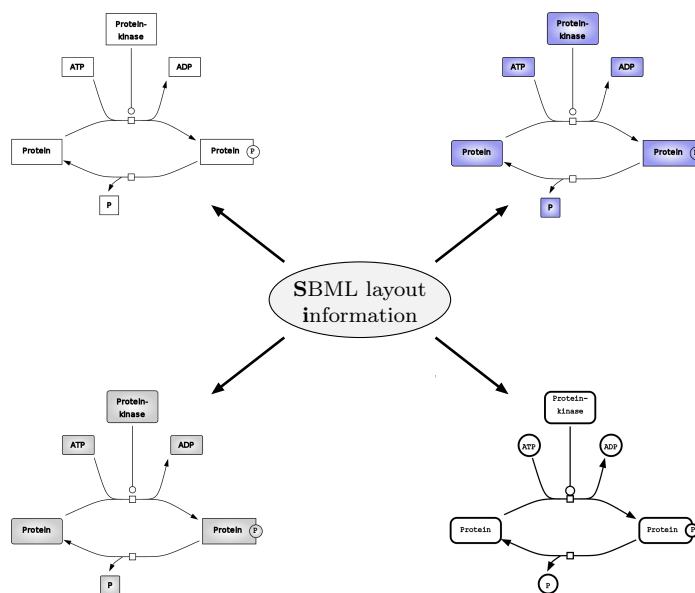


Figure 3.9: The same layout information rendered in four different ways.

3.5 Implementation Of The Layout Extension

The SBML community has created a software library called libsbml[70] which provides developers of software tools with an easy way to read and write SBML model files and check those models for consistency and conformance to the SBML specification.

Since the SBML specification is fairly complex and checking for syntactic as well as semantic error within an SBML model is no trivial task, having this library greatly reduces the work necessary for implementing support for the SBML standard in software tools. And while in the beginning many tools were creating invalid SBML due to missing possibilities to check the models that were created, there now is a mature infrastructure based on libsbml which can be used for testing and therefore finding and eliminating incorrect SBML files.

A lot of developers have recognized the advantages of using such a stable and feature rich library to facilitate their work and many software projects therefore use libsbml for the implementation of SBML support.

Libsbml Implementation

The success of SBML has demonstrated that it is important to have a standard, but even better to have a library that helps developers in implementing that standard. To provide the same level of support for the SBML layout extension, an implementation of the layout extension as part of libsbml has been written. This way developers already using libsbml are provided with an easy way to read layout information stored as SBML layout extension data in SBML files.

The first implementation was finished in April of 2004 and released as a patch against the sources of libsbml version 2.0.3.

To use it, developers need to download the patch as well as the source code for libsbml, apply the patch and compile the patched source code. Just as the specification of the SBML layout extension followed the SBML specification in style, the implementation of the layout extension closely followed the rules and standards set by libsbml. This way developers already familiar with using libsbml could apply their knowledge directly to the implementation of the layout extension. Figure 3.10 shows the inheritance tree of the SBML layout classes and how the individual classes of the layout implementation extension are used by other classes.

The implementation provides language bindings for the C, C++, Python and Java programming languages allowing programmers who are using any of those languages to read and write layout information according to the SBML layout extension.

Documentation was provided in the form of application programming interface (API) documentation as well as examples for the different programming languages.

The layout extension continued to be developed as a patch against the different version of libsbml and new versions of the patch were released on a regular basis, following changes and developments in libsbml.

Starting with libsbml 2.3.0 the code for the support of the layout extension was integrated into the main libsbml code and is now included in each release of libsbml. In the beginning the extension was marked as experimental and the developer had to enable the feature explicitly if they wanted to use it. Later releases of libsbml marked it as stable and enabled it per default. By now all precompiled versions of libsbml are distributed with layout extension support enabled.

The libsbml code is very stable and mature and to ensure the quality of the code most features are tested with a comprehensive set of unit tests. The unit testing framework that is used in libsbml is called check[135].

To achieve a high level of quality for the implementation of the layout

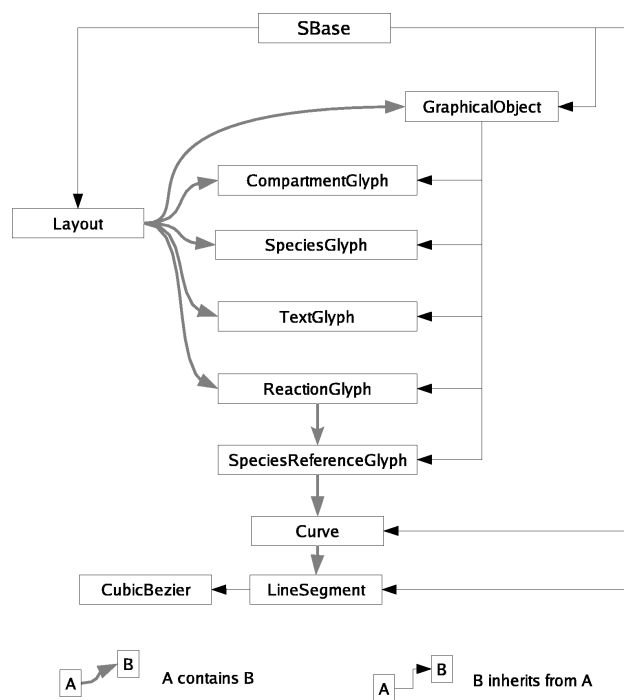


Figure 3.10: Inheritance and usage diagram for SBML layout classes.

extension a set of 254 unit tests covering all aspects of the layout extension have been written. These tests can now be run prior to a release to make sure that neither changes to the code of the core of libsbml nor changes to the code for the extension did break existing functionality.

Implementation Of Layout Rendering

Since one of the goal of the layout and render extension is to provide the user with the possibility to store high quality diagrams in SBML models, e.g. for publications, reading and writing the layout information is not enough. In addition to an implementation for reading and writing SBML layout extension data in libsbml there are several implementations for creating images from this layout information.

The first implementation that I created is based on XSL Transformations (XSLT)[136]. In XSLT a set of transformation rules, a so called XSLT stylesheet, is defined that specifies how a program, called XSLT processor, transforms input in XML format to textual output. The output data itself can also be in XML format, but it doesn't have to be. In this case the XSLT stylesheet defines a set of transformations that specify how an SBML file with layout information according to the SBML layout extension is converted to a Scalable Vector Graphics (SVG) diagram[133].

The identifier of the layout and some additional parameters that define the style of the final rendering can be given as arguments to the XSLT processor. In a second step the resulting SVG diagram can be converted to a bitmap image of arbitrary size. This procedure is ideally suited for making high quality images for publications.

When this implementation was first completed in 2005 there was no full implementation of the SVG standard and consequently it was difficult to convert the SVG file to the corresponding bitmap image. The results would usually depend on the program that was used to do the conversion. Today support for SVG has improved and there are a number of reliable implementations for rendering bitmap images from SVG diagrams including libsvg[122], cairo[137] or batik[121]. In addition to those software libraries, most browsers[119, 118, 120] can render SVG files as well as some image manipulation programs like gimp[124]. The quality and consistency of bitmaps rendered by any of these programs is usually very high.

In order to test this implementation, a set of SBML files with layout information has been created which tries to cover as many aspects of the conversion and rendering process as possible. These test files were especially important when I later extended the XSLT stylesheet with new features (see 3.6) because they could be used to assure that the new functionality did not

interfere with the existing functionality.

SBML Level 3 & Standardization

SBML was intentionally limited to a feature set that provide the means to exchange models of reaction networks between programs with an emphasis of using differential equations to describe the models. This can be seen as a least common denominator that can be understood by most programs in that field. However, more and more programs start to be limited by the narrow scope of SBML. For example the information needed to describe spatial models can not be stored using the core SBML functionality which makes it unsuitable for these types of models.

Due to this a number of extension to SBML have been proposed[138], similarly to what the described layout extension does but for different purposes, e.g. extensions for spatial simulation[139] or for model composition[140]. Since the number of proposals is growing continuously, the SBML community has come up with a set of procedures that has to be followed before an extension proposal can become a recommended SBML extension or an SBML package which is the term for SBML extensions to SBML Level 3.

In order to become a recommended SBML extension there has to be an official proposal that can be reviewed. This proposal has to be implemented at least twice independently and finally there has to be a vote by the members of the SBML community on whether a proposal will become an officially recommended extension or not[141].

This is a very lengthy process and until today the layout extension proposal is the only proposal that has made it through the first of the two stages.[142]

3.6 The SBML Render Extension

Overview & Design

In recent years diagrams for reaction networks have become more complex and the individual nodes in the diagram contain more information which is conveyed by the way the nodes are drawn. A very prominent example for this trend is the SBGN standard[42].

With the diagrams encoding parts of the information about the process in the style of the individual nodes and edges, it is no longer enough to be able to specify the positions and the sizes of the elements and leave choosing the style to the program. In order to store the complete information present

in a diagram, programs need to be able to specify exactly how to draw a certain layout element. For this purpose the SBML render extension has been developed.

While the layout information was more or less independent of the underlying model information, the information provided by the render extension is only meaningful together with layout information as specified by the SBML layout extension.

The layout extension provides information on the position and size of the individual diagram elements while the render extension provides information about how these elements are represented on screen. Since the layout information is able to provide information in the form of curves for e.g. the species reference glyphs, there is a small overlap in information between the layout and the render extension. This means that sometimes there is more than one way to specify information for the edges of a diagram.

Since implementing the layout extension is simpler than implementing both the layout and render extension, it might be preferable to provide as much information in the layout part as possible and to only supplement this information with render information where there is no way to specify the same information in the context of the layout.

Since the layout extension tries to stay as close to the SBML specification in style as possible, the SBML layout extension contains the same extension mechanism as SBML. This means that the layout extension itself can be extended the same way the SBML specification has been extended by e.g. the layout information. This mechanism was used to supplement the SBML layout extension with style information for the layout elements.

The same design decision that were used for the layout extension were also applied to the SBML render extension. The render extension is supposed to be easy to implement while providing a certain level of flexibility.

As with the layout extension, it was decided to keep all the render information in one place rather than distribute it over all the individual layout elements.

Render Extension Specification

Global And Local Render Information

Two different types of render information have been specified which are attached in two different places within the layout extension (see figure 3.11). The so called global render information is attached to the list of all layouts and can be applied to any of the layouts in that list. As a consequence, global render information is normally defined in a very unspecific way, e.g. all nodes

and edges are assigned similar styles. This is similar to what programs have been doing prior to the release of the render extension.

```

...
<listOfLayouts xmlns="http://projects.embl.org/bcb/sbml/level2"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <annotation>
    <listOfGlobalRenderInformation>
      <renderInformation id="global_render_info_1">
        ...
      </renderInformation>
    </listOfGlobalRenderInformation>
  </annotation>
  <layout id="layout_1">
    <annotation>
      <listOfRenderInformation xmlns="http://projects.embl.org/bcb/sbml/render/level2"
                              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <renderInformation id="local_render_info_1">
          ...
        </renderInformation>
      </listOfRenderInformation>
    </annotation>
  </layout>
</listOfLayouts>
...

```

Figure 3.11: Example of how render information is embedded within the layout information. The layout information is printed in gray while the render information is printed in black. The global render information appears towards the top and the local render information towards the bottom.

Since the global render information should be applicable to any layout, it is not possible to associate a render style to specific layout elements. For this the render extension defines the so called local render information. Local render information is attached to a specific layout and can associate a style to elements within that layout via the identifiers of those elements.

The render extension allows the user to derive new render information based on existing render information, similar to an inheritance schema found in most object oriented programming languages. This way it is very easy to create new render information that only deviates in a few small details from an existing render information.

Save for the fact that local render information can reference specific layout

elements via their identifiers and global render information can't, the overall structure of the two render information types are more or less identical. The structures and principles in the following sections therefore apply to global as well as local render information. If there are differences between the two render information structures they are specifically highlighted.

Colors

Render information is divided into four sections. The first section consists of an optional list of color definitions. Colors can be specified in different ways in the render extension. A color can either be specified as an RGB value where R stands for the amount of the red color component, G stands for the amount of green color component and B stands for the amount of blue color component. Each color component can be in the range of 0..255 and is specified as a hexadecimal value. Alternatively a color can be specified as an RGBA value. This is identical to the RGB value only that there is an additional alpha component A. This alpha component specifies the opacity of the color. A value of 0 means that the color is completely transparent whereas a value of 255 means that the color is completely opaque. This method of specifying colors is identical to the way colors are specified in e.g. HTML documents or SVG diagrams (see figure 3.12).

RGB Value	RGBA Value
#RRGGBB	#RRGGBBAA
e.g. #FF0000	e.g. #00FF00A0
full red 100% opaque	full green ~ 60% opaque

The value for each of the three or four channels (RR=red, GG=green, BB=blue and AA=alpha) has to be given as a hexadecimal number in the range of 0x0 to 0xFF this is the decimal equivalent of the range from 0 to 255.

Figure 3.12: specifying RGB and RGBA values

An additional way to specify a color for an element in the SBML render extension is through a unique identifier of a color definition. Using names instead of numerical values for colors often has the advantage of making the render information more consistent and easier to understand.

An example of two color definitions is given in figure 3.13.

```

<listOfColorDefinitions>
  <colorDefinition id="darkred" value="#200000" />
  <colorDefinition id="transp_blue" value="#00008080" />
</listOfColorDefinitions>

```

Figure 3.13: List with two color definitions. One defines a dark red color as an RGB value that is completely opaque the other specifies a partially transparent blue color value through an RGBA value.

Gradients

The second section in a render information element represents a list of gradient definitions that can be used in the style section (see below) to define the fill style of two dimensional objects.

The gradient definitions are restricted to linear and radial gradients because these are the most popular gradient types being used in reaction network diagrams. Adding more gradient types would be possible if needed, but each new gradient type increases the complexity of a potential implementation, so it was decided to limit the first version of the render extension to the two mentioned gradient types.

Each gradient object has to have an identifier that is unique within a render information object and the gradient can later be referenced via this identifier.

The definition of a gradient object can consist of an arbitrary number of so called gradient stops, making the gradient definitions very versatile.

Examples of a linear gradient and a radial gradient are depicted in figures 3.14 and 3.15.

Instead of specifying colors as RGB values as done in these examples, colors could also be specified as identifiers for color definitions as described in the preceding section.

```

<linearGradient id="diagonal_bw"
  x1="0%" y1="0%"
  x2="100%" y2="100%">
  <stop offset="0%" stop-color="#FF0000" />
  <stop offset="50%" stop-color="#00FF00" />
  <stop offset="100%" stop-color="#0000FF" />
</linearGradient>

```

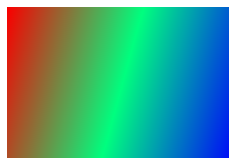


Figure 3.14: Example of a linear gradient definition (left) that runs diagonally from red over green to blue and a rendering of the gradient applied to a rectangle (right).

```

<radialGradient id="diagonal_bw"
  cx="50%" cy="50%"
  r="50%">
  <stop offset="0%" stop-color="#FFFFFF" />
  <stop offset="100%" stop-color="#000000" />
</radialGradient>

```

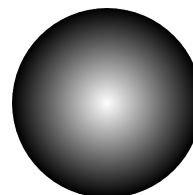


Figure 3.15: Example of a radial gradient definition (left) that runs from white to black and a rendering of the gradient applied to a circle (right).

Line Endings

The third section in a render information object defines a list of line endings. This is a list of definitions for arrow heads that can be used to describe how the start and/or end of a curve object should be decorated. Again each line ending definition has to have a unique identifier that can be used to reference the line ending within a style.

This feature will be discussed in more detail in the section on curves below.

Styles

The first three sections consist of definitions that can be used in the fourth section which defines the list of styles that can be applied to layout elements. Each object that is defined in sections one to three has to have an identifier that is unique within the render information so that it can be used to reference the corresponding definition from within a style definition in section four.

Each style that is defined in section four has to specify two things:

- a) which layout elements it applies to
- b) how these layout elements are to be rendered

There are three different ways how a style can specify the layout elements it can be applied to.

The least specific way to specify that a style can be applied to a certain layout element is through the type. E.g. a style can specify that it can be applied to compartment glyphs which have an associated type called COMPARTMENTGLYPH. Each of the different glyph types in the layout extension has a corresponding type name in the render extension (see Table 3.2).

A style can apply to zero or more of the types defined above. The ANY type has the same meaning as listing all the other type names together and it means that the style can be applied to any layout object.

layout object	layout object type
compartment glyph	COMPARTMENTGLYPH
species glyph	SPECIESGLYPH
reaction glyph	REACTIONGLYPH
species reference glyph	SPECIESREFERENCEGLYPH
text glyph	TEXTGLYPH
graphical object	GRAPHICALOBJECT
any layout object	ANY

Table 3.2: Type names for the different layout elements.

An example of how a style can be associated to a layout element via the type is shown in figure 3.16.

```

<compartmentGlyph id="cg1"
  render::objectRole="cytosol">
  ...
</compartmentGlyph>
...
<speciesGlyph id="sg1"
  render::objectRole="important">
  ...
</speciesGlyph>
...
<style id="style_1" typeList="COMPARTMENTGLYPH SPECIESGLYPH">
  <g stroke="#0000FF" stroke-width="2.0" >
    <rectangle x="0.0" y="0.0" width="30.0" height="15.0" />
  </g>
</style>
...

```

Figure 3.16: Example of a style that defines its targets via types. The style defines that it can be applied to compartment glyphs and/or species glyphs.

A more specific way of associating a style with an object is via the role of the layout element. Each layout element has an optional *objectRole* attribute and this role can be used to describe that a certain style can be applied to all layout objects with the given role. Again a style object can specify that it applies to more than one role.

The principle of associating a style with layout elements via the role of

the layout elements is demonstrated in figure 3.17.

```

<compartmentGlyph id="cg1"
  render::objectRole="cytosol">
  ...
</compartmentGlyph>
...
<speciesGlyph id="sg1"
  render::objectRole="important">
  ...
</speciesGlyph>
...
<style id="style_1" roleList="important">
  <g stroke="#0000FF" stroke-width="2.0" >
    <rectangle x="0.0" y="0.0" width="30.0" height="15.0" />
  </g>
</style>
<style id="style_1" roleList="cytosol">
  <g stroke="#00FF00" stroke-width="2.0" >
    <rectangle x="0.0" y="0.0" width="130.0" height="200.0" />
  </g>
</style>
...

```

Figure 3.17: Example of styles that define their targets via roles. One style defines that it is applicable to all layout objects with role `cytosol` and the other is applicable to all layout objects with role `important`.

The most specific way of associating a style with a certain layout object is via the identifier of the layout object. As mentioned above, this method of associating a style with a layout object is only available within local render information elements.

Just as with the "association by type" and the "association by role", the style can specify that it applies to more than one identifier.

An example of association via identifier is depicted in figure 3.18.

Also combinations of associations of styles to layout objects via type, role or id are possible.

This association schema can sometimes lead to conflicts, e.g. if one style would apply to an object via the objects type and another style would apply based on the identifier of the object. To resolve such conflicts, the render extension defines a set of resolution rules.

```

<compartmentGlyph id="cg1"
  render::objectRole="cytosol">
  ...
</compartmentGlyph>
...
<speciesGlyph id="sg1"
  render::objectRole="important">
  ...
</speciesGlyph>
...
<style id="style_1" idList="sg1">
  <g stroke="#0000FF" stroke-width="2.0" >
    <rectangle x="0.0" y="0.0" width="30.0" height="15.0" />
  </g>
</style>
<style id="style_1" idList="cg1">
  <g stroke="#00FF00" stroke-width="2.0" >
    <rectangle x="0.0" y="0.0" width="130.0" height="200.0" />
  </g>
</style>
...

```

Figure 3.18: Example of local styles that define their targets via identifiers. One style is defined to apply to the layout object with identifier `sg1` and the other to the layout object with identifier `cg1`.

Generally a more specific association type takes precedence over a less specific association. If there are several possible associations within a certain render information element that have the same precedence, e.g. two styles that are associated with text glyphs via the TEXTGLYPH type, the first one that is encountered in the style list takes precedence.

Association Precedence

TYPE < ROLE < ID

The detailed set of rules for style resolution are given in the specification of the render extension.

Primitives

To define how layout elements are to be rendered when a certain style is applied to them, the render extension defines a number of graphical primitives which can be combined in arbitrary ways to create complex graphical elements.

The primitives defined in the current version of the SBML render extension are curves, polygons, rectangles, ellipses, text elements and bitmap images (see figure 3.19). To allow hierarchical arrangement of several primitives the group element has been introduced.

A style contains one or more attributes that specify which layout elements the style can be applied to as well as a single group element that contains the specification of how the layout elements are to be drawn (see figures 3.16, 3.17 or 3.18).

In the following sections each of the individual primitives available in the current version of the render extension is described in more detail.

Curves

The curve element is a one dimensional object and it therefore only has attributes to define the stroke color and the stroke width. Optionally a line stippling pattern can be specified to create stippled curves.

For examples on the use of the different ways a curve can be rendered based on these attributes see figure 3.21.

The curve definition in figure 3.20 essentially defines the same curve object as was defined in figure 3.6, but instead of using the layout notation, it uses the render notation. The notation for curve objects in the render extension differs slightly from the definition of curves in the layout extension for a good reason.

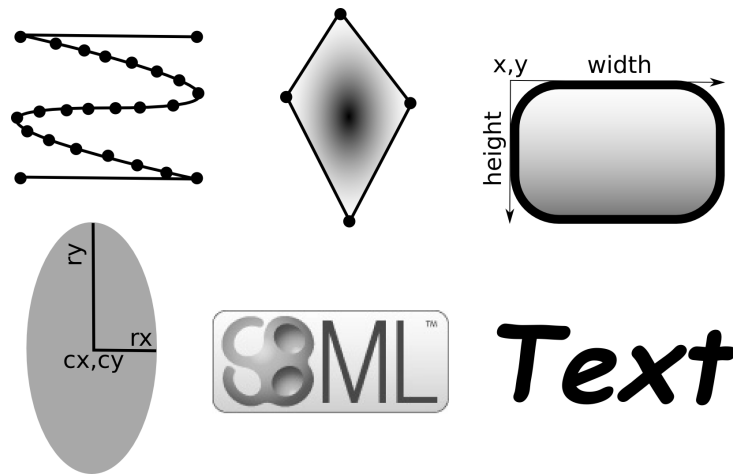


Figure 3.19: Primitives available in the SBML render extension (curve, polygon, rectangle, ellipse, image & text)

```

<curve stroke-width="2.0" stroke="#0000A0">
  <listOfElements>
    <element xsi:type="RenderPoint" x="0.0" y="0.0" />
    <element xsi:type="RenderPoint" x="1.0" y="0.0" />
    <element xsi:type="RenderCubicBezier" x="1.0" y="2.0"
      basepoint1_x="2.0" basepoint1_y="1.0"
      basepoint2_x="0.0" basepoint2_y="2.0" />
    <element xsi:type="RenderPoint" x="2.0" y="2.0" />
  </listOfElements>
</curve>

```

Figure 3.20: Curve example with a straight line element followed by a cubic bézier element followed by a straight line element.

In the layout extension stippled lines are created by defining a number of line segments with gaps. The gaps are created by defining the start point of line segments n to be different from the end of the preceding line segments $n - 1$. In the render extension, stippled lines can be created by setting a line stipple pattern attribute on a curve. In addition to being able to specify line stipple patterns on curves, the render extension also allows the user to specify the same line stipple pattern for the outlines of certain primitive types (polygon, rectangle and ellipse). This way we can use one line stippling specification mechanism for all the primitives in the render extension instead of having one mechanism for curves and another one for the other primitives. If the curves had the possibility to create stipple patterns in the same way this is done in the layout extension in addition to the line stipple attribute, implementers of the render extension would have to deal with all possible combinations of these two features, which makes the specification as well as implementations unnecessarily complicated. As a consequence the way curves are defined in the render extension has been changed in a way that prohibits the creation of stippled lines via the curve segments. As a side effect, the curve specification has become more concise.

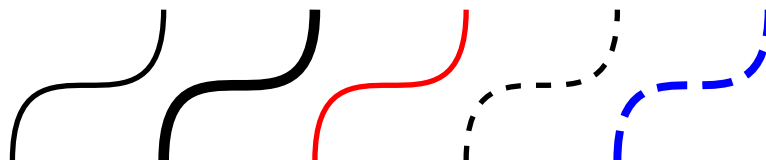


Figure 3.21: The stroke for curves (and the stroke for the outline of two dimensional objects) can vary in width, color and stippling pattern or a combination of these attributes.

Line Endings

Curves in reaction network diagrams are often used to represent edges and consequently certain relations between entities in the diagram. In order to specify what type of relation is represented by an edge, one or both ends of the edge are decorated with certain symbols, e.g. arrow heads. Since this feature is used very often in reaction network diagrams, the render extension supports this feature in a very flexible manner.

The SBML render extension allows users to define a number of line endings (see figure 3.22) that can be applied to either end of a curve element.

The definition of lines endings is similar to that of styles described above. Each line ending defines a group element that combines an arbitrary number of graphical primitives to form a graphical representation of a line decoration. Each line ending element also has a unique identifier and a curve can reference this identifier to determine what decoration will be applied to the start or the end of that curve (see figure 3.23).

In addition to the unique identifier and the graphical representation, a line ending element needs to specify certain attributes that determine how the line ending is applied to the curve. In order to be able to place line decorations correctly the line ending has to specify a bounding box for its size and an offset from the curve end. The user also has the possibility to specify if the line ending should be rotated according to the slope of the curve or not. The effects of this feature are demonstrated in figure 3.24.

```
<listOfLineEndings>
  <lineEnding id="simpleHead_red"
    enableRotationalMapping="true">
    <boundingBox>
      <position x="-8" y="-3"/>
      <dimensions width="10" height="6"/>
    </boundingBox>
    <g stroke="red" stroke-width="1.0" fill="#000000">
      <polygon>
        <listOfElements>
          <element xsi:type="RenderPoint" x="0.0" y="0.0" />
          <element xsi:type="RenderPoint" x="10.0" y="3.0" />
          <element xsi:type="RenderPoint" x="0.0" y="6.0" />
        </listOfElements>
      </polygon>
    </g>
  </lineEnding>
</listOfLineEndings>
```

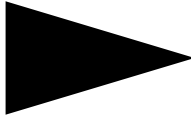


Figure 3.22: Definition of a simple red arrow head with rotational mapping enabled.

Polygons

The polygon object is very similar to curves in the render extension (see figure 3.25). The major differences are that the curve segments in the polygon definition define the outline of the polygon and that the last point in the definition of the outline of the polygon is implicitly connected to the first point. This ensures that the polygon always defines a closed two dimensional area. The polygon has the same attributes as the curve to define properties of the polygons outline.

```

... <curve stroke="#000000"
      startHead="simpleHead_red">
  <listOfElements>
    <element xsi:type="RenderPoint" x="0.0" y="0.0" />
    <element xsi:type="RenderPoint" x="20.0" y="20.0" />
  </listOfElements>
</curve>
<curve stroke="#000000"
      endHead="simpleHead_red">
  <listOfElements>
    <element xsi:type="RenderPoint" x="20.0" y="20.0" />
    <element xsi:type="RenderPoint" x="40.0" y="40.0" />
  </listOfElements>
</curve>
<curve stroke="#000000"
      startHead="simpleHead_red" endHead="simpleHead_red">
  <listOfElements>
    <element xsi:type="RenderPoint" x="40.0" y="40.0" />
    <element xsi:type="RenderPoint" x="60.0" y="60.0" />
  </listOfElements>
</curve>
...

```

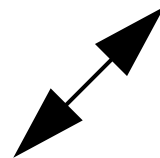
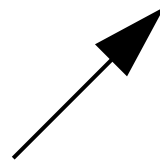
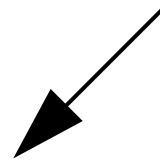


Figure 3.23: The line ending defined in figure 3.22 applied to three curve objects. Once applied only to the start, once to the end and once to start as well as end of the curve.

```

<listOfLineEndings>
  ...
  <lineEnding id="simpleHead_red2"
    enableRotationalMapping="false">
    <boundingBox>
      <position x="-8" y="-3"/>
      <dimensions width="10" height="6"/>
    </boundingBox>
    <g stroke="red" stroke-width="1.0" fill="#000000">
      <polygon>
        <listOfElements>
          <element xsi:type="RenderPoint" x="0.0" y="0.0" />
          <element xsi:type="RenderPoint" x="10.0" y="3.0" />
          <element xsi:type="RenderPoint" x="0.0" y="6.0" />
        </listOfElements>
      </polygon>
    </g>
  </lineEnding>
</listOfLineEndings>

```

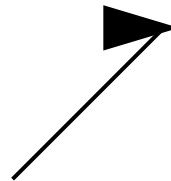
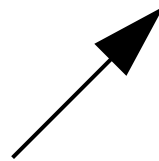


Figure 3.24: Difference between having a line ending adjusted to the slope of the curve and not having it adjusted. The line ending definition is identical to the one in figure 3.22, just the attribute for the rotational adjustment has been set to false. The result can be seen on the right. At the top, application of the previous line ending is depicted, at the bottom application of this new line ending definition is shown.

In addition to the attributes for the outline the polygon has some attributes to define how the area is to be rendered. This allows for unfilled polygons as well as polygons filled with a single color or a gradient.

An example of some of the possibilities how the rendering of two dimensional objects can be influenced by these attributes is given in figure 3.26.

```
<polygon stroke-width="2.0" stroke="#000000"
  fill="#707070">
  <listOfElements>
    <element xsi:type="RenderPoint" x="1.0" y="0.0" />
    <element xsi:type="RenderPoint" x="2.0" y="2.0" />
    <element xsi:type="RenderPoint" x="0.0" y="2.0" />
  </listOfElements>
</polygon>
```

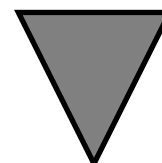


Figure 3.25: polygon example defining a gray triangle with black outline of width 2

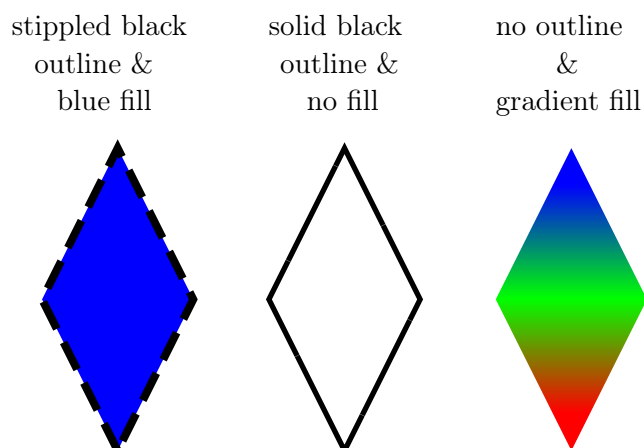


Figure 3.26: Polygons with different fill and outline attributes. These attributes can also be applied to other two dimensional objects (ellipses & rectangles).

Rectangles

Although the polygon class can in principle be used to create any two dimensional object, the render extension specifies separate primitive types for the definition of rectangles and ellipses since those two element types are commonly used in reaction network diagrams. This adds only a little bit of

additional complexity to the render extension but makes defining those commonly used primitives considerably easier, especially with respect to ellipses.

The rectangle primitive is defined via a position in the form of x and y attributes as well as *width* and *height* attributes that define its size (see figure 3.27). There are also attributes to define rounded corners for the rectangle (see figure 3.28). Since a rectangle is a two dimensional object it has the same attributes to define the outline and area fill properties as described for the polygon primitive above.

```
<rectangle x="0.0" y="0.0" width="20.0" height="20.0"
  stroke="#000000" stroke-width="1.0"
  fill="none"/>
```



Figure 3.27: Definition of a square with solid black outline and no fill.

```
<rectangle x="0.0" y="0.0"
  width="20.0" height="20.0"
  rx="5.0" ry="5.0"
  stroke="none" fill="#A0A0A0"/>
```



Figure 3.28: Definition of a gray square without outline and rounded corners.

Ellipses (& Circles)

The ellipse element can be used to define ellipses as well as circles (see figure 3.29 & 3.30). The most general form, the ellipse, is defined via a center point and radii along the x and the y axis. If the radii for the x and the y axis have the same value, the resulting shape is a circle. Since an ellipse is a two dimensional shape, it also has all the attributes for defining the properties for the outline and the area fill.

```
<ellipse cx="50.0" cy="75.0" rx="75.0" ry="50.0"
  stroke="#000000"
  fill="none"/>
```

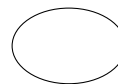


Figure 3.29: Definition of an ellipse with a solid outline and no fill.

Text Elements

Text elements are used to add text to the rendering of a layout element. The text element has attributes to define a font and a number of font properties like the font size and the font style, the vertical and horizontal text alignment.

```
<ellipse cx="25.0" cy="25.0" rx="25.0"
stroke="#000000" stroke-width="2.0"
fill="#707070"/>
```



Figure 3.30: Definition of a gray circle with black outline of width 2.0.

The color of text can be specified the same way as the color for the outline of other elements, but the attributes that define the stroke width or the line stippling have no effect on text elements.

A few other features commonly found in the context of text elements, like specifying a fill color or placing text along a curve, that are found in more general rendering frameworks like SVG have not been added to the render extension because they add a lot of complexity with only little benefit for the user.

An example for the definition of a text element can be seen in figure 3.31.

```
<text x="10.0" y="10.0"
stroke="#0000FF"
text-anchor="start"
vtext-anchor="top"
font-family="monospace"
font-weight="bold"
font-size="18.0">"Monospace"</text>
```

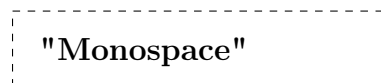


Figure 3.31: Example of defining a text element with a certain font, font style and alignment. The left and the top of the text are aligned with the upper left corner of the associated bounding box (dashed line) and then the text is moved 10 units along the positive x axis and 10 units along the positive y axis.

Bitmaps

The last graphical primitive the current render extension offers is the image element.

The image element can be used to include bitmap objects to a render extension style. The image element defines the position and the size of the bitmap and specifies the location and name of a JPEG or PNG file (see figure 3.32).

So rather than embedding the bitmap information within the render extension data, it has to be loaded when the layout and render information is used to draw the diagram.

In order to make an implementation easier the supported file formats were limited to JPEG and PNG, if the need should arise, the specification of the render extension can easily be extended to support other file formats.

If the actual size of the bitmap image is not the same as the size given in the declaration of the corresponding image element, the image has to be scaled according to the width and height specified for the image element.

If the file referenced in the image element can not be found, the application can decide if it draws nothing at all or if it draws a certain placeholder, e.g. an error message text that notifies the user that the image could not be rendered.

```
<image x="30.0" y="45.0"
width="100.0" height="100.0"
href="glucose.png" />
```

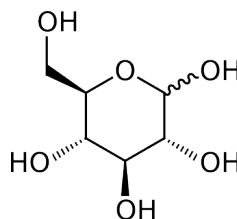


Figure 3.32: Example how to include bitmaps in a render style. (Bitmap was taken from Wikipedia[143] article on glucose.)

Grouping

Using a single primitive in a style only allows the specification of relatively simple diagrams. In order to create more complex diagrams the simple primitives discussed above need to be combined. To combine graphical primitives the render extension uses group elements.

A group element can contain one or more children which can either be graphical primitives or further groups elements (see figure 3.33). This enables the creation of arbitrarily complex nested style definitions.

The group element itself can define properties for lines, area fill and text attributes. These properties are applied to all children unless they are redefined in those children. This makes the definition of styles easier because e.g. the color and width for the outline of objects can be defined once at the top level group element and be left undefined in the individual children as long as they don't need to change. This reduces the resulting file size and makes the render information more readable.

Relative Versus Absolute Coordinates

To make the render extension even more versatile coordinates for primitives, gradient definitions, line endings and style definitions can either be given in absolute coordinates, relative coordinates or a combination of relative and


```

<g stroke="#000000" stroke-width="1.0"
  font-family="serif"
  font-size="18.0">
  <rectangle x="0.0" y="0.0"
    width="100.0" height="120.0"
    fill="#E6E6E6" />
  <image x="0.0" y="0.0"
    width="100.0" height="100.0"
    href="glucose.png" />
  <text x="10.0" y="-5.0"
    vtext-anchor="bottom"
    font-weight="bold"
    >"Glucose"</text>
</g>

```

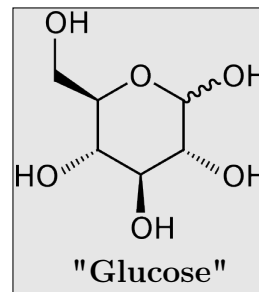


Figure 3.33: Example of how primitives can be grouped to define more complex styles. In this example a style for a glucose node is defined that draws a bitmap of glucose on a light gray rectangle and adds some descriptive text. The top level group element defines some attributes that are inherited by all children, unless they are redefined in the children.

absolute coordinate values. Relative coordinates are always in relation to the bounding box of the layout object the style is applied to.

For example this way a style can be defined that draws a rectangle starting at 10% of a layout objects width and 10% of the objects height and that covers 80% of the width and 80% of the height, effectively creating a 10% border on each side independent of the size of the layout object. Specifying coordinates in styles as relative values also allows the creation of styles that scale well with the size of the bounding boxes of layout objects (see figure 3.35).

Absolute coordinates are given as floating point numbers, e.g. 23.5 while relative coordinates have a % sign attached to the number, e.g. 10%. When specifying a combination of relative and absolute values, the absolute value is given first followed by a + symbol and the relative value (see figure 3.34). Fixing the order of components for absolute/relative coordinate combinations facilitates parsing coordinate values and therefore makes the implementation of the render extension easier without sacrificing versatility.

Transformations

Another feature that adds a lot of flexibility to the render extension is the possibility to add transformations to primitives and groups.

Normally rectangles, ellipses, images and text elements are drawn so that the width or in the case of the ellipse the x radius is parallel to the x axis of the coordinate system. If objects are needed that do not align with the

absolute value relative value

$$23.5 + 20.0\%$$

Figure 3.34: Schema for writing coordinates in the SBML render extension.

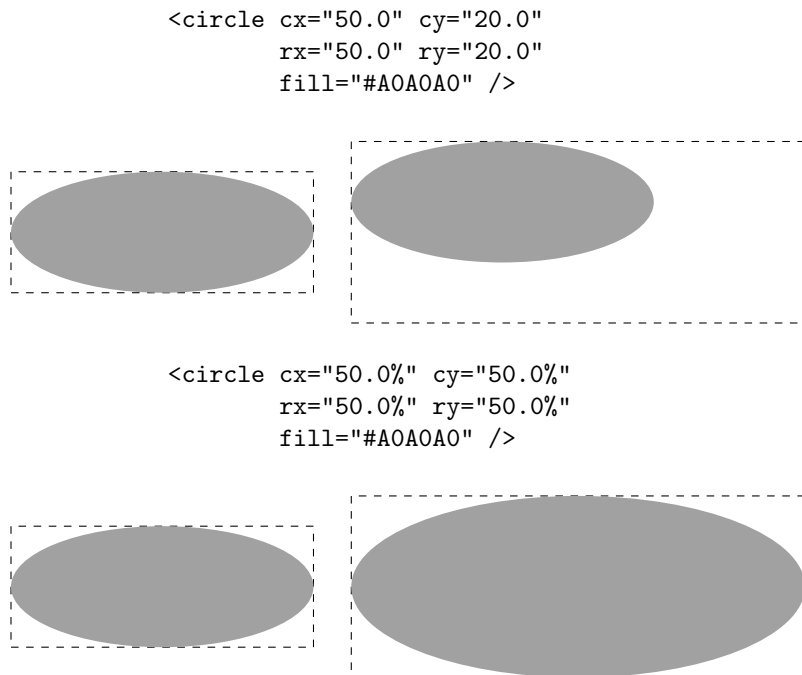


Figure 3.35: Comparison of relative coordinates versus absolute coordinates when defining an ellipse and applying it to different bounding boxes. The comparison shows that the ellipse with the relative coordinates scales better with differing bounding box dimensions. (top: ellipse with absolute coordinates applied to a box of size 100x20 and to a box sized 150x30, bottom: ellipse defined with relative coordinates applied to the same boxes as above)

x and y axis of the coordinate system, the possibility to specify an angle of rotation around any of the axes is required.

To enable this general transformations in the form of affine transformation matrices can be specified with each render object. With such a general transformation matrix not only rotation, but also translation, scaling and skewing as well as combinations of those transformations on render objects are possible (see figure 3.36).





<pre><rectangle x="0.0" y="0.0" width="10.0" height="10.0" /></pre>	
<pre><rectangle x="5.0" y="12.0" width="10.0" height="10.0" transform="0.707, 0.707, -0.707, 0.707.0, 0.0, 0.0" /></pre>	
<pre><rectangle x="0.0" y="24.0" width="10.0" height="10.0" transform="0.5, 0.0, 0.0, 1.5, 0.0, 0.0" /></pre>	
<pre><rectangle x="5.0" y="36.0" width="10.0" height="10.0" transform="0.3535, 1.0605, -0.3535, 1.0605, 0.0, 0.0" /></pre>	

Figure 3.36: Transformation example with an untransformed, a rotated, a scaled as well as a scaled and rotated square.

Libsbml Implementation

Just like the first versions of the implementation for the layout extension, the implementation for the render extension was done as a patch against the sources of libsbml. With each new release of libsbml a new patch adding the functionality to read and write render information in SBML files is released. The first released implementation was in October 2009 against the sources of libsbml 3.4.1. The patches are updated for newer releases of libsbml and there are patches for libsbml 4.1 as well as libsbml 4.2. Shortly a new major version of libsbml, libsbml 5, will be released making a partial rewrite of the render extension patch necessary.

As with the implementation for the layout extension, the render extension has been implemented for several programming languages (C++, Python and

Java). Support for the C programming language has not been implemented yet since it does not seem to be used by many software projects any more.

A diagram of all the classes of the render extension implementation for libsbml is depicted in figure 3.37.

To ensure the quality of the implementation 126 unit tests for the C++ implementation have been written as well as 83 and 62 for the Python and the Java implementations respectively. These tests can be used prior to making a release for checking if any changes in libsbml affect the implementation of the render extension.

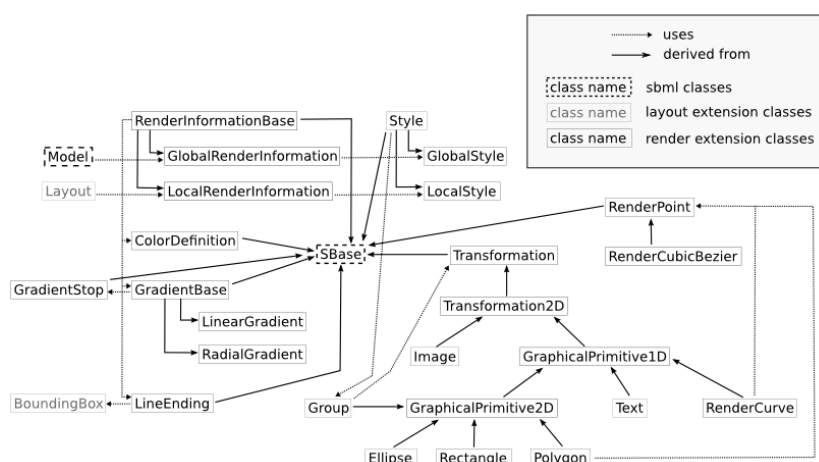


Figure 3.37: Inheritance and usage diagram for render extension classes.

Rendering Implementations

XSLT Style Sheet

The implementation of the reading and writing capabilities on top of libsbml is only one aspect of software support for the SBML render extension. In order to be really useful, the information also has to be rendered. For this the XSLT stylesheet, that has already been discussed in the section on the layout extension, has been extended to enable the interpretation of render information and its conversion to SVG constructs.

In addition to taking the identifier of the layout to be rendered, the extended XSLT stylesheet can now take the identifier of a render information element that is to be applied to the layout. If no identifier for a render information element is given, the XSLT stylesheet falls back to the original behavior of drawing just the layout by applying a default style.

If a render information identifier is specified, it has to belong either to a global render information element or to a local render information element that is attached to the layout specified by the layout identifier.

If both a valid layout and a valid render information object are found, the render information is applied to the layout according to the rules explained above and the result is a SVG drawing. There exists a number of high quality software implementations like batik[121] or cairo[137] that can convert the SVG graphics into a bitmap drawing of arbitrary size.

Since the render extension is a lot more complex than the layout extension, the extension to the XSLT stylesheet are quite extensive which makes testing even more important. For this a suite of test files that covers as much of the specification as possible has been created.

Currently this test suite consists of more than 100 test files. The tests are implemented as a set of C++ source code files that use the layout and render implementation in libsbml to create SBML files with layout and render information. The layout and render information systematically covers the individual elements of the render extension, from the rendering of individual graphical primitives with different combinations of outline and fill properties to complex combinations of primitives. Since the storage of layout and render information in SBML Level 2 files differs slightly from the way this information will likely be stored in SBML Level 3 files, the test files cover both versions of the SBML specification.

Implementing the tests using the libsbml implementation has the added advantage that it serves as an additional test for the libsbml based implementation.

Since this test suite might also be interesting and helpful to other developers implementing the layout and render extension, all test cases have been released on our web page together with the implementations[144]. All the files are distributed under the liberal Lesser GNU Public License (LGPL).

OpenGL Based Rendering Library

The XSLT style sheet allows users to convert layout and render information to bitmap images using an XSLT processor and some SVG rendering software. However it would be beneficial for software developers if there wasn't just a library for reading and writing the layout and render extension, as provided by the extensions to libsbml described above, but also a library for rendering layout and render information from within software tools.

To this end, a library was implemented that takes layout and render information and renders it using the OpenGL standard graphics API.

There are several graphical standards that would have been suitable to

implement the SBML render extension. OpenGL 1.3 was chosen because it is a standard that is available on all platforms as opposed to e.g. DirectX which is only available on Microsoft Windows. In addition to being cross platform, OpenGL 1.3 is one of the first version of OpenGL which was released in 2001, so the chances of having a driver that fully supports this standard are very high on all platforms independent of the graphics hardware that is being used. Due to these reasons, the implementation is likely to work on the vast majority of computers in use today.

The implementation was written in C++ and due to time constraints, no implementation for Python, Java or any other programming language have been created so far. But since the OpenGL API is available for most programming languages, there is no technical reason why such an implementation can't be written.

Again this implementation is not trivial due to the flexibility of the SBML render extension and being able to test it is very important. Since there already is a test suite that was created for testing the XSLT implementation, we also use that test suite to test the OpenGL implementation. Testing was mainly done by comparing the bitmap renderings created from an SVG drawing that was created by the XSLT stylesheet to the corresponding image created by the OpenGL implementation.

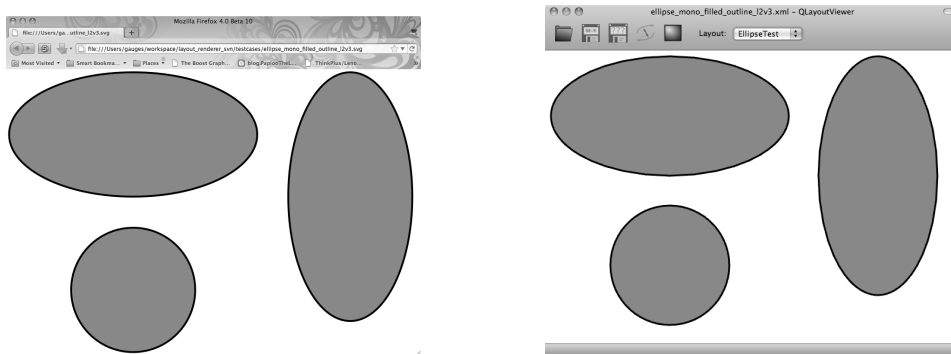


Figure 3.38: One of the render extension tests rendered in two different applications. Left: firefox rendering the SVG file created with the XSLT style sheet. Right: Demo application rendering the layout information from the SBML file for the test using the OpenGL based rendering library.

To demonstrate the usefulness of the library a small example program was written (see figure 3.38 (right)) using the Qt framework[100]. The application can read SBML files with layout and render information, display the layout with applied render information and create bitmaps of those renderings. The

program runs on Microsoft Windows, Mac OS X as well as Linux. Binaries for the different platforms as well as the source code for the library and the example program have been released for use by interested developers[144].

3.7 Third Party Implementations

So far only the implementations that were written in our group have been discussed, but by now there are several implementations of the SBML layout and/or render extension by other groups. This is very important if the layout and render extensions are to become official SBML extensions since the SBML standardization process requires at least two independent implementations.

Libsbml 5

SBML Level 3 is will support extensions of the standard with so called packages[145]. The only difference between an extension in an annotation and an official SBML Level 3 extension package is that the later is not enclosed by the <annotation> tags and that is has gone through all the stages of the standardization process as described on the SBML web page[141]. In order for this extension mechanism to work, support for reading and writing them has to be implemented in libsbml. While currently libsbml version 4.2 is the latest stable version, work on its successor libsbml 5 has been ongoing for several years.

To test the package extension mechanism, the SBML layout extension has been implemented in as a package in libsbml 5 by Akiya Jouraku, a former core developer of libsbml.

So when libsbml 5 will finally come out in a stable version, it will already have support for reading and writing the SBML layout extension as a package in SBML Level 3 files.

CellDesigner Plugin

In 2005 Yasunori Osana presented an implementation[146] that enables the CellDesigner software to convert their internal diagram format to SBML files with layout information. Because this implementation is limited to the layout extension, not all of the information present in CellDesigner diagrams can be converted. Since the styles of the nodes in CellDesigner are often essential to the understanding of the diagram, this is a major drawback. Although implementing support for the SBML layout and render extension has been on

the agenda of the CellDesigner developers for quite a while[147], no such implementation is currently in sight. A solution to this problem might actually be provided very soon by the program that I have written for the conversion of CellDesigner diagrams to SBML layout and render information (see 3.8).

SBW Modules

Developers from the group of Herbert Sauro have written several implementations[148, 149] of the layout and render extensions in the context of the Systems Biology Workbench (SBW) framework. They have written modules that are able to create view and modify layout and render information according to the SBML layout and render extension. These modules can be used as standalone programs or be accessed by other programs through SBWs communication framework.

The developers of the group also wrote a .NET library for working with SBML layout and render information[150]. This implementation is also using the XSLT stylesheet presented above to create high quality SVG and subsequently bitmap images from SBML files with layout and render information.

Latex Converter

Only recently the group has published another paper[151] about a library that is able to convert SBML layout and render information into L^AT_EX-Tikz[152, 153] code for the inclusion in publications written in L^AT_EX.

Arcadia Software

Another program that implements the SBML layout extension is Arcadia[154]. Arcadia is a visualization tool for metabolic pathways. It takes an SBML file without layout information and creates layout information based on the reactions defined in the SBML model. The resulting layout can then be stored in the SBML model and written to file.

3.8 The SBML Layout And Render Extension In NF- κ B Modeling

As described in the introduction, we are participating in the Virtual Liver Network[] and in this context we are collaborating with an experimental research group in Heidelberg in order to find the interaction points between the NF- κ B signaling pathway and the Hippo signaling pathway. For this,

models for the individual pathways have to be build and eventually combined. As a first step, one of our collaborators build an initial model for the NF- κ B signaling pathway.

In order to build such a model, the knowledge about the biological processes involved in this signaling pathway has to be encoded in a way that can be processed by a computer program. Since we also want to simulate the model eventually, one of the standards mentioned in the introduction would provide the means to do this. Because we do have extensive experience with SBML and since this standard seems to provide the broadest range of software support, SBML was chosen for the storage of the model.

As our collaboration partners know more about the biological processes then we currently do, they set out to create an initial topological model describing the known interactions between some of the elements involved in the NF- κ B pathway. The most intuitive way for biochemists to encode such interactions is by drawing the corresponding reaction network. The CellDesigner[31] software is ideally suited for that task. The software uses specific symbols for certain concepts often found in biochemical networks and the diagrams created with CellDesigner are similar to process diagrams as described by the SBGN specification. This is no coincidence as the CellDesigner notation was used as the basis for the creation of the SBGN process diagram notation. The possibility to create models of a reaction networks graphically is especially targeted at biologists and biochemists because it allows them to encode their knowledge in a form that they are familiar with because the diagrams look similar to diagrams commonly found in biological publications and text books. The information about the reaction network as described by the diagram is then converted to the SBML format without any prior knowledge from the user.

The initial representation of the NF- κ B model created in CellDesigner by Frederico Pinna is depicted in figure 3.39.

As a modeling tool CellDesigner specializes in the creation of reaction networks and there is not much functionality beyond that. There are some extensions to CellDesigner by third party developers that allow users of CellDesigner to add reaction kinetics to a topological model[155] or to do some time course simulations, for example through the COPASI language bindings as described in chapter 4.5. But as soon as it more sophisticated types of analysis on a model are required, other tools have to be used.

One tool that provides users with a number of different methods for the analysis of such models is COPASI. COPASI is developed in our group in collaboration with research groups from the University of Manchester in the U.K. as well as from the Virginia Biotech Institute in Blacksburg, Virginia, in the U.S.A.

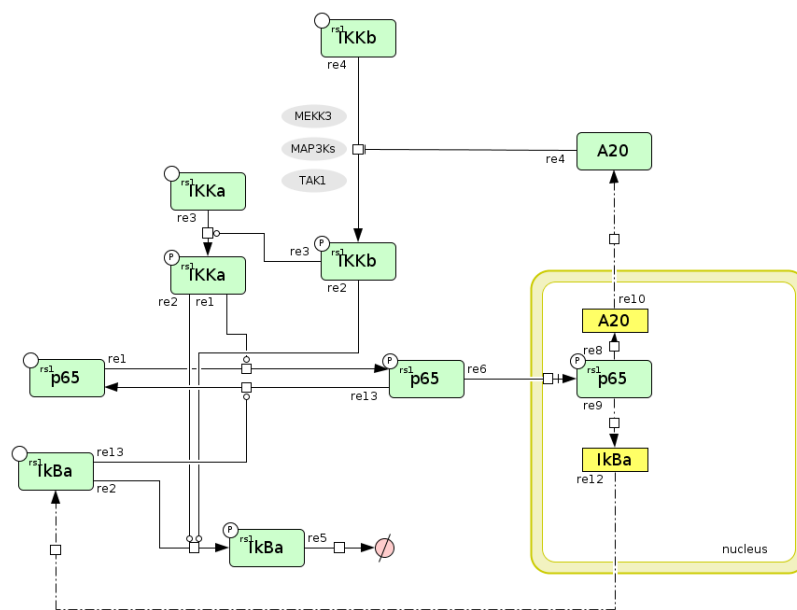


Figure 3.39: CellDesigner diagram of a NF- κ B activation model created by Federico Pinna. This is used in one of the sub-projects of the Virtual Liver Network as a starting point to find the interactions between the NF- κ B signaling pathway and the Hippo signaling pathway.

COPASI provides excellent support for the SBML document format due to the work described in chapter 4.1. Since CellDesigner also uses SBML documents to store reaction network models, seamless data exchange between the two programs with respect to the mathematical model is possible. This is something that was not possible prior to the work described in this thesis. Back then, the user would have had to recreate the model when switching from one tool to another.

Unfortunately the diagram description created with CellDesigner can not be stored using the core functionality of SBML, so the only data that can be exchanged using core functionality of SBML is the mathematical description of the reaction network. This was one of the reasons why we developed the extension to SBML that is described in chapter 3.4 and 3.6. With the help of these extensions it is theoretically possible to exchange the graphical information together with the mathematical description.

However, so far the developers of CellDesigner have not been able to implement support for these extensions, but rather use their own proprietary format to store the graphical information. This means that when we started this project, the graphical information created by Frederica Pinna could not be transferred from CellDesigner to COPASI.

Since it would be nice to have the graphical version of the network available in COPASI, e.g. to display analysis results graphically as described in chapters 4.3 and 4.4, we implemented a tool that converts the graphical information provided by CellDesigner to the graphical extensions for SBML and COPASI that are described in chapters 3.4,3.6 and 4.2.

Because the CellDesigner format is not well documented, this work is not trivial and much of the informations gathered about the format have been gained by trial and error as well as by analyzing how certain changes to a diagram in CellDesigner influence the resulting diagram notation when it is stored in the file.

As we have only used the latest version of CellDesigner for this work, support for converting CellDesigner diagrams is limited to that version and only those parts of the diagram notation that are relevant to this collaboration are supported so far. Files written with older versions of CellDesigner have to be loaded into the latest version and saved again in order to be converted. Most features commonly used in reaction network diagrams are supported already, but a lot of small details still have to be dealt with as can be seen in figure 3.40.

The implementation of this new feature has taken about one week so far and the current results are already very promising. With the help of the converted diagrams, it is already possible to display analysis for e.g. the elementary flux mode analysis graphically as demonstrated in chapter 4.6.

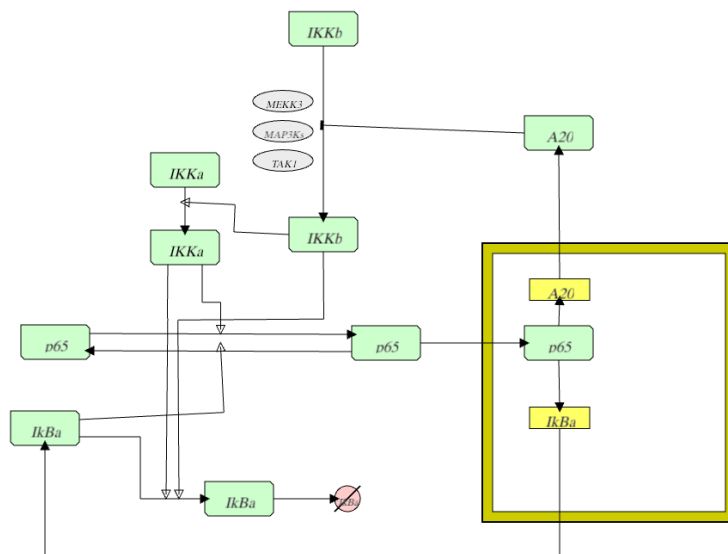


Figure 3.40: Same diagram as presented in figure 3.39, only this time it is displayed in COPASI after the CellDesigner specific layout information has been converted to the SBML layout & render extension format (see chapters 3.4 and 3.6).

This conversion will also be extended and improved as this collaboration continues and hopefully this will eventually lead to a tool that is also useful to other researchers in the field of systems biology.

Chapter 4

Standards In COPASI

4.1 SBML Support In COPASI

COMplex PATHway SIMulator (COPASI)[69, 156] is a software package for the creation, editing, simulation and analysis of reaction network models. The program is the successor of the well known Gepasi[30, 29] software by Pedro Mendes.

COPASI is a joint development by the groups of Pedro Mendes at Virginia Tech University (VBI) in Blacksburg, Virginia and the Manchester Interdisciplinary Biocentre (MIB) in the United Kingdom as well as the group of Ursula Kummer at the University of Heidelberg.

While its predecessor Gepasi was only available for the Microsoft Windows platform[67], COPASI has been developed to work on all popular platforms including GNU/Linux[60], Microsoft Windows as well as Mac OS X[64]. In addition to being available in binary form for the afore mentioned platforms, the source code for COPASI is freely available and can be used to create binaries for other platforms, e.g. Solaris[56] or any of the BSD[59] distributions.

To achieve this level of platform independence, a number of cross-platform standards are used in the development of COPASI. First of all, COPASI is written using the standardized programming language C++[157] for which a large numbers of compilers for the different platforms are available. In addition to using a standardized programming language, COPASI uses the platform independent Qt GUI toolkit for the development of the user interface. This allows for a consistent user experience across all supported platforms.

The first official release of COPASI was done in June of 2006 with unofficial releases dating back to the end of 2004[156].

Standards were not only important in choosing appropriate software libraries for the development of COPASI, but from the beginning COPASI provided users with the possibility to load and store models using the SBML file format, the de facto standard for the storage of reaction network models in systems biology. In addition to SBML, COPASI can read and write reaction network models in several different formats. Normally COPASI uses its own proprietary file format to read and write models, but for backward compatibility to its predecessor it can also read models in Gepasis file format. Writing Gepasis files however is not supported by COPASI. For writing models to file, COPASI supports several additional formats, e.g. COPASI can write models in a format suitable for the Berkeley Madonna Simulation Software[158] or the popular free simulation tool XPP-Auth[159]. Additionally models can be written as source code for the C programming language.

Arguably the most important format that is supported by COPASI is the SBML standard and this section will deal predominantly with the implementation of the SBML format in the context of COPASI.

COPASI has been developed since the year 2000 and the first version of SBML (SBML Level 1 Version 1) has been released in March of 2001[40].

SBML Level 1 Version 1 was very limited in how users could specify reaction network models which lead to the development of a successor that was supposed to eliminate most of these shortcomings. SBML Level 1 Version 2 was released August of 2003 and it fixed some of the problems perceived with SBML Level 1 Version 1, but some of the old problems were also still present. In parallel to SBML Level 1 Version 2, a new major version of SBML, SBML Level 2 Version 1 was developed and released in June of 2003. This new major version introduced new concepts, e.g. events and arbitrary function definitions for use in kinetic laws. The first official release of COPASI could read models in SBML Level 1 Version 1, SBML Level 1 Version 2 or SBML Level 2 Version 1, which were all the SBML versions released at that time.

Export of models was constrained to SBML Level 2 Version 1 for the first releases of COPASI because at that time SBML Level 1 has been obsoleted by SBML Level 2. The further use of SBML Level 1 was strongly discouraged by the SBML editors.

Later releases of COPASI followed the development process of SBML very closely which means that COPASI was usually among the first programs to support new versions of the SBML standard.

Current versions of COPASI provide support for reading and writing a large number of different SBML levels and versions (see table 4.1).

COPASI basically supports all available version of SBML for import and export available today, the only exception being export to SBML Level 1

	import	export
SBML Level 1 Version 1	✓	X
SBML Level 1 Version 2	✓	✓
SBML Level 2 Version 1	✓	✓
SBML Level 2 Version 2	✓	✓
SBML Level 2 Version 3	✓	✓
SBML Level 2 Version 4	✓	✓
SBML Level 3 Version 1	✓	✓

Table 4.1: Supported version of SBML for importing and exporting reaction network models in COPASI 4.6.34

Version 1. This has several reasons:

- a) SBML Level 1 Version 1 is very limited and only a very small subset of models published today can be represented as Level 1 Version 1
- b) only very few programs are still in use today that only support SBML Level 1 Version 1 as their import format
- c) due to the limitations of SBML Level 1 Version 1, it is not trivial to convert an arbitrary model to the requirements of SBML Level 1 Version 1

Implementing support for exporting models to SBML Level 1 Version 1 would mean investing of lot of time for little to no benefit to the users of COPASI. For this reason support for exporting SBML Level 1 Version 1 has not been implemented in COPASI.

It is also noteworthy, that while COPASI supports most features of SBML, it currently does not support all features. SBML features currently not supported by COPASI are so called "fast reactions", algebraic rules as well as the delay function. A more detailed description of these features and why they are currently not supported by COPASI will be give below.

Implementation

COPASI uses libsbml[70] for reading and writing as well as verifying SBML documents.

libsbml is developed by the SBML community and the developers of COPASI have actively supported the development of libsbml by providing code, bug fixes as well as feedback.

The first version of libsbml was released in 2003 and development of the library progressed so fast that the first officially released version of COPASI in 2006 already used libsbml 2.3.2 for handling SBML documents. That version was already very stable with respect to the base functionality, which was reading and writing of SBML models, but support for the verification of SBML documents was still in its infancy.

libsbml was developed in conjunction with SBML and usually a release of a new version of SBML would be accompanied by a new release of libsbml.

In order to keep up with new version of SBML as well as new versions of libsbml, COPASI had to be modified and extended over time to incorporate new SBML features or to accommodate API changes in libsbml. Sometimes these changes were so significant that major parts of the SBML import/export code in COPASI had to be rewritten.

COPASI development started at around the same time the first version of libsbml was released. Since these early version provided only little support for checking models for correctness, a lot of code to do these kinds of checks have been implemented in COPASI. Back then this feature was used by many other developers to tests the validity of the SBML files they were creating with their tools. Although recent version of COPASI use newer version of libsbml that have very good support for checking SBML models, the validation tests implemented in COPASI remain in place are still used to check SBML models when they are imported. This creates very little overhead in term of computation resources and provides COPASI with a fallback solution for model validation that can potentially catch errors in the corresponding check done by libsbml. Because of the high quality standards with respect to SBML import/export, COPASI is being used for many years in the curation of the BioModels database[160].

SBML is not the native file format of COPASI but the COPASIs file format and data structures are very similar to those of SBML. As can be seen in figure 4.1 the core elements of the model definition, namely the definitions for functions, compartments, species, parameters, reactions and events are more or less the same in COPASI and SBML. Although initial assignments and rules are not listed in the COPASI data structures, these features exist.

They are defined together with the data structure to which they apply, e.g. the assignment rule for a species is not stored separately from the species, but directly with the definition of the species.

A feature of SBML that is not supported in COPASI, or at least not to the same extent, are unit definitions. SBML allows the user to define arbitrary units. In COPASI, the user can only choose from a predefined set of units. COPASI also has no notion of species types or compartment types.

On the other hand, COPASI has data structures specific to features found in COPASI that are not available in SBML, e.g. the definition and setting for the different simulation and analysis methods as well as the definitions for the different output types (plots & reports).

These are only the major differences between COPASI and SBML but there are many more small differences in the details of the individual data structures or in the interpretation of the data structures that are significant when it comes to importing models from SBML and exporting models to SBML in COPASI.

So although the overall structures of SBML documents and COPASI files are similar, the abundance of these small differences makes it a non-trivial task to correctly import an SBML document into COPASI or to export a COPASI model to a correct SBML model. Because small changes in the semantics of a model can lead to different results when working with a model, implementing this correctly was and still is our top priority.

SBML Unit Definitions

COPASI does not support unit definitions to the same extent as SBML. In COPASI there is a predefined set of units that the user can choose from and that are applied to a certain concept. E.g. if the user chooses seconds as the time units all elements that have a temporal component will use seconds as the unit for that temporal component. E.g. the kinetic parameter for mass action kinetics will have a unit in terms of $1/s$. These schema works well for the large majority of use cases.

In SBML arbitrary units can be defined and set on individual elements, e.g. a model can define that a certain parameter has the units *Volt/second* while another parameters has the units *mole/hour*.

When importing an SBML model, COPASI will try to identify the units defined in the SBML model and convert them to the corresponding units in COPASI if possible. If COPASI encounters unit definitions that it can not convert to one of its supported units, a warning will be issued.

Since SBML units do not have any influence on the results of expression evaluations and numerical calculations, the only consequence this has for the

model	
unit definitions	function definitions
function definitions	model
compartment types	compartments
species types	species
compartments	parameters
species	reactions
parameters	reactants
initial assignments	products
rules	modifiers
reactions	kinetic law
reactants	events
products	event assignments
modifiers	task descriptions
kinetic law	report definitions
events	plot definitions
event assignments	misc

Figure 4.1: Comparison of the SBML data structure and the data structures used in COPASI documents (left: SBML, right: COPASI). Structures that are common to both have been printed dark.

user is that some units in the graphical user interface might be displayed incorrectly. The numerical results of time course simulations and other analysis methods will not be affected by this limitation.

SBML Compartment Types & Species Types

In SBML documents "compartment types" and "species types" are used to establish a relationship between different compartments or species in a model. E.g. a model that contains three compartments named mito1, mito2 and mito3 could create a compartment type mitochondrium and declare that the compartments mito1, mito2 and mito3 are all of type mitochondrium.

COPASI does not have a corresponding grouping mechanism, so this information will be ignored when a model that contains such information is imported. Since this feature does not have any influence on numerical calculations with the model, analysis results from that model created with COPASI will not be affected by ignoring this information.

Because ignoring this information has no immediate consequence for the user, there is no warning or error message about it being ignored. However one has to keep in mind that this information can not be stored in a COPASI

file, so if a user imports an SBML model and stores it as a COPASI model, the information is lost.

SBML Constraints

In SBML the modeler can put constraints on certain values in a model and if the value violates such a constraint, e.g. during a simulation, the results are to be considered invalid.

COPASI does not currently support this feature and constraints are ignored when an SBML model is imported. In this case, the user will see a warning about the fact that COPASI has ignored these constraints. As a consequence the user can work with the model, but violations of constraints will not be reported during any kind of analysis.

Currently work is underway to add the data structures necessary for the support of constraints in COPASI. Once those supporting data structures have been implemented, support for importing constraints from SBML models will be added as well.

Algebraic Rules

SBML supports three different types of rules. Two of those, assignment rules and rate rules, are directly supported in COPASI. The third type of rules, algebraic rules, are currently not supported. If a model contains an algebraic rule, COPASI will import the model, but the algebraic rule will be ignored and the user will be notified via a warning message.

For many models, ignoring algebraic rules will have no or little effect, while for other models the results from simulations and other analysis methods may differ from what the modeler intended. Since the user is informed about COPASI ignoring these algebraic rules, he/she has to decide if this will influence the results that are created with this model in COPASI.

"fast" Reactions

In SBML reactions can be marked as "fast" by setting the corresponding flag on the reaction. These fast reactions are to be considered to occur at infinite speed. The fast flag effectively partitions the reactions that make up the model into two sets. Each of the sets working in a different time scale. In order to handle the model correctly, the "fast" reactions have to be handled in a special way. Failing to do so will likely lead to results different from what the creator of the model intended.

Because currently the simulation engine in COPASI can not handle fast reactions appropriately, the fast flag on reactions is ignored on import. The

user is informed about this and about the fact that the results of simulating the model might deviate from the behavior the modeler had intended when writing the model.

Since not all types of analysis are affected by "fast" reactions, the model is still imported and the user has to make sure that he only does those types of analysis that do not depend on the correct interpretation of the kinetic data. Network based analysis methods as for example the calculation of elementary flux modes will work with this type of model and will produce correct results.

Time Delays

SBML has support for a so called *delay* function. The delay function $delay(x, y)$ has two arguments, both of which are arbitrary mathematical expressions. The result of evaluating the delay function is the value of expression x at y time units before the current time. This can for example be used to encode delay differential equations in SBML documents.

The simulation methods in COPASI currently can not handle delays and although SBML models with delays can be imported in COPASI, all calls to the delay function will evaluate to the special value "Not a Number" (NaN) to signify that the result is invalid. So any method that relies on the calculation of states of the model will return invalid results.

If the user imports a model that contains expressions with delays, COPASI will issue a warning, notifying the user that the model can not be simulated in COPASI.

As for the "fast" reactions described above, network based analysis methods, e.g. the calculation of elementary flux modes will not be affected by this and the user can still carry out these kinds of analysis.

Species With *hasOnlySubstanceUnits* Flag Set

Normally if the identifier of a species appears in any mathematical expression in an SBML file, it represents the concentration of that species. However, species have a flag called `hasOnlySubstanceUnits`. If that flag is set, the identifier of the species has to be interpreted as the amount of the species rather than the concentration.

COPASI does not make such a distinction on the species, but rather each time the concentration or the amount of a species is used in an expression, the user has to specify if it is the amount or the concentration. Kinetic laws in COPASI are an exception to that rule. In kinetic laws, a species is represented by its concentration if the reaction spans a single compartment,

in multi compartment reactions COPASI uses the amount rather than the concentration.

This means that on import COPASI has to monitor the use of the `hasOnlySubstanceUnits` flag for each species and replace references to species in expressions according to the way they are interpreted in COPASI. This also has implications for the import of other model elements, e.g. assignments to species in rules and/or events.

This can lead to a number of minor changes to the way the model is represented. Because the changes do not affect the semantics of the model, calculation results will still be correct. However, when the model is exported again to an SBML file, it will contain all the changes introduced during import.

Reactions Spanning Multiple Compartments

Traditionally rate laws specify the change of a reactants concentration over time. This works very well for reactions taking place in a single compartment, but fails once the reactants of a reaction span multiple compartments.

A detailed explanation of this problem is given in the specification of SBML Level 2 Version 4[113].

As a consequence of the problems described in that specification, the SBML community decided that rate laws in SBML documents specify the change in amount per time rather than the traditional concentration per time.

Because COPASI strives to be as user friendly as possible and biologist are used to specifying rate laws in terms of concentration per time, COPASI has kept this concept and all rate laws for single compartment reactions specify the change in concentration per time for the reactants.

Since this does not work well for multi compartment reactions as detailed in the SBML specification, COPASI makes a compromise here and rate laws for multi compartment reactions are interpreted as the change of amount per time just as in SBML models.

This difference in interpretation of rate laws between COPASI and SBML makes it necessary to sometimes modify rate law expressions on import as well as export to preserve the semantics of a model.

This process can be further complicated by other differences between COPASI and SBML, e.g. the `hasOnlySubstanceUnits` flag described above.

Stoichiometry Math & Species Reference Rules

SBML versions up to SBML Level 3 declare an attribute called `stoichiometry-Math` on species references which can be used to specify a mathematical expression instead of a constant numerical value for the stoichiometry of a certain species in a reaction. In SBML Level 3 and above the same feature is represented via rules for species references.

This feature is only partially supported by COPASI. COPASI itself can only handle fixed numerical values for the stoichiometries of species references. When COPASI encounters an expression for a stoichiometry it evaluates the expression once and uses the resulting value as the fixed value for that stoichiometry. This works well if the stoichiometric expression itself only contains fixed elements, for all other cases COPASI will not be able to do calculations with the model correctly. Since this can lead to incorrect results, the user is informed about the conversion from a potentially variable expression to a fixed numerical value.

COPASI Features Not Supported In SBML

In the preceding sections SBML features that are not or only partially supported in COPASI have been described, but there are also features available in COPASI that are not supported by SBML or that are supported in a different way.

Since the SBML file format only covers the specification of reaction networks, almost all features of COPASI that are not part of the reaction network specification can not be exported to SBML.

COPASI for a large set of analysis methods and the user can modify the settings for these methods as well as define output in the form of plot or reports. Currently all these settings can not be exported to SBML files. So if a user exports his model from COPASI to SBML to work in another program and returns to COPASI with that model later, all settings and output definitions are gone.

We are currently evaluating ways of solving this problems, e.g. by saving the settings into annotations in the SBML model or using a separate SED-ML[161] file to preserve the information.

There are also some features with respect to the models themselves that are supported by COPASI, but that are not supported in SBML, or only in certain versions of SBML. If COPASI encounters such an unsupported feature during export to SBML, the user will be notified and the export will be stopped.

An example of a feature that is supported COPASI, but not in SBML are

the random distribution functions `runiform` and `rnormal`. These functions can be used to include (pseudo-)random numbers from uniform or normal distributions in mathematical expressions.

Another example is the `arctanh` function which is supported in SBML Level 2 and above, but not in SBML Level 1. This problem can be solved by converting the `arctanh` function into an equivalent mathematical expression which is provided by the term $1/2 * (\log(1 + X) - \log(1 - X))$. In this case the results generated with the model do not change, but the expressions formerly using the `arctanh` will look different after export and the user has to be aware of this. However there are also cases where no equivalent mathematical expression can be found as is the case for the `arccoth` function. This function is also only supported in SBML Level 2 and above and there is no simple mathematical expression it can be substituted with when exporting to SBML Level 1. If that function is encountered in a model, COPASI will stop the export with an error message.

So depending on the Level and Version of SBML to be exported, COPASI has to check a number of things to make sure that exporting the model is actually possible and sometimes has to replace incompatible parts to make the export possible.

Since the number of tests implemented for the different levels and versions of SBML are quite numerous, a complete discussion of all of them is beyond this work.

Conversion Between SBML Levels And Versions

Another problem with respect to exporting to SBML is support for converting between SBML levels and versions.

By now, there are seven different official versions of SBML and COPASI can read all of them and all but SBML Level 1 Version 1 can also be exported.

Sometimes it happens that a user loads a model in a certain version of SBML, but wants to export it as another version of SBML. For certain features of SBML this can lead to problems. E.g. if the model uses features that are either not supported in the SBML version he/she want to export to or if the feature is interpreted in a different way.

Most of the trivial conversions when changing SBML Levels are by now supported by `libsbml`, but there are still some necessary conversions that aren't.

For many of these cases not covered by `libsbml`, COPASI contains extra code handling these conversions prior to passing the model to `libsbml` for processing. This allows the user to convert most models encountered today into any version of SBML that is supported by COPASI. Due to this, the

user can always choose the tool that best fits the problem he/she is working on without having to worry about whether this tool supports a particular version of SBML or not.

SBML Level 1 Export

A rather special case for the conversion problems mentioned above is the export to SBML Level 1 Version 2. While most programs support at least SBML Level 2, there are some older programs still in use today that only support SBML Level 1. In order to allow users to export SBML models to this older version of SBML, COPASI contains a lot of code that checks if the model is compatible with SBML Level 1 as well as a lot of code to modify the model to make it suitable for export to SBML Level 1 in case minor incompatibilities were found.

For example SBML Level 1 does not know the symbols for π or the euler number e , so all occurrences of those symbols within mathematical expressions have to be converted to equivalent expressions, e.g. the euler number e is converted to the numerically equivalent function call `exp(1.0)`.

Likewise most of the trigonometric functions other than `sin`, `cos` and `tan` are not supported in SBML Level 1 Version 2 and therefore they also have to be converted to equivalent mathematical expressions based on other function calls, e.g. `coth(x)` is replaced by $\frac{e^x + e^{-x}}{e^x - e^{-x}}$.

Only after these conversions have been made, we can use `libsbml` to do the rest of the conversion.

The main emphasis on all those conversions is to modify the model as little as possible. If modifications are made they may not change the semantics of the model. If an appropriate conversion can't be found as it is for example the case for the `arccoth` function, COPASI will refuse to export the model to SBML Level 1 and issue a corresponding error message.

Testing

Even so the data structures in SBML and COPASI have a lot of similarities, importing and exporting of SBML in COPASI is not always trivial due to the many small differences, some of which have been described above.

To test and further improve the quality of SBML import/export in COPASI a large number of automatic tests have been written. Some of the tests are based on bugs found in COPASI, others have been implemented to assure that newly implemented SBML features are working as expected.

The code for testing SBML compliance in COPASI consists of about 270 individual tests which check most aspects of SBML import and export for

different versions of SBML. The tests have been implemented using the C++ programming language and the CppUnit[92] unit testing framework.

Very early in the development of SBML, Andrew Finney recognized the need to be able to test SBML compliance of different software tools. For this purpose he developed a set of roughly 150 SBML Level 2 test files that covered much of the functionality of SBML. The emphasis of these tests was on testing SBML compliance with respect to deterministic time course simulations, so in addition to the test files, the test suite contained data files representing time course simulation results for the individual test models. Since the result files had been created with an early implementation of a tool called MathSBML[162], these result files initially contained some errors and when I started to use the test suite with COPASI, several of the tests failed because COPASI did not produce the results expected by the test suite. Some of these failures could be traced back to errors in the SBML support of COPASI, but others could not be explained even after detailed analysis. We reported this back to the developers of the test suite and together we found that the corresponding result files created by MathSBML were incorrect. So while testing support for SBML in our own tool using this test suite, we did find errors in our implementation as well as errors in the tests suite. So on the one hand, we benefited from these tests by finding the errors in our implementation and by providing feedback to the author of the test suite, we can take credit for many of the fixes in the test suite as well as in other implementations of the SBML standard as e.g. MathSBML. This process finally lead to a correct and very useful set of tests that has been used in COPASI for a long time to ascertain the correctness of SBML import and export.

This test suite was not only beneficial to developers, it also allowed the users of different simulation tools to test these tools and choose the one that provided the best value, i.e. the highest SBML compliance.

The 150 tests of Andrew Finney test suite very well reflected the capabilities and features of early version of SBML and of the tools implementing support for SBML. As SBML kept developing more and more features were added and new versions were released, unfortunately the test suite could not keep up with this development. This was especially troublesome since the increase in complexity of the SBML standards made testing for SBML compliance during tool development even more important.

This lack of proper ways for testing SBML compliance lead to the development of a new test suite and in April of 2008 a successor for the SBML semantic test suite was presented[163]. The new test suite had the same goal

as the old test suite, but it provided a more systematic set of test cases, a graphical user interface written in Java as well as filtering mechanisms that allow the user to specify what features of SBML should be included in a test run.

Just as for the semantic test suite before, developers who wanted to use the test suite had to provide a wrapper program that was called by the test suite program to simulate the test models and create the required output. The results provided by this wrapper program were then compared to standard results provided by the test suite.

In the beginning this test suite also contained some errors which were soon eliminated, again partially due to feedback from us. Today, the number of tests available in the test suite has reached almost 1000[164] and most of these tests are available for a number of different versions of SBML. The SBML versions supported by the test suite currently include SBML Level 1 Version 2 and all versions of SBML Level 2.

The SBML test suite was released at around the same time as COPASI version 4.5.30. Figure 4.2 shows results from this early version of COPASI (left) and results for the same tests from version 4.6.32 of COPASI.

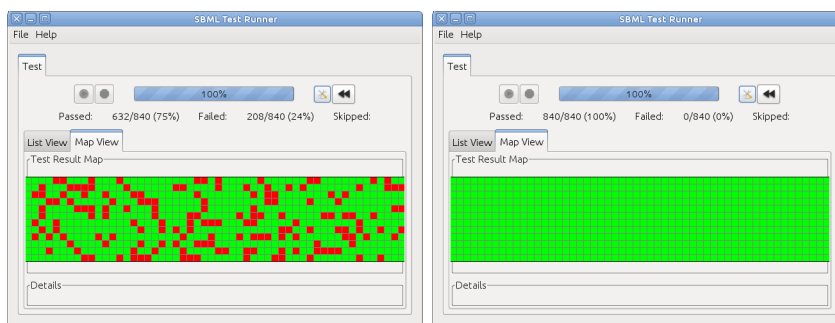


Figure 4.2: Results for the SBML test suite for COPASI 4.5.30 (left) and COPASI 4.6.32 (right). While version 4.5.30 had issues with several of the tests (red blocks), version 4.6.32 passed all the tests possible with the feature set implemented. (Test for algebraic rules and "fast" reactions were excluded from the test runs.)

Even so, the graphical user interface never made it out of the beta stage, the test cases themselves are actively developed, updated and extended and the occasional errors are soon caught by the SBML community.

The graphical user interface has been abandoned in favor of a web based solution[165] where the user specifies a set of features and the version of SBML he/she would like to test. A web server then provides the user with

an archive file that contains the test files, instructions on how the individual test models have to be simulated and what output is expected from the individual tests. Once the user has simulated all test models according to the information specified, the result files are uploaded to the web server for evaluation. The result of this analysis is then displayed in the browser as depicted in figure 4.3.

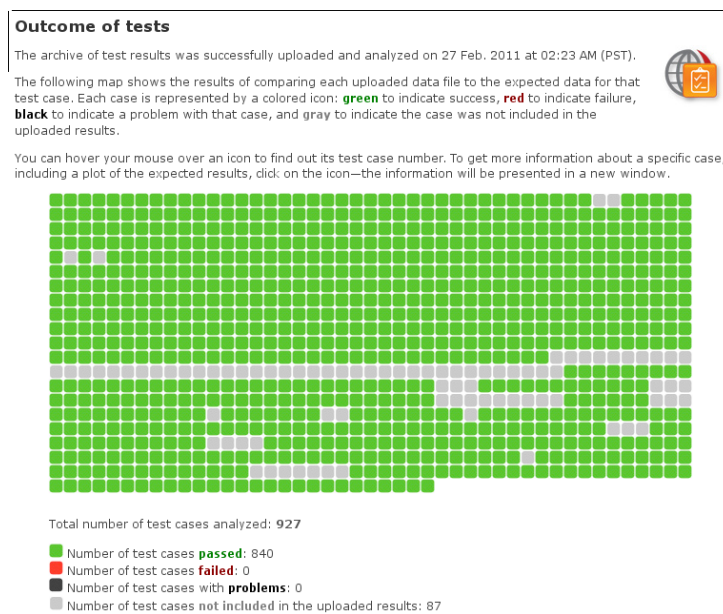


Figure 4.3: Analysis result of running the SBML online test suite with COPASI Build 34 against SBML Level 2 Version 4 files. The result shows that out of the available 927 tests, 840 tests were run and of those 840 tests all passed (green boxes). The remaining 87 tests (gray boxes) that were excluded contain algebraic rules and/or fast reaction which are currently not supported by COPASI.

Due to its much larger scope, testing SBML compliance in COPASI is now done with the new test suite. The test suite enables us to check SBML support for many different SBML versions on a regular basis, ensuring the continued quality of SBML import and export on the one hand and correct behavior of the simulation engines on the other. Being part of the SBML community, we also provide the developers of the SBML test suite with feedback by reporting errors in the test files.

Another set of tests that is used to compare the status of SBML support in

different simulation tools has been developed by Frank Bergmann[166, 167]. His test compares how different simulation programs simulate a set of SBML models from the BioModels database[168]. Developers of tools who want to participate in the testing process have to simulate all curated models from a certain release of BioModels according to instructions stated on the web page for this test. The results of these simulations are then graphically compared to simulation results from other programs.

This test has several benefits. First of all, authors of programs can compare the performance of their tools with respect to SBML compliance with the performance of other tools. If the test shows that there are differences in simulation results between different programs, this finding can be analyzed and discussed with the authors of the other tools. This way all tool developers benefit from the resulting discussions allowing them to improve their software.

The test can also provide some insights as to how well SBML or certain features of SBML are supported by different tools. Figure 4.4 shows how many of the simulation runs from the different simulators provided a valid result. This figure can be taken as a hint of how well SBML models and the different features used in the 150 models used in the test are supported by the different simulation tools. The diagram however does not provide any insights as to whether the results provided by the different simulation tools are in agreement or correct.

Based on this idea, this test is used as a sort of a realistic stress test to see how well COPASI can handle SBML models. The test cases we use for this test are regularly updated whenever a new version of BioModels is released. Importing, exporting and simulating almost 600 models provides a very good real world test of how well SBML models are handled within COPASI.

The semantic test suite by Andrew Finney and its successor the SBML Test Suite as well as the test by Frank Bergmann are all used to test SBML compliance of simulation tools with respect to deterministic simulation.

While most models stored as SBML documents are meant to be simulated deterministically, models in SBML documents can in principle also be used for stochastic simulation provided that some constraints apply.

In order to test compliance of stochastic simulation programs, Daren Wilkinson has created a test suite[169] that consists of a number of SBML documents describing stochastic models together with instructions on how to simulate them as well as the expected simulation results. In order to run the tests, the user has to provide a wrapper program that simulates the models and creates simulation results as provided by the instructions in the test suite.

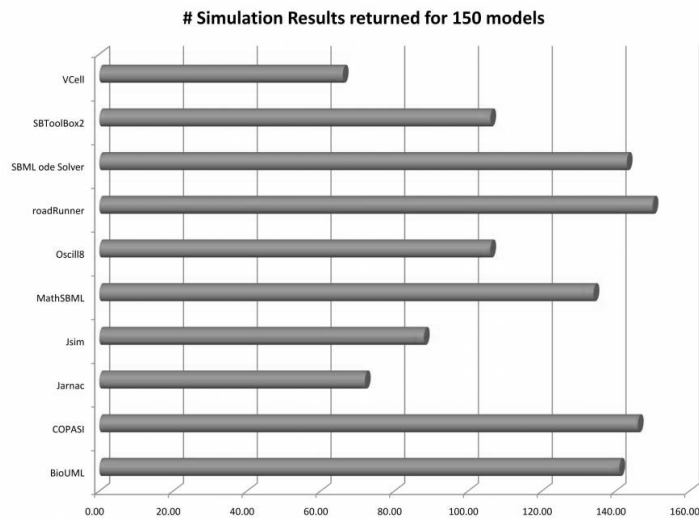


Figure 4.4: One of the result diagrams of Frank Bergmanns simulation tool comparison. Results for COPASI version 4.4.26 is displayed in the second to last row. As can be seen, COPASI is able to provide results for almost all of the 150 model files used in the comparison (source: <http://www.sysbio.org/sbwWiki/compare>).

Since the stochastic test suite does not provide any means of analyzing the results of running the tests, I had to write these tools myself.

COPASI contains several stochastic simulation engines and we use the stochastic test suite on a regular basis to make sure that the stochastic simulation engines work as expected. For this we wrote a wrapper program that can simulate the test models of the stochastic test suite according to the instructions provided and we have some scripts written in Python scripting language that do the analysis of the results. COPASI provides several stochastic simulation methods and therefore the wrapper has been written in a way that lets the user specify the desired method when the program is executed. This way the stochastic test suite can be run automatically for all stochastic simulation methods with a single program and a small script.

Standard compliance and quality, especially with respect to SBML is very important in COPASI. For that reason a lot of effort has been spent to assure this compliance and quality. A large number of tests, either written by us or provided by others in the SBML community assure that we can keep these high standards and sometimes even improve on them.

```

dsmts-001-01 passed
dsmts-001-02 passed
ERROR at ./dsmts-24/dsmts-001-03-sd.RESULT (40, 1): Var: 45.685402, RefVar: 41.647533, Tol: 2.944925.
ERROR at ./dsmts-24/dsmts-001-03-sd.RESULT (41, 1): Var: 41.970626, RefVar: 37.758321, Tol: 2.669917.
ERROR at ./dsmts-24/dsmts-001-03-sd.RESULT (42, 1): Var: 36.966971, RefVar: 34.225893, Tol: 2.420136.
ERROR at ./dsmts-24/dsmts-001-03-sd.RESULT (43, 1): Var: 33.288396, RefVar: 31.018439, Tol: 2.193335.
ERROR at ./dsmts-24/dsmts-001-03-sd.RESULT (44, 1): Var: 30.190950, RefVar: 28.107387, Tol: 1.987492.
dsmts-001-03 failed
dsmts-001-04 passed
dsmts-001-05 passed
...

```

Figure 4.5: Excerpt of the output from a sample run of the stochastic test suite with COPASI 4.6.32. Each model of the test suite has been simulated 10000 times using the "direct method" as the stochastic simulation method. Of the first five test runs, 4 tests passed and one test failed with data points being outside the allowed tolerance. Since this is a stochastic process, there is always a certain probability that one or the other test fails. Usually we run the test suite several times and to check if COPASI fails one or more of these tests consistently.

MIRIAM Support

While SBML surely is the largest and most important standard COPASI supports, it is not the only one. Another standard that is implemented in COPASI is the Minimum Information Required in the Annotation of Models (MIRIAM) standard[170, 171]. MIRIAM is less of a format, but rather a set of rules that describe what additional information should be present in a model document.

The information favored by the MIRIAM standard include for example the author of the model, references to publications concerning the model, or references to biological databases for model entities etc. This information can sometimes be essential for the proper understanding of a model.

In order to be accepted to the BioModels database, a model file has to adhere to the MIRIAM standard.

COPASI's graphical user interface provides the user with the means to annotate models according to the MIRIAM guidelines. MIRIAM annotation information present in model files loaded by COPASI are displayed in the GUI and can be edited and complemented by the user (see figure 4.6).

COPASI as well as SBML stores MIRIAM annotations in the XML based Resource Description Framework (RDF)[110] format. However just as with the actual model data, there are differences in the details of how the MIRIAM information is stored.

Currently the MIRIAM information that can be stored as described by

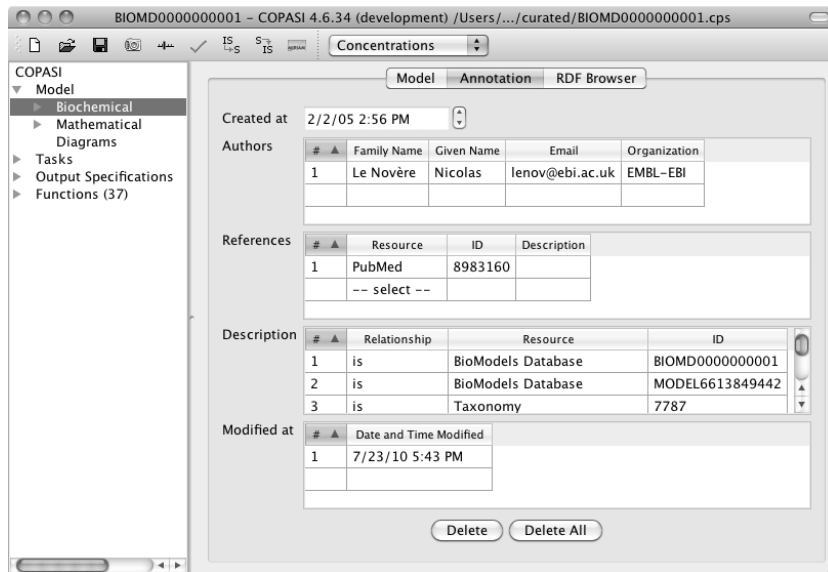


Figure 4.6: Screen shot of MIRIAM annotation for a model from the BioModels database displayed in the graphical user interface of COPASI.

the SBML specification is more constrained than what can be stored in a COPASI file. So if MIRIAM annotation that has been created in COPASI is exported to SBML, we have to make sure that the restrictions as specified in the SBML specification are followed. This on the other hand means that annotation provided by the user would be lost when the model is exported to SBML.

In order to preserve this information, the MIRIAM information is stored twice in the SBML file. Once in the way specified by the SBML specification and once as an annotation with the complete information as it would be stored in a COPASI file. The mechanism of these so called annotations to SBML has already been described in the section on the SBML layout and render extension where it is used to store layout and style information in SBML documents (see chapter 3.4).

Saving the MIRIAM information in this way allows other programs that only understand MIRIAM annotation as described in the SBML specification to read that part of the information, while COPASI can read back the full information when the model is reimported into COPASI.

A bit of a drawback is the fact that other programs sometimes change the SBML specific annotation while the COPASI specific annotation remains unchanged, creating annotations that are out of sync. This conflict is partially resolved by always reading the annotations in the COPASI format first.

This way the SBML annotation that might have been changed by another program overwrites any obsolete entries.

4.2 Layout And Render Information In COPASI

In the section on the SBML layout and render extension (see chapters 3.4 & 3.6), the concept and elements that are involved in storing layout and render information in SBML files have already been described. It was also mentioned that one of the design principles of the layout and render extension was to keep the format as independent of the underlying model format as possible.

Having the layout and render information independent of the model format made it possible to store layout and render information in COPASI files in almost exactly the same way as in SBML files.

There are some very small differences, e.g. in SBML models, elements are referenced by the value of their *id* attribute while in COPASI elements are referenced by a so called key. So if elements from a COPASI model are referenced from a layout in the COPASI file, the reference is to the elements key. Likewise elements in the layout and render information that need to be uniquely identifiable have a *key* attribute in COPASI files instead of an *id* attribute.

Also the element names in COPASI files follow a different schema with respect to uppercase and lowercase letters and the names of the layout and render information element tags have been adjusted to respect this schema.

Last but not least, layout and render information in COPASI does not reside in an annotation, but it is part of the model file structure, similar to how it is intended for SBML Level 3.

Using the same principles to store layout and render information in both SBML and COPASI files has several advantages. First of all, it was possible to take the classes that had already been implemented for libsbml and reuse most of the code in COPASI. Because of this, the data structures for storing layout and render information in SBML and COPASI are virtually identical which simplifies importing and exporting layout and render information from and to SBML documents. The initial data structures for layout elements have been implemented in COPASI by Sven Sahle while I implemented the classes for the render information (see figure 4.8).

Rendering the layout and style elements also profited from having similar data structures in COPASI and SBML. Since I had already implemented a library for rendering layout and render information from SBML files (see 3.6), I was able to use that library to display layout and render information

```

<COPASI>
  <ListOfFunctions>
    ...
  </ListOfFunctions>
  <Model>
    ...
  </Model>
  <ListOfTasks>
    ...
  </ListOfTasks>
  <ListOfPlots>
    ...
  </ListOfPlots>
  <ListOfReports>
    ...
  </ListOfReports>
  <ListOfLayouts>
    <Layout>
      ...
    </Layout>
    ...
  </ListOfLayouts>
</COPASI>

```

Figure 4.7: Example of layout information stored in a COPASI ML file. The layout information (dark section) follows the list of plots and reports towards the end of the file.

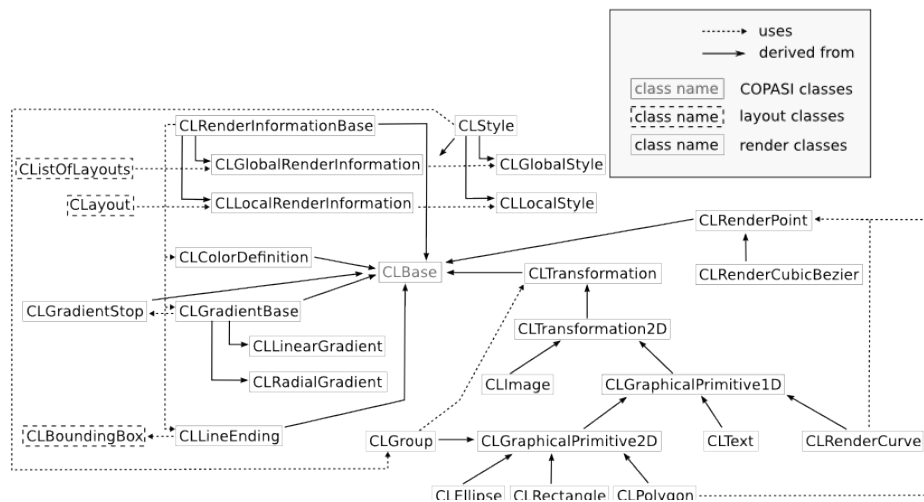


Figure 4.8: Inheritance diagram for the render classes as implemented in COPASI.

within COPASI with only minor modifications.

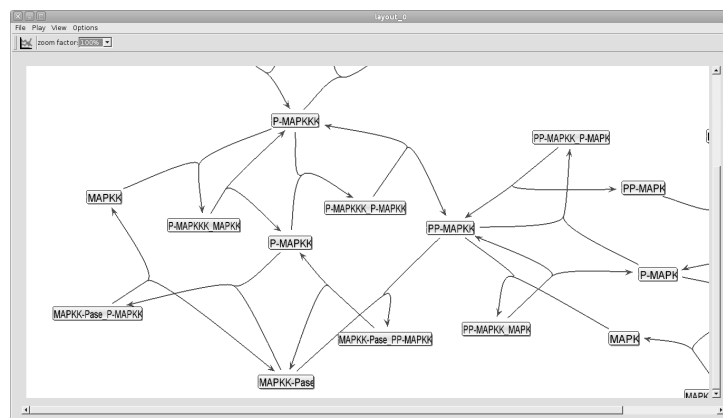


Figure 4.9: Example rendering of layout information in COPASI 4.5.31.

A complete test suite for the SBML layout and render extension already existed as described in the corresponding part on the SBML layout and render extension (see chapter 3.6), so I was able to use these tests to check the implementation of SBML layout and render information import and export to and from COPASI. To use this test suite, all tests written for the SBML implementation were imported in COPASI, thereby testing the import code for the layout and render information. Afterwards the layout and render information was written to a COPASI file. In the last step the layout and render information was read back from the COPASI file and displayed. To see whether the conversion as well as the rendering worked as expected, the displayed images were compared to the corresponding images as displayed by the implementation for SBML files.

In a further step the generated COPASI files were converted back to SBML files and the layout and render information was again rendered with one of the SBML based implementations, thereby testing the layout and render information export from COPASI files.

This way we could test all aspects of reading writing, rendering as well as the conversion to and from SBML with this set of tests.

Having an implementation of layout and render information in COPASI allows users to convert their SBML models including layout and render information to COPASI and back without losing any information in the process.

Besides providing a graphical overview over the reaction network, being able to store and exchange this type of graphical information has additional benefits for users as well as developers of systems biology programs because

the diagrams can often be used to display results of simulations or other types of analysis. An example of how this is done in COPASI will be provided in the following section.

4.3 Graphical Display Of Time Course Simulation Data

An example where COPASI uses layout information to display analysis data is the graphical display of simulation results as an animation. The concept is based on a tool called SimWiz which has been developed by Dr. Ursula Rost[172]. The original SimWiz was a standalone Java based program that allowed users to load SBML models and simulation results and display the simulation results as an animation. During this animation the size or the color of a node in the reaction network diagram changes over time according to the concentration data of the corresponding model species from a time course simulation.

Eventually, development of SimWiz as a separate program has been abandoned in favor of an implementation in COPASI. The first implementation in COPASI was done by Dr. Ursula Rost and was released with version 4.4.28 of COPASI in December of 2008. Since then, the original code has been almost completely replaced by a new implementation which is more stable, reliable, faster and uses less compute resources. The main advantage of this is that the implementation can now also be used on older hardware that is not as powerful.

While the main functionality of this visualization tool has been mostly unchanged over the years, the implementation in COPASI provides several advantages. For one, the tool can use layout from SBML files as well as layout from COPASI files and will directly profit from any new feature with respect to layout support in COPASI as for example the CellDesigner conversion tool described in chapter 4.6. Another advantage is that the tool has direct access to the time course simulation data calculated in COPASI and can visualize it without having to load the data from some external file first.

There is a number of settings that the user can make, for example whether the concentration is represented by the size of the nodes or by the color (see figure 4.10). The user can also choose how the concentration data is scaled for display. The data can either be scaled for each node individually, or globally for all nodes. This means that the size or color range of a certain node is either scaled between the minimal and maximal concentrations of the species that corresponds to a certain node or between the minimal and

maximal concentrations of all metabolites in the model.

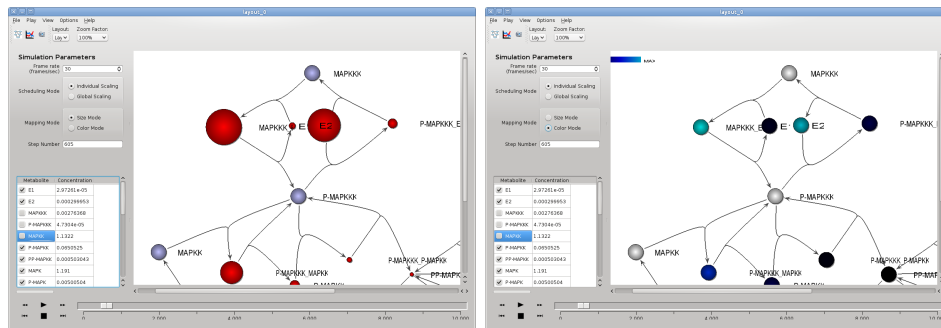


Figure 4.10: Rendering of a single frame of the time course animation (Left: size mode, Right: color mode).

The animation for the simulation data is controlled similar to a CD player. Clicking on the run button starts the animation. For each time point in the simulation data, the GUI displays a new frame. The user can stop and pause the animation as well as single step forward and backward through the animation. The slider at the bottom represents the time range of the simulation and it can be used to jump to an arbitrary time point of the simulation.

With the current functionality the tool is already very useful for displaying time course simulation data in an intuitive way. Ways to extend this feature to make it even more useful are currently being worked on.

4.4 Graphical Display Of Elementary Modes

Another very useful way of using the layout information to display analysis results in COPASI is for the display of elementary flux modes[173].

COPASI has the possibility to calculate the elementary flux modes of a reaction network, but until now the calculated elementary flux modes were displayed as a list of reaction names, which is not very intuitive and which does not provide the user with an overview of the elementary modes in the context of the complete reaction network.

Using existing layout information for the reaction network, the reactions belonging to a certain elementary mode can be displayed by highlighting the graphical representation of the reactions within the diagram (see figure 4.11). This allows displaying the elementary mode in the context of the complete reaction network.

This is a very recent feature, but it has been tested extensively and it will be released with one of the next versions of COPASI.

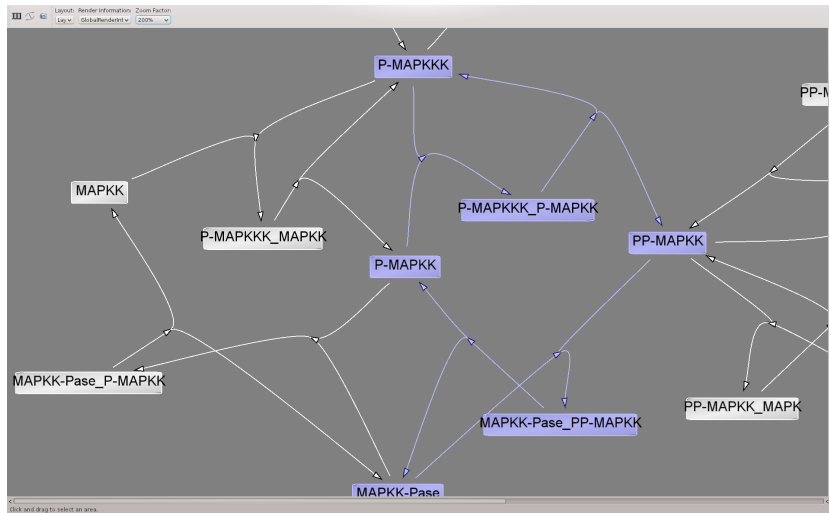


Figure 4.11: Display of reaction network in COPASI with one elementary mode highlighted (blue).

4.5 COPASI Language Bindings

This project started as a collaboration between the developers of CellDesigner[31] and COPASI.

CellDesigner is a software tool that focuses on the creation of reaction network models by graphically composing the reactions. For doing time course simulations or other types of analysis on the resulting models CellDesigner relies on third party tools.

In the beginning CellDesigner allowed simulation via a tool called SBML ODE Solver[174] which was embedded within the CellDesigner software. In 2008 a collaboration with the developers of CellDesigner was initiated to extend CellDesigner with the option of simulating models using the methods provided by COPASI.

In order to achieve this goal, several problems had to be solved. For one, COPASI was never intended to be used by other programs, but it was designed as a standalone program. In order to be usable from other programs, it had to be converted to a software library. Since many sub-components of COPASI had already been set up to be build as separate libraries, it was only

a matter of changing the build system to combine these individual libraries into one large library.

The bigger problem that we had to solve was that the COPASI software was written in the C++ programming language and CellDesigner was written using the Java programming language and functionality from one of those languages can not easily be used from the other. To solve this problem, we had to write an interface between the COPASIs C++ code and the Java programming language. Such interfaces between Java and C/C++ code are usually written using the Java Native Interface (JNI) API which is provided by every Java implementation. The JNI API makes parts of the Java virtual machine and the Java API accessible from low level languages like C and C++. This makes it possible for code written with the two programming languages to communicate. In order for this interaction to work, developers have to write code that translate C++ methods and data structures to the corresponding Java methods and data structures and vice versa. Such interface code can either be written manually, or created in an automatic or semi-automatic way with the help of specific software tools. These software tools are able to create the JNI interface code from C/C++ method and data structure declarations without much work from the side of the developer.

One program that can create interface code from C/C++ declarations is called Simplified Wrapper and Interface Generator (SWIG). SWIG takes a number of so called C/C++ header files which contain the declaration for the methods and data types to be called from Java and creates the JNI interface code for these methods.

While SWIG can create most of the interface code automatically if provided with the corresponding C/C++ declarations, some extra information usually needs to be provided to SWIG for the resulting interface code to work correctly. This extra information is provided in the form of SWIG interface files and contains information about which elements in the declaration files are to be made accessible to the target language (e.g. Java) and how certain data structures are to be interpreted and converted to ensure that they are understood by the JVM.

An additional advantage of using SWIG instead of manually writing the interface code is that SWIG is not limited to creating interface code for Java, but it can create interface code for a large number of high level languages (see table 2.1).

Since each of the target languages supported by SWIG is different, some extra code has to be written to support additional languages, but most of the extra information in the SWIG interface files can be applied to all target languages. This means that with some extra effort, interfaces to COPASIs functionality for a large number of languages can be generated allowing soft-

ware projects written in any of these languages to profit from the many sophisticated simulation and analysis methods implemented in COPASI.

In theory any of the languages from table 2.1 could be supported, but in practice, we are limited to those languages we know how to program in. Due to this, currently only Java and Python language bindings in addition to the C++ interface to COPASI are available.

The first version of the language bindings for Java and Python have been officially released in June of 2008 and since then new versions have been released with every stable version of COPASI.

The language bindings are released as source code as well as in binary form for Java and Python. The releases cover all platforms that are supported by COPASI. A list of languages and platforms supported by the language bindings has been collected in table 4.2.

operating system	Java	Python
Windows XP, Vista, 7	32 Bit	Python 2.6, 32 Bit
Linux	32Bit	Python 2.6, 32 Bit
Mac OS X 10.4	32 Bit	Python 2.3, 32 Bit, PowerPC & Intel
Mac OS X 10.5	32 Bit	Python 2.5, 32 Bit, PowerPC & Intel
Mac OS X 10.6	32 Bit & 64 Bit	Python 2.6, 32 Bit & 64 Bit, PowerPC & Intel

Table 4.2: Operating systems and programming languages supported by the COPASI language bindings.

Future version of COPASI will also support 64 bit versions of Linux, Mac OS X as well as Windows, significantly enlarging the number of language bindings version that have to be build.

Exposing COPASIs simulation and analysis methods to Java via SWIG was only part of the work necessary to being able to simulate models in CellDesigner. Another essential part was to write a user interface in Java that let the user set certain parameters for the simulation, e.g. the end time of the simulation, or the number of steps to be calculated by the simulation.

These graphical user interface classes as well as the interface classes created with SWIG are used in current versions of CellDesigner to allow users to simulate models with COPASI from within CellDesigner. The graphical user interface files are also part of the Java language bindings distribution files and can be used by other Java programs as well.

This project started out as a collaboration between us and the developers of CellDesigner, to bring COPASI's functionality to CellDesigner, but has quickly grown beyond this initial goal. As can be seen by the user support forum on the COPASI web server, other software and research projects are using the COPASI language bindings[175, 176].

Since COPASI was not intended to be used in this way in the beginning, documentation for COPASI has been limited to the user interface and the methods used in COPASI's backend. In order for other developers to be able to use COPASI's functionality, documentation on COPASI's internal structures and data types is needed.

As COPASI has grown to be a large project, creating a complete documentation of all data structures and methods would take a lot of work. In addition to that, the API is still growing and changing with every release. Although full documentation is out of scope for now, documentation on the core classes, concepts and methods as well as examples of how certain tasks can be implemented using the different programming languages are provided. The documentation consists of roughly 70 pages for each of the programming languages and is supposed to give the interested developer an overview over the core functionality of COPASI and how it can be used from other languages. The documentation can be downloaded in PDF format from the COPASI website[156]. The ten examples that are currently shipped with the language bindings demonstrate how to import and export SBML models, run time course simulations, create models or how to run parameter scans, optimizations and parameter estimations. Each of the examples is available in a version for C++, Java and Python.

4.6 NF- κ B Modeling with COPASI

Because of its ability to create topological reaction network models graphically, CellDesigner has been used to create the initial version of the NF- κ B model, as described in chapter 3.8.

CellDesigner focuses on model creation and has very limited analysis and simulation capabilities. Most of these additional capabilities are provided by other software tools, e.g. parts of the simulation capabilities found in CellDesigner have been implemented using the COPASI language bindings described above.

Due to this lack of analysis and simulation methods, other tools have to be used for these tasks and one of the tools that provides a large number of such methods is COPASI.

Due to the implementation of the SBML standard in COPASI it is no

problem to transfer the model from the CellDesigner software, that uses SBML as its native file format, to COPASI. With the help of the additional converter described in 3.8 it is now also possible to transfer the diagram information from CellDesigner to SBML and to subsequently import that SBML model into COPASI.

So far only an early topological model has been created. In order to eventually come up with a model that can reproduce the experimental data measured by the group of Dr. Breuhahn, this model needs to be extended with rate laws for the individual reactions as well as with initial values for the individual components participating in these reactions.

For this, the model elements (compartments, species and reactions) are first annotated according to the MIRIAM standard[170]. MIRIAM annotations associate biological meaning to the individual elements of the model. They can for example be used to identify identical or homologue entities in other models or in biological databases for automated data mining.

These annotations have to be added manually, but COPASI provides an intuitive user interface for this task. Since MIRIAM annotations are fully supported by SBML as well as by the COPASI file format (see chapter 4.1), the information is preserved no matter which file format is chosen as the storage format and it enables the use of this information in other programs that support SBML.

These annotations we added on the one hand provide biological meaning to the model elements and on the other hand they can be used to do some (semi-)automatic data mining using some of the available biological databases (see 5.6).

Another type of analysis that can be useful at this stage of the model creation process is the elementary flux mode analysis[173, 177]. The elementary flux mode analysis tries to find connected reaction sub-networks that can form steady states if examined separately. A steady state is a state of the model where the concentrations of the individual reaction components no longer change. This means that for each reactant of the network, the consumption and the production have to cancel each other out. A trivial example would be if the reaction rates of all the reactions in a reaction network are zero. Since this is true for any network it is not considered by the elementary flux mode analysis. The elementary flux mode analysis only considers those steady states where the fluxes through the reactions are different from zero.

If for a certain element or reaction of the network no elementary mode can be found by the elementary flux mode analysis, this can have two reasons:

- a) The concentration of that element always changes (more substance is continuously produced than consumed or vice versa). This is unphysiological because there are no unlimited amounts to substances within organisms.
- b) The trivial case occurs, which means that the reaction rate to and from that element is zero. Reaction rates of zero are equivalent to a dead organism, so this is also not what one usually wants.

So what one usually expects from a model is to find elementary flux modes that cover all the reactions and reaction elements.

Running an elementary mode analysis on the current model provides us with only a single elementary mode which consists of just two reactions. The results of the elementary flux mode analysis are depicted in figure 4.12. From the graphical display of the results where the elementary mode is highlighted in red, it is very obvious that the elementary mode consists of the pair of reactions that phosphorylates and dephosphorylates NF- κ B.

This result can also be visualized using the layout information from CellDesigner. In figure 4.12 the elementary mode is highlighted in red. It consists of the reactions from unphosphorylated NF- κ B to phosphorylated NF- κ B and the corresponding reverse reaction.

Most of the elements in the current model are not part of any elementary mode. This means that for these elements no stable steady state can be reached and that maybe essential parts are still missing from the model.

Provided that the desired end result would be to have a model that can not only describe a single signaling event, but also repeated events and/or the systems behavior without any events at all, model needs to be able to reach a steady state. That means that the model has to be modified until this is the case.

Looking at the diagram a first guess as to why most elements can not form a stable steady state can be made. Most elements in the model are part of a linear chain of irreversible reactions which means that all of the initial reactant of such a chain will eventually be converted to the final element in the chain. Once all substrates have been converted to the final products no more reaction events will occur. This can be fixed by making sure that there is always substrate that can react and that products can't accumulate.

This can for example be achieved by providing influx reactions that deliver substrates into the system as well as efflux or degradation reactions that consume the products that would otherwise accumulate.

For this the model is transferred to CellDesigner again and the missing reactions are added. We could also do this in COPASI, but since COPASI does

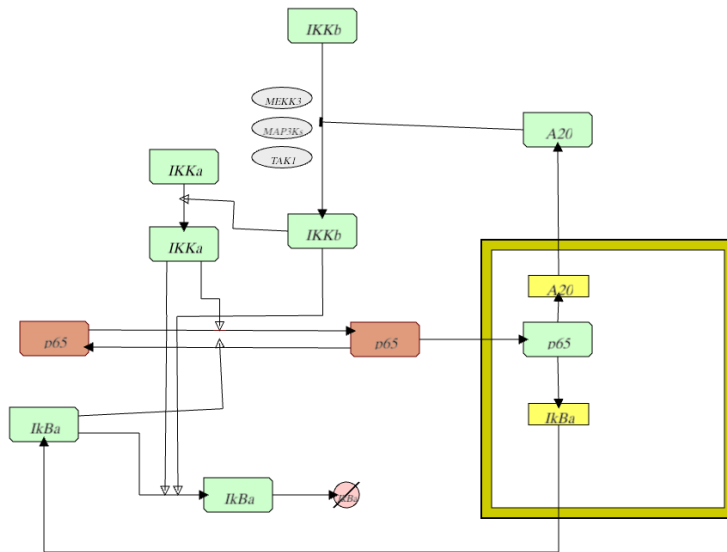
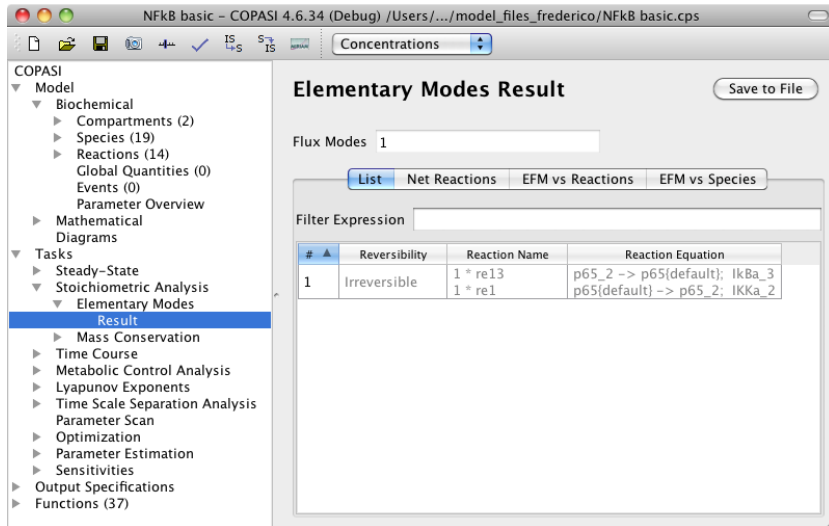


Figure 4.12: Textual (top) and graphical (bottom) result of the elementary flux mode analysis of the initial NF- κ B signaling model. Only a single elementary mode was found which has been highlighted in red in the diagram. The flux mode consists of just two reactions.

not provide any means yet to also create the corresponding visual elements, CellDesigner is the preferred tool for this task.

Figure 4.13 shows the extended model after degradation reactions have been added for the phosphorylated IKKb, phosphorylated IKKa as well as A20. In addition to these degradation reactions, the substrates representing unphosphorylated IKKb, unphosphorylated IKKa as well as unphosphorylated NF- κ B have been set to constant. This means that their concentration will not change during a time course calculation. This is equivalent to adding influx reactions that transport these substances into the system.

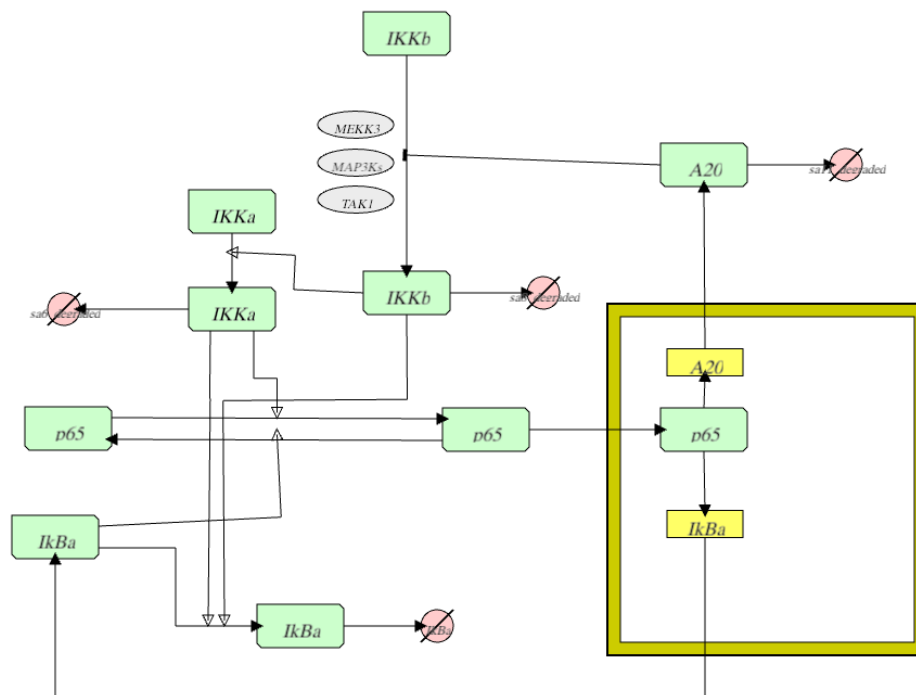


Figure 4.13: Extended NF- κ B model displayed after re-import into COPASI. Three new degradation reactions have been added graphically using CellDesigner.

Rerunning the elementary flux mode analysis now, shows (see figure 4.14) that more elementary modes have been found and by highlighting all elementary modes in the graphical display, it becomes clear that these elementary flux modes cover all reactions of the model. This is something that is not immediately obvious from looking at the textual result which is also provided in figure 4.14. Here being able to display the results graphically provides a clear advantage for the users.

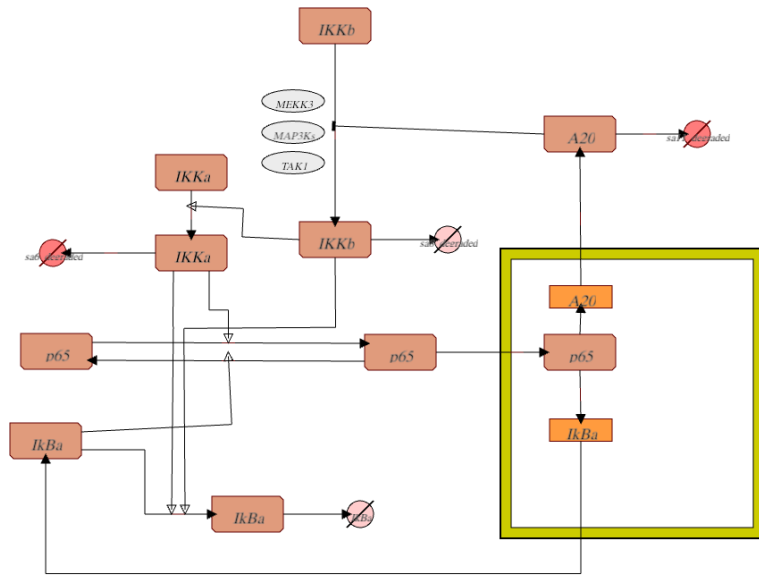
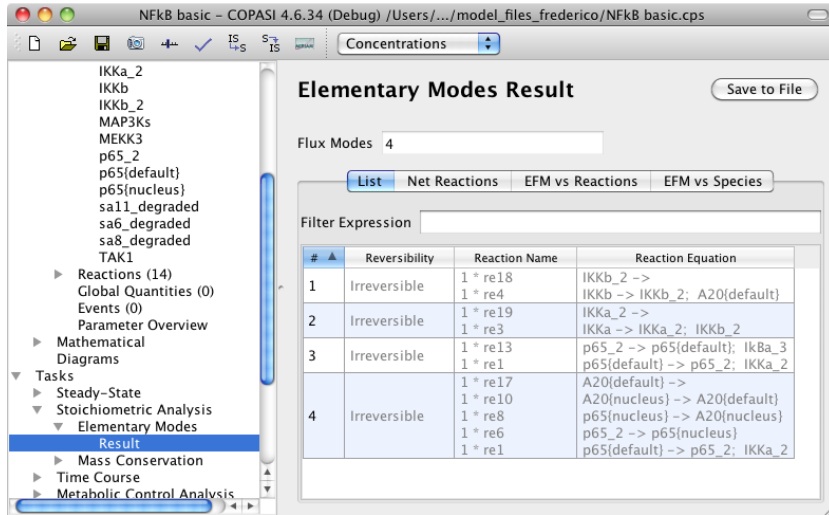


Figure 4.14: Textual (top) and graphical (bottom) result of the elementary flux mode analysis of the extended NF- κ B signaling model. This time four elementary modes have been found covering all of the reactions and elements in the model. In the diagram all flux modes have been highlighted in red.

Now we at least have a model that can potentially describe the systems behavior over a certain time. The tedious work that follows now is to go through the biological databases in order to find suitable initial values for the individual species and suitable rate laws for the different reactions. How this work can be simplified to a certain extent by using a combination of the language bindings described in chapter 4.5, the MIRIAM annotations that were added to the model as well as a framework for the comparison of mathematical expressions (see chapter 5) will be explained in chapter 5.6.

After finding out (see chapter 5.6) that most models from the BioModels database use mass action type kinetics for protein phosphorylation reactions this kinetic rate law is also used in the NF- κ B model. To test if the model really can run into a steady state if it is simulated, estimated initial values have been added and a time course was calculated in COPASI. The time course data for some of the species in the model as it is displayed using COPASIs plotting facility is depicted in figure 4.15. And from the plot it really does seem as if the model eventually runs into a stable steady state just as predicted by the elementary flux mode analysis.

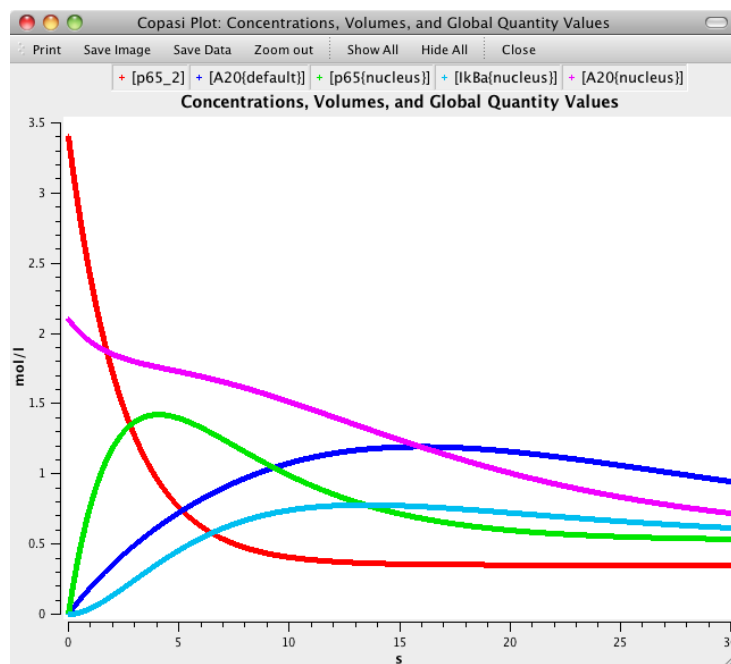


Figure 4.15: Screen shot of a time course data plot for the NF- κ B model in COPASI.

Using the visualization work described in chapter 4.3, the time course data can also be displayed as an animation in the context of the diagram.

Since it is not possible to show an animation here, several frames from the complete animation at certain time points have been saved and are displayed in figure 4.16.

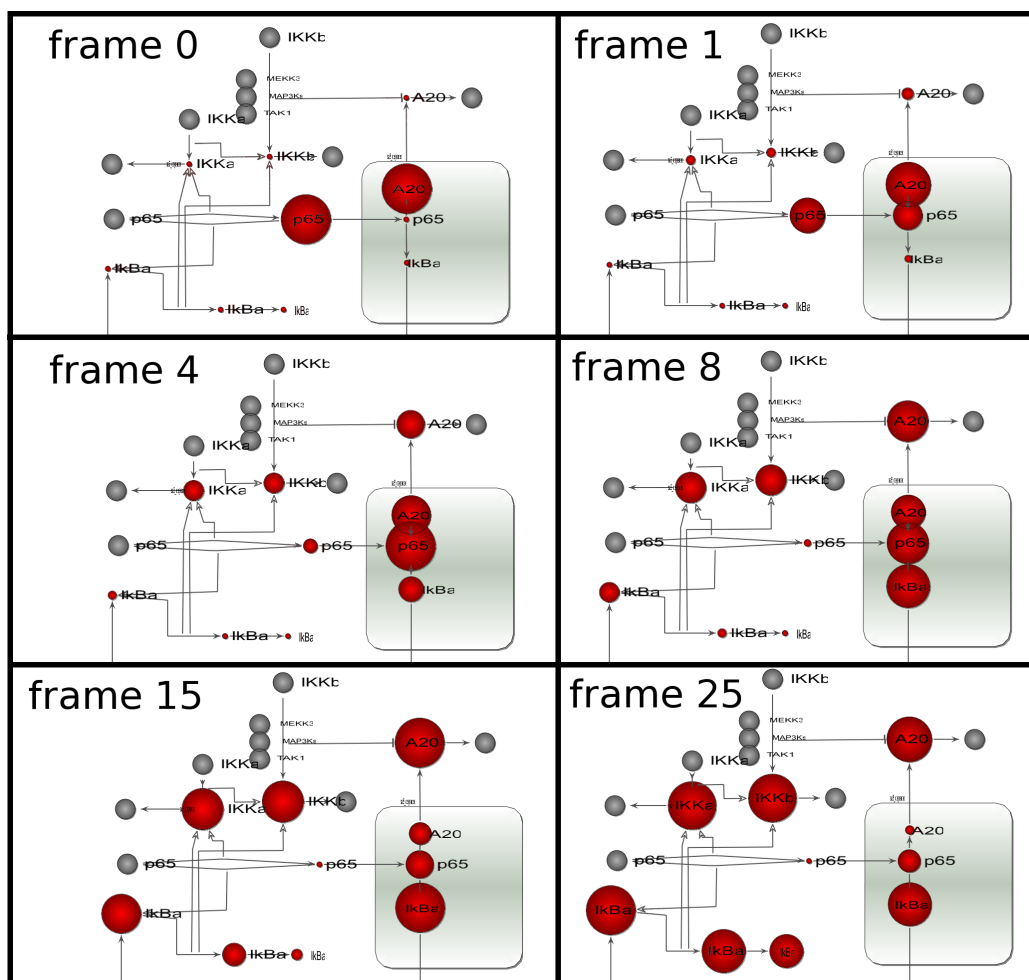


Figure 4.16: Exported animation frames for a time series of the NF- κ B model. The frames display the state of the model at 0, 1, 4, 8, 15 and 25 time units. Variable species are drawn as red spheres and the size of the spheres reflects the concentration of the associated species.

This is ongoing work and we hope that we can use the feedback from our collaboration partners as well as the experiences we make in this project to further improve on the existing methods as well as find new ways of using the graphical display of the reaction network for the visualization of analysis results.

4.7 Work Contributions

COPASI is a large project with several groups participating in its development. While most of the work described in this section has been done by myself, there have been significant contributions by colleagues from within the COPASI project as well as from collaboration partners outside the COPASI project. To highlight contributions to this work by others, the individual sub-projects together with my contribution and work contributed by others have been summarized below:

SBML import/export in COPASI: All the implementation work has been done by myself. This includes import and export of the core SBML specifications as well as import and export of the MIRIAM annotations.

Dr. Sven Sahle provided feedback and fruitful discussions. Dr. Stefan Hoops wrote most of the code necessary to read and write MIRIAM information to COPASI files.

SBML layout extension import/export: I did much of the testing and some bug fixes.

Most of the code was written by Dr. Sven Sahle.

SBML render extension import/export: The complete implementation was written by myself.

Dr. Sven Sahle helped in testing the implementation.

display of layout and render information: The complete implementation was written by myself.

Dr. Sven Sahle helped in testing the implementation.

display of simulation results: I rewrote most of the code and added functionality that was not present in the initial implementation.

Dr. Ursula Rost wrote the initial implementation.

display of elementary modes: The implementation was done by myself.

Dr. Sven Sahle helped in testing.

automatic layout creation: I wrote the code providing input for the layout algorithm and part of the graphical user interface for this feature. I also helped in testing the layout algorithm.

Dr. Sven Sahle wrote the force directed layout algorithm.

COPASI language bindings: The complete implementation was done by myself including documentation and examples.

Dr. Akira Funahashi and Dr. Akiya Jouraku help in testing the code in CellDesigner.

Chapter 5

Comparing And Identifying Mathematical Expressions

Since systems biology deals with mathematical descriptions of biochemical reactions networks, computer programs in this field often have to deal with mathematical expressions which for example describe the kinetic laws of the individual reactions. In order to be as flexible as possible, model description frameworks like SBML allow user to specify arbitrary mathematical expression.

There are several use cases where it would be advantageous if a program could identify the individual mathematical expressions in a model, e.g. to determine if a certain expression represents a certain type of reaction kinetics, like e.g. Michaelis-Menten kinetics.

COPASI for example can do deterministic as well as stochastic time course simulations, but stochastic simulations are only possible if the reaction kinetics of the reaction network consists of only irreversible reaction kinetics, or if the reversible reaction kinetics can be transcribed to irreversible kinetic terms. In COPASI the conversion from a reversible to irreversible kinetic reaction can be done automatically, but only if the kinetic law of the reversible reaction can be identified to be one of the reversible rate laws for which such a conversion has been implemented.

Another feature that could profit from being able to identify or compare expressions is model merging. Sometimes modelers would like to combine several individual models to create a larger model including all aspects of the original models. To achieve this, it is important to be able to identify elements in the individual models that overlap and especially if the models are large, it would be beneficial if this identification could be done automatically. For this, the program would need to be able to determine if e.g. a rule in one model is identical to another rule in a second model. If they are identical

the rule can just be copied to the merged model while in the other case, the user has to be prompted for a decision as to which rule should be used in the merged model.

In order to be able to compare and identify arbitrary mathematical expressions, a software framework has been created that converts mathematical expressions to a general normal form. In many cases, identical expressions when normalized will result in the same normal form and the software can identify this by comparing the normal forms of these expressions.

A very simple example for this would be these two different notations for the change in concentration of substance A by a reversible mass actions kinetics of the reaction $A + B \rightarrow C + D$.

$$\begin{aligned} \frac{d[A]}{dt} &= -k_f * [A] * [B] + k_r * [C] * [D] \\ \frac{d[A]}{dt} &= k_r * [C] * [D] - k_f * [A] * [B] \end{aligned}$$

Although the two equations are written in a slightly different way, it is immediately evident that the first equation can be converted to the second equation by changing the order of the two summands.

At least for a human this is a rather trivial task, but for a computer these two expressions are different expressions until they are converted to a common normal form that can be compared and be identified as being equal.

This is a rather simple example, but mathematical expressions in model files can be arbitrarily complex and the more complex these expressions are, the more difficult it is to normalize and compare them.

5.1 Classes For The Representation Of The Normal Form

The first step towards being able to compare and identify arbitrary mathematical expressions was to define what the normalized form for any mathematical expression should look like.

In discussions between Dr. Sven Sahle, Sarah Lilienthal and myself, it was decided that all expression can be converted to a fraction of sums of products. Which means that the final structure after normalizing a mathematical expression is a fraction with a numerator and a denominator which are sums. The summands of these sums are products of different mathematical entities (see figure 5.1).

$$\frac{f_1 * f_2 * f_3 * \dots + f_4 * f_5 * f_6 * \dots + \dots}{f_7 * f_8 * f_9 * \dots + f_{10} * f_{11} * f_{12} * \dots + \dots}$$

Figure 5.1: Final structured of a normalized mathematical expression: A fraction where the numerator and the denominator consist of a sum of products.

A further restriction was that the fraction may not contain other nested fractions unless those fractions can not be converted to a common denominator.

To support the normalization of mathematical expressions data structures for the individual parts of the normal form as well as data structures needed for intermediate results in the normalization process have been created using the C++ programming language.

A first implementation of these data structures and methods was done by Sarah Lilienthal. Meanwhile this first implementation has been rewritten in large parts and extended by myself. The overall principles and data types however are unchanged.

In the following the individual classes used for the representation and creation of normalized mathematical expressions will be explained in more detail.

An overview over the classes used for the representation of the normal form can be gained from figure 5.2.

All classes of the normal form data structures are derived from one common base class called *CNormalBase*. This class is an abstract class that can not be instantiated and it provides an interface common to all derived classes.

The top level class for the final normal form is the class that represents a fraction called *CNormalFraction*. A *CNormalFraction* consists of two sets of ordered summands. One set represents the numerator and one set represents the denominator (see figure 5.3). The *CNormalFraction* class also contains methods for some mathematical operations as well as methods to query the components and the status of instances of this class.

The *CNormalSum* class represents the individual summands in the numerator and the denominator of the *CNormalFraction* class. In the final normalized form, each instance of the *CNormalSum* class contains an ordered set of one or more products. Intermediate results of the normalization process may also contain instances of *CNormalSum* that contain sets of one

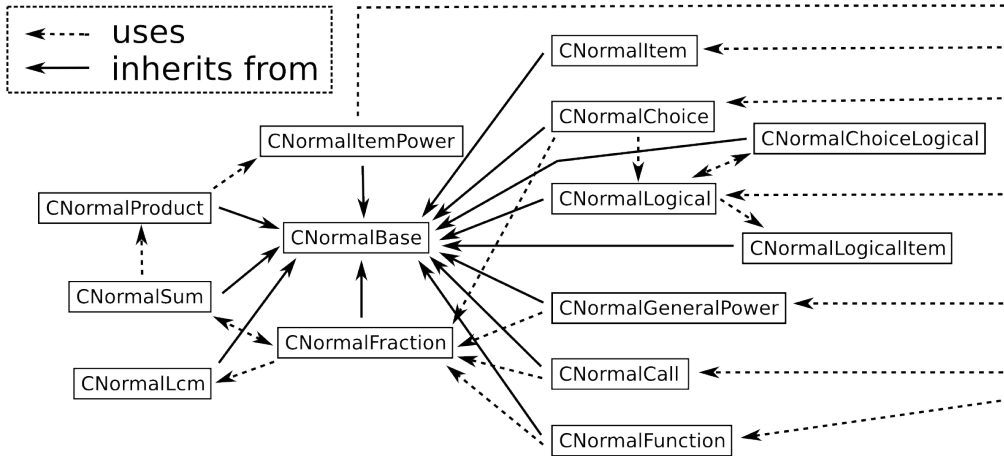


Figure 5.2: Inheritance diagram for the classes used in calculating and representing normal forms of mathematical expressions.

$$\frac{S_1+S_2+S_3+\dots}{S_4+S_5+S_6+\dots}$$

Figure 5.3: Mathematical expression represented by an instance of class *CNormalFraction*. S_n are the individual summands of the numerator and the denominator of the fraction.

or more Fractions in the form of instances of the *CNormalFraction* class (see figure 5.4).

$$P_1 * P_2 * P_3 * \dots * \left(\frac{N_1}{D_1} * \frac{N_2}{D_2} * \dots \right)$$

Figure 5.4: Mathematical expression represented by an instance of class *CNormalSum*. P_n are the individual factors of the product represented by an instance of *CNormalSum*. Intermediate results in the normalization process may contain additional fraction elements. N_n and D_n represent numerators and denominators of these fractions.

The *CNormalSum* class also provides methods for some mathematical operations as well as methods to change and query the current state of instances of the class.

The *CNormalProduct* class is the representation of a product consisting of a numerical factor and a set of ordered mathematical entities of type *CNormalItemPower* (see figure 5.5).

$$N * IP_1 * IP_2 * IP_3 * \dots$$

Figure 5.5: Mathematical expression represented by an instance of class *CNormalProduct*. N represents a numerical value and IP_n stands for an instance of class *CNormalItemPower*.

The class provides methods for some mathematical operations as well as methods to set and query the current state of an instance.

The *CNormalItemPower* class is used to represent a mathematical entity to some numerical power. This means that the class has an attribute to store a number that represents the powers exponent as well as an attribute for the base of the power (see figure 5.6).

Valid objects for the base of a *CNormalItemPower* instance can be instances of *CNormalItem*, *CNormalFunction*, *CNormalGeneralPower*, *CNormalChoice*, *CNormalCall* or *CNormalLogical*.

This class provides mostly methods to change and query the state of instances of the class.

The *CNormalItem* class represents either a variable name or the name of

$$X^N$$

Figure 5.6: Mathematical expression represented by an instance of class *CNormalItemPower*. N represents a numerical value and X stands for an instance of one of the classes *CNormalItem*, *CNormalFunction*, *CNormalGeneralPower*, *CNormalChoice*, *CNormalCall*, *CNormalLogical*.

a mathematical constant, like the number π . The class provides methods to query and change the current state of instances of the class.

CNormalFunction is used to represent calls to predefined functions, e.g. the trigonometric functions `sin`, `cos` or `tan`. The class provides a type attribute which defines which predefined function call is represented as well as an instance of *CNormalFraction* which represents the single argument to the function. Since all predefined functions represented by *CNormalFunction* only expect a single argument, one fraction to represent the argument is enough.

The set of predefined functions corresponds to the functions allowed in SBML models and is described in the SBML specification documents[178, 179, 180, 113, 181]. The class provides methods to query and change the current state of instances of the class.

The *CNormalGeneralPower* is similar to the class *CNormalItemPower* and represents a mathematical power expression with a base and an exponent. In contrast to the *CNormalItemPower* class, the *CNormalGeneralPower* uses an instance of *CNormalFraction* as the base as well as for the exponent (see figure 5.7).

$$\left(\frac{N_{Base}}{D_{Base}}\right)^{\left(\frac{N_{Exponent}}{D_{Exponent}}\right)}$$

Figure 5.7: Mathematical expression represented by an instance of class *CNormalGeneralPower*. The base and the exponent are both instances of *CNormalFraction*. N_{Base} and D_{Base} represent the numerator and the denominator of the base while $N_{Exponent}$ and $D_{Exponent}$ stand for the numerator and the denominator of the exponent.

The *CNormalCall* class is used to represent calls to user defined functions

as opposed to predefined functions which are represented by the *CNormalFunction* class mentioned above. Since user defined functions can have an arbitrary number of arguments, the *CNormalCall* class contains a vector of instances of *CNormalFraction* to represent these arguments.

So far all the classes described are used in mathematical representations of expressions leading to numerical results. In SBML models some expressions can be logical expressions leading to boolean result values (**true** or **false**).

The *CNormalChoice* class represents a choice statement that consists of three parts. The first part is a condition expression that leads to a boolean result, meaning either the value **true** or the value **false**. Depending on the result of the condition, one of the other two expressions is evaluated (see figure 5.8).

$$\begin{cases} 3.0 & \text{if } X < 3.0 \\ X & \text{if } X \geq 3.0 \end{cases}$$

Figure 5.8: Example of a mathematical expression represented by an instance of class *CNormalChoice*. The value 3.0 is returned if the value of X is less than 3.0, otherwise the value of X is returned.

This is equivalent to an

```
if CONDITION then
  DO_SOMETHING
else
  DO_SOMETHING_ELSE
```

statement as it is found in most programming languages.

The attribute representing the condition statement is an instance of class *CNormalLogical* while the two attributes for the **true** and the **false** branch are instances of the *CNormalFraction* class.

The *CNormalLogical* class represents a logical expression as a combination of boolean items combined by the logical operation \wedge (**and**) which are again combined by the logical operation \vee (**or**).

Intermediate results during the normalization process can also contain one or more instances of *CNormalChoiceLogical*.

The *CNormalChoiceLogical* is similar to the class *CNormalChoice* described above. The main difference is that the attributes representing the

$$(B_1 \vee B_2 \vee B_3 \vee \dots) \wedge (B_4 \vee B_5 \vee B_6 \vee \dots) \wedge (B_7 \vee B_8 \vee B_9 \vee \dots) \wedge \dots$$

Figure 5.9: Mathematical expression represented by an instance of class *CNormalLogical*. B_n are the individual boolean elements or elements evaluating to boolean values that are combined in the expression.

`true` and `false` branch of the choice are instances of *CNormalLogical*. So the result of evaluating this expression is a boolean value while the result of evaluating an expression represented by an instance of *CNormalChoice* leads to a numerical result.

$$\begin{cases} \textit{true} & \textit{if } X = \textit{false} \\ \textit{false} & \textit{if } X \neq \textit{false} \end{cases}$$

Figure 5.10: Example of a mathematical expression represented by an instance of class *CNormalChoiceLogical*. The value *true* is returned if the value of X is *false*, otherwise the value *false* is returned.

The *CNormalLogicalItem* is used to represent boolean values of type `true` or `false` as well as comparison expression like $>$ or $=$. If the class is used to represent a comparison expression, the two operands for the comparison are represented by instances of type *CNormalFraction*. If the class is used to represent the boolean values `true` or `false`, these two attributes are unset.

The class *CNormalLcm* is not directly used to represent parts of mathematical expressions. It is used in the multiplication of fractions to determine and store the "least common multiple" of the denominators of two fractions.

5.2 Classes For The Representation Of Expressions In COPASI

The classes and methods described above are used to represent the normal form or intermediate forms that are created during the normalization of mathematical expressions. The actual data structures used in COPASI to represent mathematical expressions are made up of a different set of classes and during the normalization process the mathematical expression is converted several times between these two data structures. Also mathematical

expression from SBML models are first converted to the tree based mathematical data structures described below before they are normalized.

Elements in mathematical expressions in COPASI are stored in so called nodes which are arranged in a tree like structure where each node can have children and those children again can have other children. Since nodes can have several children, but only one parent, this automatically leads to a tree with a single so called root node at one end and one or more leaf nodes (nodes without children) on the other ends (see figure 5.12). Whether a node can hold children or not is determined by the type of the node. The different node types present in COPASI are described below:

The base class for all nodes in such an expression tree is the class *CEvaluationNode*. *CEvaluationNode* is derived from the class *CCopasiNode*. This class defines a data structure which can store zero or more children and can have a link to a single parent. In addition to that, each node can store some data value.

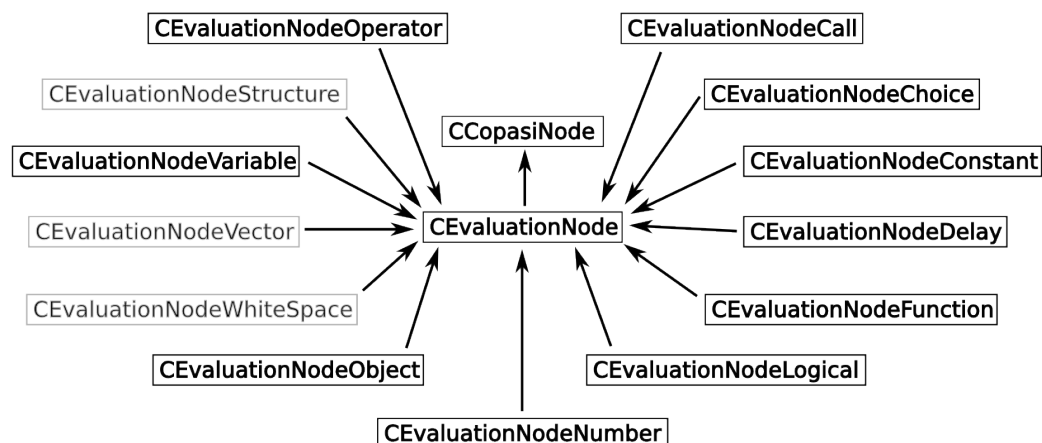


Figure 5.11: Inheritance tree for the expression tree node classes in COPASI. Since theoretically each class can use all other classes, usage information has not been included in the diagram. Classes not used in the expression normalization are marked in gray.

Methods implemented in *CCopasiNode* are mostly limited to changing the state and the content of individual instances as well as for the management any querying of children or ancestors of node instances.

The class is implemented as a template with the type of information that is stored as the template parameter. This way it is possible to create

node types that store textual information as well as node types that store numerical information, by specifying the desired data type.

CEvaluationNode is a direct subclass of *CCopasiNode* and it instantiates the data type to be a string, so all nodes in a mathematical expression tree store their information in the form of a string.

Just like the class framework for the representation of the normal form, the tree data structure uses different node types to represent different mathematical entities (see figure 5.11).

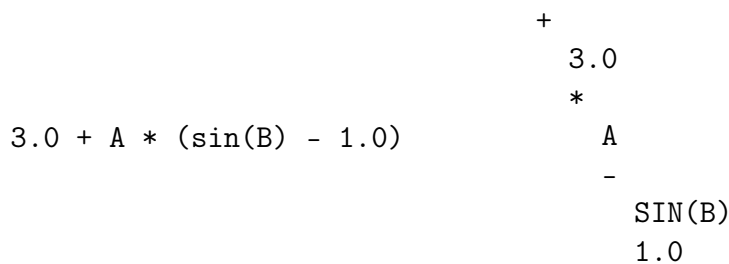


Figure 5.12: Example of a mathematical expression (left) and the corresponding expression tree (right) as it would be represented in COPASI. The different nodes in the tree are represented by different subclasses of *CEvaluationNode*.

The class *CEvaluationNodeNumber* is used to represent numerical values in the expression tree. The class distinguishes between integer numbers, double value numbers, numbers in exponent notation where the number is split into a mantissa and an exponent as well as rational numbers with a numerator and a denominator.

The class *CEvaluationNodeConstant* is used to represent different constants. The constants that can be represented by instances of this class are listed in table 5.1.

π	Euler's number e
<i>true</i>	<i>false</i>
infinity (∞)	Not a number (<i>NaN</i>)

Table 5.1: Constant types that can be represented by instances of the class *CEvaluationNodeConstant*. *NaN* is a term in computer numerics to represent an invalid numerical value, as it is e.g. created by the evaluation of $\frac{0}{0}$.

The class *CEvaluationNodeVariable* is used to represent named variables in an expression. Variables occur in function definitions as the parameters of the functions. E.g. in the function definition $f(x) = 3 * x$, x would be represented by an instance of *CEvaluationNodeVariable*.

The class *CEvaluationNodeObject* represents references to numerical values associated with entities from the model. E.g. an instance of *CEvaluationNodeObject* would be used to represent the volume of a compartment or the concentrations of a metabolite from the reaction network within a mathematical expression.

The class *CEvaluationNodeOperator* is used to represent the mathematical operations for addition, subtraction, multiplication, division, exponentiation and the modulo operation (remainder of an integer division).

The class *CEvaluationNodeFunction* corresponds to the *CNormalFunction* class described above. It is used to represent predefined function calls, e.g. calls to trigonometric functions, in an expression tree. The functions that can be represented by instances of this class are listed in table 5.2. The types of functions that can be represented by this node class mostly corresponds to the functions allowed in mathematical expression for SBML models. There are a few minor differences however, e.g. COPASI knows two functions for the generation of random values from different distributions *RUNIFORM* and *RNORMAL* which are not available in SBML.

LOG	LOG10	EXP	SIN	COS
TAN	SEC	CSC	COT	SINH
COSH	TANH	SECH	CSCH	COTH
ARCSIN	ARCCOS	ARCTAN	ARCSEC	ARCCSC
ARCCOT	ARCSINH	ARCCOSH	ARCTANH	ARCSECH
ARCCSCH	ARCCOTH	SQRT	ABS	FLOOR
CEIL	FACTORIAL	MINUS	PLUS	NOT
RUNIFORM	RNORMAL			

Table 5.2: List of functions that can be represented by instances of the class *CEvaluationNodeFunction*.

The class *CEvaluationNodeCall* corresponds to the class *CNormalCall* described above. Instances of this class are used to represent calls to user

defined functions.

The class *CEvaluationNodeChoice* corresponds to the classes *CNormalChoice* and *CNormalChoiceLogical* described above. It represents a conditional expression that can either evaluate to the value *true* or to the value *false*. Depending on the evaluation result of the condition, one of two child node expressions is evaluated (see figure 5.13).

$$IF(X < 5, X, 5)$$

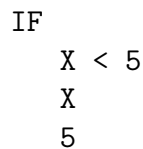


Figure 5.13: Example for the representation of a choice node. The choice node is represented by the string `IF` and it contains three child nodes. The condition is $X < 5$ which means that it evaluates to *true* if the value of X is less than 5 and to *false* otherwise. In the case it evaluates to *true*, the choice node returns the value of the second child, which is X . If the condition evaluates to *false* the value of the third child is returned, which in this example is represented by the numerical value 5.

The *CEvaluationNodeLogical* is used to represent logical operations or comparisons. The result of such a logical operation is either the boolean value *true* or the boolean value *false*. Nodes of type *CEvaluationNodeLogical* are for example used as the condition element in an instance of *CEvaluationNodeChoice*. The operations that can be described by instances of this class are listed in table 5.3.

logical or (\vee)	logical exclusive or (\oplus)	logical and (\wedge)
equal ($=$)	not equal (\neq)	greater $>$
greater or equal (\geq)	less ($<$)	less or equal (\leq)

Table 5.3: Logical and comparison operations that can be represented by instances of class *CEvaluationNodeLogical*.

Instances of the class *CEvaluationNodeDelay* are used to model the delay function as specified in the SBML specification. The delay function takes two arguments, one is an expression that evaluates to a number that stands

for a time delay t , the other is an expression that is evaluated for a time t time units prior to the current time.

There are some more subclasses of the class *CEvaluationNode*, which have been marked in gray in the class inheritance diagram (see figure 5.11). Since those are not used in the normalization of mathematical expressions, so they won't be described here.

5.3 Normalization Algorithm

The normalization process for any mathematical expression starts with a *CEvaluationNode* based expression tree as described above. libSBML which is used to load SBML files also has a node tree based format for mathematical expressions and all expressions from SBML model files are first converted to the COPASI tree representation. For this a number of methods have been written which are also used for the import of SBML model files into COPASI.

Since many steps of the normalization process are easier to do on the tree like expression representation, a large part of the normalization process is done on the nodes of the tree while other steps of the normalization process are executed on the normal form representation (see chapter 5.1). The normalization process is an iterative process and in the course of a single normalization the expression is converted several times between these two formats. For the conversion from the *CEvaluationNode* based expression representation to the *CNormalBase* representation and vice versa special methods have been written.

Function Call Expansion

COPASI as well as SBML model files can contain user defined function definitions. So the first step when normalizing an expression is to eliminate calls to these user defined functions. For this, all calls to user defined functions are replaced by the expression of the function. All function variables in that expressions have to be replaced by the arguments to the function from the function call.

Since function definitions can call other function definitions, this process has to be repeated until there are no more calls to user defined functions in the expression (see figure 5.14).

This expansion of the expression is done on the *CEvaluationNode* based representation and the result of the expansion is again represented by a *CEvaluationNode* based expression tree.

$$\begin{aligned}
& \text{Function definitions:} \\
& f(x) = 3.0 * x \\
& g(y) = f(y)^{2.0} \\
\\
& 4.0 * g(a) + f(b) \\
\rightarrow & 4.0 * f(a)^{2.0} + 3.0 * b \\
\rightarrow & 4.0 * (3.0 * a)^{2.0} + 3.0 * b
\end{aligned}$$

Figure 5.14: Example of a function call elimination. The expression calls contains two function calls, f and g . In the first elimination step both calls are replaced by the expressions of the corresponding function definitions and the function variables are replaced by the arguments to these function calls. Since the function g calls the function f , the expanded code still contains a function call, so the expression needs to be expanded a second time to finally get the fully expanded expression.

Only normal forms of expressions expanded in this way can be compared reliably.

Normalization Iteration

The expanded expression tree is then simplified and normalized repeatedly either until the infix (meaning the string representing the expression) has not changed between two iterations of the simplification and normalization process or until a certain iteration limit has been reached.

Since on the one hand mathematical expressions can be arbitrarily complex and on the other hand the algorithm is not guaranteed to converge for all possible mathematical expressions, the algorithm will not necessarily stop by itself for any given expressions. For this reason a limit for the number of iterations that are done has been added to the algorithm which guarantees that the algorithm will not spent infinite time on the normalization of any expression. The current iteration limit for this top level iteration is 20. This value seems to provide a good compromise between allowing the algorithm to converge for most expressions while making certain that it does stop within a reasonable time frame.

Each iteration consists of four distinct processing steps (see figure 5.15).

The algorithm starts out with the tree representation and simplifies this tree as much as possible. For the result of this simplification, the exponents of all power nodes are expanded again and the result is converted to the *CNormalBase* representation which is then again simplified.

Since the power node expansion is also done during the simplification of the tree based expression, the details of this function are explained in that context.

Those parts of the simplification process that are easier done on the tree are done on the tree and those the are easier to do on the normal form representation are done there.

The result of this process is an instance of the class *CNormalFraction*.

```

repeat
  - store infix
  - simplify tree based expression
  - expand power nodes on simplified tree
  - convert tree based expression to CNormalFraction
  - simplify fraction
until (stored infix = infix from fraction  $\vee$  number of iterations > iteration limit)

```

Figure 5.15: Steps of the normalization of mathematical expressions.

In the following paragraphs, the four steps from figure 5.15 will be explained in more detail. First we will have a look at the simplification of the tree based expression representation.

Simplification Of The Tree Based Expression

The overall algorithm for this step is similar to the iterative process described above. A number of simplification steps are executed on the nodes of the tree based expression representation and afterwards, the infix of the expression is compared to the infix as it was before these simplification steps. If the two string representations are identical, the algorithm assumes that the process of simplifying the expression is finished and stops, otherwise it continues with another round of the same simplification steps.

This either continues until no more changes to the tree have occurred within one iteration or if a certain iteration limit has been reached. Currently this iteration limit is the same as for the overall process which is 20 iterations.

The simplification steps per iteration are the following:

repeat

- elimination of certain structures
- evaluation of operations on numerical nodes
- collecting identical elements in addition/subtraction and multiplication/division chains
- expansion of power bases
- expansion of power nodes
- expansion of products

until (stored infix = infix from fraction \vee number of iterations $>$ iteration limit)

Eliminations

The elimination process gets rid of several undesired structures again in an iterative way, meaning the process is repeated until there are no more changes to the expression between two iterations.

The first thing the method does is to eliminate some unnecessary elements like multiplications or divisions by the number 1 as well as additions and/or subtractions of the number 0.

The next thing it does it to eliminate nested powers. E.g. the term A^{x^y} is converted to the term $A^{(x*y)}$.

Another structure that is converted are powers of fractions. A fraction to a power is converted to a fraction where the numerator and the denominator are each taken to that power. E.g. $\left(\frac{A}{B}\right)^5$ is converted to $\frac{A^5}{B^5}$.

Another elimination is done on nested fractions. Nested fractions are converted to a single fractions. E.g. the term $\frac{\frac{A}{B}}{C}$ is converted to the term $\frac{A}{B*C}$.

And last but not least, the method collects identical elements in the resulting expressions as explained below in the paragraph on "collecting identical elements".

Evaluation Of Operations On Number Nodes

In expressions one often finds operations that act only on numbers, e.g. something like the term $2.0 + 3.0/6.0$. The algorithm searches for such operations, evaluates them and replaces the complete subexpression by the resulting number (see figure 5.16).

Collecting Identical Elements

This methods tries to identify chains of additions/subtractions and/or chains of multiplications/divisions. It then tries to identify elements in these chains

$A * 4.0 * (3.0 - 1.0)$	$A + 8.0$
$+$	$+$
A	A
$*$	$+$
4.0	8.0
$-$	
3.0	
1.0	

Figure 5.16: Example of the evaluation of operation on numbers. A complete subexpression of the tree is replaced by a single number after the expression has been evaluated.

that cancel each other out, either partially or completely. E.g. a multiplication would be canceled by a division with the identical term, or an additions would be canceled out by a subtraction of the same term (see figure 5.17).

$3.0 * A * C^2 / (B * C)$	$2.4 - x + T + x$
$\rightarrow 3.0 * A * C / B$	$\rightarrow 2.4 + T$

Figure 5.17: Example of a multiplication (left) and an addition (right) where terms cancel each other out. In the second addition example two terms cancel each other out completely and can therefore be eliminated.

For multiplication chains this method also considers the exponents of potential power nodes. E.g. if in a multiplication/division chain a factor A^4 is found and in the same chain there is a divisor A^3 , these two nodes would be replaced by the term A .

Expansion Of Power Bases

This method expands products to some power to the corresponding products of the individual factors to the same power.

E.g. the expression $(A * B)^x$ would be converted to the new expression $A^x * B^x$

Expansion Of Power Nodes

This method converts nodes representing a power operation where the exponent is a sum into a product of two or more power nodes where the exponents

are no longer sums.

An example for this would be the term $A^{(x+y)}$ which would be converted to $A^x * A^y$.

Expansion Of Products

This method expands products of sums into sums of products.

E.g. the term $(A + B) * (C + D)$ would be expanded to the new term $(A * C) + (A * D) + (B * C) + (B * D)$.

Once the expression tree has been simplified this way power nodes are expanded again before the expression is converted to the *CNormalBase* based representation. This additional final expansion of the power nodes, simplifies the following conversion to the normal form. The resulting instance of *CNormalFraction* is now again subjected to a number of simplification steps that are supposed to lead to the final normalized form.

The steps undertaken for the simplification and normalization of this second representation greatly depends on the element that is being processed. Each class has a method called `simplify` which is used to further rearrange the mathematical representation to create the final final normal form. Usually each class will call the `simplify` method for all instances of other elements it contains. E.g. the fraction representation will call `simplify` on its numerator and its denominator.

In the following paragraphs the processing steps for the individual classes are described.

CNormalFraction

In the `simplify` method for *CNormalFraction* the corresponding methods for the numerator and the denominator are called which are both instances of *CNormalSum*.

If the instances of *CNormalSum* of the numerator and the denominator still contain fractions in addition to the products, the method expands these fractions and converts them to new instances of *CNormalSum* which are added to the numerator and the denominator accordingly.

Afterwards the method tries to identify common factors in the numerator and the denominator and cancels those.

CNormalSum

If the instance of *CNormalSum* still contains fractions, these fractions are simplified first. Next the algorithm traverses all instances of *CNormalProduct* and simplifies them as well.

The method also tries to eliminate nested fractions as might be present in items of type *CNormalGeneralPower* in the instances of *CNormalProduct*. By converting the *CNormalGeneralPower* to an instance of *CNormalFraction*, the fraction can be eliminated by the *simplify* method of the parent fraction instance in the following iteration (see figure 5.18).

$$\begin{aligned} & \dots + \left(\frac{N_b \left(\frac{N_e}{B_e} \right)^n}{D_b} \right) + \dots \\ & \rightarrow \dots + \frac{N_b^{(n * \frac{N_e}{D_e})}}{D_b^{(n * \frac{N_e}{D_e})}} + \dots \end{aligned}$$

Figure 5.18: Conversion of an instance of *CNormalGeneralPower* in a *CNormalSum* to a *CNormalFraction*, which can be eliminated by the *simplify* method of the fraction object containing the instance of *CNormalSum*.

CNormalProduct

The *CNormalProduct* class first calls *simplify* on all instances of *CNormalItemPower*. Next the method traverses all items and checks for items which are instances of type *CNormalGeneralPower*. If such an instance is found and that instance has a base of 1, is eliminated otherwise it is combined with the other instances of *CNormalGeneralPower* in the product. This combination of *CNormalGeneralPower* instances again leads to a *CNormalGeneralPower* which might be simplified further in the next iteration.

CNormalItemPower

Instances of *CNormalItemPower* only *simplify* the item they contain. Since the item, can be an instance of one of several classes, the corresponding *simplify* method for that class is called.

CNormalItem

Instances of *CNormalItem* are not simplified.

CNormalFunction

Instances of *CNormalFunction* simplify their argument which is an instance of *CNormalFraction*.

CNormalCall

Instances of *CNormalFunction* simplify all their arguments which are instances of the *CNormalFraction* class.

CNormalGeneralPower

Instances of *CNormalGeneralPower* simplify the two *CNormalFraction* instances representing the base and the exponent.

CNormalChoice

Instances of the class *CNormalChoice* simplify the condition which is an instance of *CNormalLogical* and the two result branches which are both instances of *CNormalFraction*.

CNormalLogical

The simplification for instances of *CNormalLogical* is rather complex. Since intermediate results can contain instances of *CNormalChoiceLogical* those have to be eliminated first. This is done by replacing each choice element $\text{IF}(\text{COND}, \text{TRUE-term}, \text{FALSE-term})$ by the term $(\text{COND} \wedge \text{TRUE-term}) \vee (\neg \text{COND} \wedge \text{FALSE-term})$.

For the creation of the disjunctive normal form in the last step, it is also necessary to eliminate all negated elements, meaning all elements to which the logical not (\neg) operation is applied.

Last but not least the sets of logical items in the instance of *CNormalLogical* are converted to the canonical disjunctive normal form [182, 183] to make it comparable to other logical expressions.

This conversion to the canonical disjunctive normal form scales exponentially with the number of logical elements. For this reason, we limit the

number of elements that are allowed to 16. Due to this, expressions containing more than 16 elements can not be normalized by the current version of this framework.

CNormalChoiceLogical

Instances of the class *CNormalChoiceLogical* simplify the condition which is an instance of *CNormalLogical* and the two result branches which are also both instances of *CNormalLogical*.

CNormalLogicalItem

For the normal form, instances of class *CNormalLogicalItem* which represent the greater ($>$) relation are converted to the less ($<$) relations and instances representing the "greater or equal" (\geq) relation are converted to the "less or equal" (\leq) relation. During this conversion the two arguments have to be switched.

If the instance of *CNormalLogicalItem* contains arguments, e.g. a relation, these arguments, which are instances of *CNormalFraction*, are simplified.

5.4 Testing

Since the task of normalizing expressions is very complex an extensive set of test cases has been implemented to ensure that the implementation works as expected. The test cases cover all aspects of the normalization process and where possible, systematic tests have been implemented. Because the numbers of possible expressions is infinite, it is obvious that these tests will never be able to cover all possibilities.

At the time of writing there were 112 tests cases of varying complexity.

All of these tests have been implemented using the CppUnit unit testing framework[92].

5.5 Normalization And Identification Of Expressions From The BioModels Database

In order to see how well the expression normalization and comparison methods work in a realistic scenario, a program was written that takes all curated models from the BioModels database and normalizes all mathematical expressions found in these models. The program also tries to identify these

expressions by comparing their resulting normal form to the normal forms of all entries in the function database of COPASI.

First the program normalizes all function definitions from the COPASI function database. This function database contains about 40 functions that can be used as kinetic laws for reactions in COPASI.

For each of the model files, first the user defined functions are normalized and compared to those normalized expressions from the COPASI function database in order to identify them.

Besides function definitions, expressions can occur in several places in an SBML model. The program normalizes expressions from rules, initial assignments, kinetic laws, stoichiometric expressions as well as triggers, delays and assignments from events.

For each expression normalization the computation time needed is recored to check the performance of the algorithm.

All normalized kinetic expressions from reactions in the model files are compared to normalized expression for function definitions in order to identify kinetic laws.

For the identification of mass action kinetic terms, a special method is used. The reason for this is that mass action kinetics does not describe a single mathematical expression, but a complete family of mathematical expressions that depends on whether a reaction is reversible or not and on the number of substrates and products of the reaction.

While running this test on all curated models from the BioModels release from September of 2010, we found that the test did not finish but it stopped with an error while processing Biomodels file 217. The reason for this is discussed in chapter 6.3. Removing this model file from the test allows the test to finish. Running the test on the remaining 268 SBML model files using a non-optimized version of the algorithm takes approximately 2 hours on a single core of a modern CPU.

Figure 5.19 shows part of the output from the analysis.

The information that can be extracted from this output is that the 268 models from the BioModels database contain a total of 9578 mathematical expression that have been normalized. This shows that the algorithm itself is already very stable.

From these 9578 expression 75 could not be converted to a normal form because the iteration limit built into the algorithm has been exceeded. Since this is less than 1% of the total number of expressions, the performance of the algorithm is good.

Of the 9578 expressions processed, the vast majority are expressions for kinetic laws (8055). Of those 8055 kinetic expressions, the algorithm was able to identify 3232, which leaves 4823 kinetic expressions unidentified. This


```

...
number of COPASI function definitions: 35
number of exceeded COPASI function definitions: 0
number of failed COPASI function definitions: 0
268 files have been processed.
number of function definitions: 179
number of exceeded function definitions: 6
number of failed function definitions: 0
number of expressions: 9578
number of exceeded expressions: 75
number of failed expressions: 0
The functions "Catalytic activation (rev)" and "Noncompetitive inhibition (rev)"
in the COPASI database are equal.
The functions "Competitive inhibition (rev)" and "Specific activation (rev)"
in the COPASI database are equal.
The functions "Constant flux (irreversible)" and "Constant flux (reversible)"
in the COPASI database are equal.
Number of function definitions that could be classified: 28
Number of function definitions that were classified incorrectly: 3
Number of function definitions that could not be classified: 142
Number of kinetic expressions: 8055
Number of kinetic expressions that could be mapped to a function definition: 3232
Number of kinetic expressions that could not be mapped to a function definition: 4823
List of the number of expressions mapped to a certain function definition:
Catalytic activation (irrev) : 3
Competitive inhibition (irr) : 1
Constant Flux : 83
Constant flux (irreversible) : 36
Henri-Michaelis-Menten (irreversible) : 12
Hill Cooperativity : 4
Mass Action : 3092
Specific activation (irrev) : 1
There are 10 different SBO Terms.
There are 2 expressions for SBO term 28.
There are 2 expressions for SBO term 47.
There are 22 expressions for SBO term 49.
There are 3 expressions for SBO term 54.
There are 2 expressions for SBO term 101.
There are 2 expressions for SBO term 103.
There are 2 expressions for SBO term 260.
There are 2 expressions for SBO term 270.
There are 1 expressions for SBO term 277.
There are 1 expressions for SBO term 432.
Number of kinetic expressions with sbo terms: 39
Number of kinetic expressions with sbo terms that could not be normalized to the same normalform: 10
The expressions that could not be mapped are divided into 733 different expressions.
...

```

Figure 5.19: Excerpt from the output generated by running the normalization test program on 268 model files from the BioModels database.

value is actually lower than expected and the reasons for this are also discussed in chapter 6.3.

From the kinetic expression that could be identified, the vast majority were mass action kinetics with 3092 instances.

Even in a real world test, the framework showed very promising results, although some rough edges still have to be smoothed out.

Out of almost 10000 mathematical expressions, only a single one lead to an error and less than 1% could not be normalized because the normalization process exceeded the iteration limit.

Besides being able to normalize and compare most expressions, many of the expressions could also be identified by this framework.

Further steps that are needed to make the framework work better with very large expressions and ways of improving the identification rate of expressions are discussed in chapter 6.3.

5.6 Identifying Kinetic Laws For Certain Reaction Types In The NF- κ B Model

Although fully automated classification of e.g. the kinetic laws of a model is not possible yet, the normalization and comparison functionality can be used for the identification of mathematical expression in a semi automatic way. This will be demonstrated here using the NF- κ B model introduced in chapters 3.8 and 4.6.

In order to be able to calculate time course simulation data for the NF- κ B signaling model, initial values have to be assigned to all components of the model and kinetic laws have to be associated with all of the reactions.

The model contains several phosphorylation reactions, e.g. the phosphorylation of NF- κ B which are all catalyzed by so called protein serine/threonine kinases. In chapter 4.6 the reactions have been annotated with the corresponding term from the Gene Ontology[184, 185] which is GO:0004674. This term is derived from the more general term for the concept of a protein kinase activity (GO:0004672). The Gene Ontology entries are arranged in a multi rooted tree structure and all concepts derived from or described by another concept are stored as child elements to that parent concept. The entry for the protein serine/threonine kinase has for example 22 children that further refine the concept, e.g. one of the children describes the concept of a "histone serine kinase activity" (GO:0035174).

Now in order to assign kinetic laws to the phosphorylation reactions in the model, the annotations introduced in chapter 4.6 are used in combination

with the expression normalization framework.

First all Gene Ontology identifiers for the protein serine/threonine kinase and those from all its child elements are extracted from Gene Ontology and used to search for equivalent reactions in the models from the BioModels database.

The extractions of the Gene Ontology identifiers provided 72 identifiers that describe the process of phosphorylating a protein at a serine or threonine side chain. Next the model files were search for reactions that are associated with one of these identifiers. This search revealed 29 models from the curated branch of the BioModels release from April 2011 which contained a total of 140 reactions describing a phosphorylation by a serine/threonine kinase.

Now the expression normalization and comparison was used to analyze the kinetic laws of these reactions to see what types of kinetic laws have been used for these reactions in other models. Parts of the output from this analysis is shown below:

```

number of expressions: 140
number of exceeded expressions: 2
number of failed expressions: 0
Number of kinetic expressions: 138
Number of kinetic expressions that could be mapped to a function definition: 84
Number of kinetic expressions that could not be mapped to a function definition: 54
List of the number of expressions mapped to a certain function definition:
Mass Action : 84
Expression with more than 5 instances (9): (C_1 * K_1 * S_1 * S_2)/(K_2 + S_2)
Expression with more than 5 instances (7): C_1 * K_1 * S_1 * S_2
Expression with more than 5 instances (6): C_1 * K_1 * S_1
Expression with more than 5 instances (6): K_1 * S_1
Expression with more than 1 instances (5): (C_1 * K_1 * S_1)/(K_3 * S_1 +
601.999843480040681 * K_2 * K_3)
Expression with more than 1 instances (5): (K_1 * S_1 * S_2)/(K_2 + S_2)
Expression with more than 1 instances (4): (-1) * C_1 * K_1 * S_1 * S_2 +
C_1 * K_1 * K_2 * S_1
Expression with more than 1 instances (3): ((-1) * C_1 * K_1 * K_2 * K_3 * S_3 +
C_1 * K_1 * K_2 * S_1 * S_2 + C_1 * K_2 * S_1 * S_2)/(K_3)
Expression with more than 1 instances (2): (K_1 * S_1)/(K_2 + S_1)
Unique Expression: (-1) * C_1 * K_2 * S_3 + (-1) * C_1 * K_3 * S_3 + C_1 * K_1 * S_1 * S_2
Unique Expression: (C_1 * K_1 * K_4 * S_1 * S_2 + C_1 * K_1 * S_1 * S_2^2 +
C_1 * K_2 * K_3 * S_2 * S_3 +
C_1 * K_3 * S_2^2 * S_3)/(K_2 * K_4 + K_2 * S_2 + K_4 * S_2 + S_2^2)
Unique Expression: (-1) * C_1 * K_2 * S_2 + C_1 * K_1 * S_1
Unique Expression: (C_1 * K_1 * S_1 * S_2^2)/(S_2^2 + 10000)
Unique Expression: (C_1 * K_1 * S_1 * (K_2)^(K_3))/(K_4 * (K_2)^(K_3) + K_4 * (S_2)^(K_3) +
S_1 * (K_2)^(K_3) + S_1 * (S_2)^(K_3))
Unique Expression: (C_1 * K_1 * S_1^2)/(S_1^2 + 16)
Unique Expression: (-1) * C_1 * K_3 * S_3 + C_1 * K_1 * S_2 + C_1 * K_2 * S_1 * S_2

```

The complete analysis of all 140 kinetic laws took about 16 seconds on a single processor of a modern computer. This included the time needed to read, check and convert the models from the SBML format to the COPASI format.

What can be seen from the output above, out of the 140 kinetic laws, 138 could be normalized. For two expressions the iteration limit has been exceeded. Out of the 138 expression that could be normalized, 84 were identified as mass action kinetics. The remaining 54 expressions are partitioned into nine clusters which contain more than one expression and an additional 7 expressions that are unique, i.e. appear only once in the set of expressions that was examined.

For better readability, the identifiers of the expression elements have been normalized. All elements that represent compartment volumes have been renamed to C_n , all elements representing species amount or concentration to S_n and all parameter elements to K_n . This makes the manual inspections of the expressions easier. In this case it can be seen that many of the unidentified expressions seem to represent mass action kinetics in addition to some some Michaelis-Menten type kinetics. The predominant kinetic law used for this type of phosphorylation reaction therefore seems to be mass action.

In order to see if other phosphorylation reaction types would lead to a different result, all Gene Ontology identifiers related to the "protein kinase activity concept" were extracted from the Gene Ontology and subjected to the same analysis as described above. This time 135 identifiers were found. This set includes all the identifiers from the first analysis. Using these 135 Gene Ontology terms, 47 models were identified with 245 expressions describing some kind of protein phosphorylation. Parts of the analysis output is shown below:

```
47 files have been processed.
number of expressions: 245
number of exceeded expressions: 2
number of failed expressions: 0
Number of kinetic expressions: 243
Number of kinetic expressions that could be mapped to a function definition: 140
Number of kinetic expressions that could not be mapped to a function definition: 103
List of the number of expressions mapped to a certain function definition:
Mass Action : 140
Expression with more than 10 instances (20): C_1 * K_1 * S_1 * S_2
Expression with more than 10 instances (18): (C_1 * K_1 * S_1 * S_2)/(K_2 + S_2)
Expression with more than 10 instances (8): C_1 * K_1 * S_1
Expression with more than 10 instances (8): K_1 * S_1
Expression with more than 10 instances (6): (-1) * C_1 * K_1 * S_1 * S_2 +
C_1 * K_1 * K_2 * S_1
Expression with more than 10 instances (6): (K_1 * S_1 * S_2)/(K_2 + S_2)
Expression with more than 10 instances (6): (C_1 * K_1 * K_3 * S_1 * S_2)/
(K_2 * K_3 + K_2 * S_3 + K_3 * S_2)
Expression with more than 10 instances (5): (C_1 * K_1 * S_1)/(K_3 * S_1 +
601.999843480040681 * K_2 * K_3)
Expression with more than 10 instances (3): ((-1) * C_1 * K_1 * K_2 * K_3 * S_3 +
C_1 * K_1 * K_2 * S_1 * S_2 + C_1 * K_2 * S_1 * S_2)/(K_3)
Expression with more than 10 instances (2): (K_1 * S_1)/(K_2 + S_1)
Expression with more than 10 instances (2): (-1) * C_1 * K_3 * S_2 * S_3 +
C_1 * K_1 * K_2 * S_1
Expression with more than 10 instances (2): (C_1 * K_1 * K_3 * K_4 * K_5 * S_1 * S_2)/
```

```

(K_2 * K_3 * K_4 * K_5 + K_2 * K_3 * K_4 * S_4 + K_2 * K_3 * K_5 * S_3 +
K_2 * K_4 * K_5 * S_3 + K_3 * K_4 * K_5 * S_2)
Expression with more than 10 instances (2): (C_1 * K_1 * K_3 * K_4 * K_5 * S_1 * S_2)/
(K_2 * K_3 * K_4 * K_5 + K_2 * K_3 * K_4 * S_4 + K_2 * K_3 * K_5 * S_3 +
K_2 * K_4 * K_5 * S_2 + K_3 * K_4 * K_5 * S_2)

```

Looking at this result it is evident that there is no significant difference between the result for the serine/threonine kinase reactions and the result for all protein kinase reactions. Of the 245 kinetic laws, 140 could be directly identified as mass action and a significant percentage of the rest also seems to represent mass action kinetic types.

With this simple analysis, it was possible to very quickly get an overview over which kinetic laws are predominantly used for protein phosphorylation reactions and more specifically serine/threonine kinase reactions in a large set of models. As it is obvious that most models use mass action type kinetics for this type of reaction, this knowledge can be used to assign kinetic laws to the phosphorylation reactions of the NF- κ B model.

Chapter 6

Discussion

In this thesis, work on important standards in systems biology has been described and how it can be applied to systems biological research.

In the following sections, unsolved problems and future directions related to the topics in this work are discussed.

6.1 Layout And Render Information In SBML

Implementation(s) & Standardization

The SBML layout and render extension, described in chapters 3.4 and 3.6, was groundbreaking work because it was the first project that extended the SBML standard with new functionality and serves as an example to authors and implementers of other extensions to SBML today.

The extension fills a gap in the SBML format by providing the possibility to store arbitrary diagrams together with the mathematical descriptions of the reaction network within SBML documents.

The fact that implementers of other software tools implemented the SBML layout and/or render extension shows that the perception of the extension is favorable in the SBML community and that other scientists and software developers are convinced of its usefulness.

A lot of software has already been implemented with respect to the layout and render extension, but as with most other software projects, work is never truly finished.

Even so the implementation of the layout and render extension on top of libsbml are very stable and mature, the work in this field is not completed yet because it is somewhat of a moving target. With work on the SBML standard

as well as libsbml continuing, the specifications and implementations of the SBML layout and render extension have to follow this development.

There already are implementations for libsbml 3.4.1, libsbml 4.1.0, libsbml 4.2.0 and libsbml 4.3.0. Just recently, libsbml 4.3.1 has been released and the implementation of the layout and render extension has to be adjusted to the changes in this new release. In parallel to the development of libsbml 4, a new version of libsbml termed libsbml 5 is being developed and alpha versions of this new libsbml have already been released to be tested by the SBML community. This means that eventually the implementation of the SBML layout and render extension will also have to be adapted to that version of libsbml. Since this probably involves many major changes to the current implementation, this will not be a minor task.

Making larger changes to the implementation for libsbml also automatically means that the library that does the rendering has to be changed accordingly in order to work with the new version of libsbml and the implementation of the layout and render extension therein.

Another issue that has not been resolved yet is the fact that the standardization process for the SBML layout and render extension has not been fully completed. We have fulfilled several of the prerequisites, e.g. the requirement that there are at least two independent implementations, but some work still has to be done until the extension will reach the status of an official recommendation. Since there is currently no funding for this kind of work, this more or less has to be done in my spare time and therefore progress is slow.

This is also a large drawback of the current standardization process in SBML. The process is very lengthy and normally there is no funding for projects that take this long. This means that people usually have to move on to new projects before the old project is completely finished, i.e. has reach the state of an official recommendation.

Maybe, based on these experiences and on the fact that no extension proposal has yet managed to go through the complete standardization process, the SBML community should think about revising the standardization process to ensure that projects developing SBML extensions have a better chance of reaching the final stage.

Competing Efforts

Despite the generally favorable reception of the SBML layout and render extension by the SBML community, not all developers have added support for the extension in their tools yet.

Actually at least one research group has come up with a competing proposal that has roughly the same goals as the SBML layout and render extension. This other layout proposal has been brought forward by the developers of BioUML[76] because they have the impression that the layout and render extension does not provide them with the flexibility they need.

While we fully agree that our proposal does make certain compromises in terms of flexibility, these limitations were introduced on purpose in order to make implementations as simple as possible. Their proposal on the other hand requires the embedding of a complete programming language by supporting tools and therefore, while being very flexible, is also very hard to implement. This has already been discussed at several meetings of the SBML community and so far this competing proposal has not received any support due to its complexity which does confirm our approach.

Future Plans

We currently have a working, stable base implementation that allows layout and render information in the context of SBML files to be exchanged between different software tools. In addition to that we have an OpenGL based library that can display this layout and render information as well as the XSLT style sheet that allows the creation of high quality vector and bitmap drawings. But there certainly is room for improvement.

Besides the errors that have not been found yet, but that are certainly there, other improvements, e.g. enhancing the drawing quality of the renderer library with respect to fonts or making the drawing process more efficient on newer hardware, are obvious candidates for further work in this area.

One big problem related to the SBML layout and render extension is the actual creation of diagrams. Currently there are not many tools that allow users to create diagrams of the reaction networks, CellDesigner being one of the exceptions. Unfortunately the diagram information that is stored by CellDesigner can not be read by any other tool, rendering this feature useless when it comes to exchanging diagrams between different software tools. The conversion tool from the CellDesigner specific diagrams to the SBML layout and render extension as described in chapter 3.8, will eventually provide a solution to this problem. Work on this conversion tool is progressing nicely and I hope to be able to include this feature in one of the next versions of COPASI. In addition to that, I plan to release the source code of this conversion tool for use by other developers in their software.

Some developers are already working on the implementation of software

to automatically create layout information based on SBML models[186] or software that allows the user to create new style definitions in a graphical way, but there is still a lot of work in that area that needs to be done. We hope that the automatic layout algorithm that we are currently implementing in COPASI will contribute to this work.

6.2 Standards In COPASI

Extension And Improvement Of SBML Support

Implementation of the SBML standard in COPASI has provided our users with the ability to exchange reaction network models between many different tools. COPASI was one of the first tools to implement extensive support for this format and by providing feedback and help to the developers of SBML in these early years, we certainly played a major role in the development and improvement of this standard as well as its propagation.

From the start, COPASI has been a tool which provided very good support for SBML and this has been recognized by the systems biology community. The fact that COPASI has been downloaded more than 20000 times from our servers along with the fact that it is one of the major tools used to curate the models in the BioModels database can probably be seen as prove of this.

I am convinced that COPASI is one of the best and most reliable tools for working with SBML files. It is my goal to uphold the good reputation COPASI has gained and maybe even improve it.

Since the development of SBML has not stopped, neither has the development of support for new SBML features in COPASI. We are continuously adding new SBML features to the program and spend a great amount of time in making sure that this is done correctly.

COPASI currently does not support some of the features found in SBML and as time permits we hope to fill those gaps. Work on implementing some of these unsupported features has already been started.

We are currently also looking for ways to store information that is specific to COPASI, e.g. settings for tasks or output definitions, in SBML files in order to preserve this information when models are stored as SBML documents. Currently this information is lost upon export to SBML and has to be recreated after the model has been reimported into COPASI.

In addition to implementing support for SBML in COPASI, we collaborate with authors of different test suites to improve SBML support in

COPASI as well as improving the quality of the tests in these test suites. This is a very fruitful collaboration, for us as well as the authors of the individual test suites. Due to this good working relationship we are usually provided with pre-release versions of new tests in order to provide feedback to the developers before the tests are officially released.

Improvements For Layout And Render Information

The implementation of layout and render information in COPASI is also finished and in a very stable state. We ensure this with an extensive set of test cases.

Although COPASI has been able to display layout and render information for a few releases now, it was not possible to create new layout information. If the user wants to have layout information, it has to be created by external tools, e.g. the web based layout viewer written by Frank Bergmann[148].

The most important piece that is currently missing in COPASI with respect the layout and render information is a good diagram editor that enables the user to create reaction network diagrams in an intuitive way. Unfortunately creating such an editor is a major effort and currently there is no funding for this type of work.

Recent work on automatic layout creation in COPASI by Sven Sahle and myself might alleviate this problem to a certain extend. It will enable the user to at least create simple reaction network diagrams from within COPASIs user interface and use these diagrams to display results from COPASIs different analysis methods as described in chapter 4.6. This is very new work and due to the fact that the implementation is not completely finished and that the code needs more testing, this feature is not included in official releases of COPASI yet. Since only a few minor things are still missing, this feature will very likely be included in one of the next versions of COPASI.

So far there is only one force directed automatic layout method, but we hope to be able to add more different layout methods eventually. Especially layout methods that can handle the special restrictions of layouts of chemical reaction networks would be a welcome addition.

Another piece of work that might help in bridging the gap until a diagram editor can be implemented is the work I have been doing on converting the diagram information stored by CellDesigner to the SBML layout and render extension format. This will allow the users of COPASI to create the diagrams using the intuitive editing capabilities of the CellDesigner software and afterwards these diagrams are converted to a format usable by COPASI.

Initial results are very promising (see chapter 3.8) and many of the features usually found in reaction network diagrams from CellDesigner are already supported by this conversion tool.

Visualization Of Analysis Results

COPASI already uses the layout and render information to display results of some of the analysis and simulation methods, but there are still many more methods that could benefit from using the layout and render information for displaying results in a more intuitive way. Recent work on displaying results for the elementary mode analysis using the layout and render information has further illustrated the usefulness of being able to display analysis results graphically in the context of the complete reaction network (see chapter 4.6).

A new feature that has recently been implemented and that is not yet included in released versions of COPASI is the possibility to export all frames from the animation of a time course simulation (see chapter 4.3). Until now users are only able to view the animation in COPASI and to capture single frames of such an animation. Since these animations sometimes consist of hundreds or thousands of frames, exporting all of them is very tedious. This new feature allows users to create complete series of images that correspond to whole sections of the animation. These images can then be used to create movies using free third party tools like mencoder[187] or ffmpeg[188] or commercial tools like Apple Quicktime[189] or Adobe Premiere[190]. Since libraries for encoding movies like ffmpeg are freely available, it would eventually also be possible to use this library in COPASI to let the user create movies directly from COPASI without the need for external tools.

So far the implementation of the layout and render information has only been tested on small or medium sized reaction networks consisting of a few dozen reactions. Recent additions to the BioModels database also contain models that are significantly larger, e.g. whole genome models of *Saccharomyces cerevisiae*[191, 192]. Since the rendering code has not been optimized for speed yet, it is to be expected that the rendering speed for models of that size will be suboptimal, especially on hardware that is not state of the art. This is something that has to be tested and acted upon if necessary in order to enable COPASI to also display graphs of large reactions networks efficiently.

Language Bindings

Another area of work related to COPASI that is becoming increasingly popular and successful are the COPASI language bindings.

This project was initially only intended as a collaboration between the developers of the CellDesigner software and us to allow CellDesigner to use COPASI for the simulation of SBML models. This initial goal has been achieved and users of CellDesigner can now use some of the simulation and analysis methods provided by COPASI in the CellDesigner program.

We thought that maybe this might also be interesting to other developers and released the Java language bindings as well as the Python bindings with some documentation and examples on our web servers. As it seems this assessment was correct because the language bindings have by now been downloaded more than 1200 times.

According to feedback we have received via private communication or in our user forum, they are being used for many different projects, from small student projects to full scale research projects and the number of these projects seems to be increasing.

Although the graphical user interface of COPASI is very powerful and allows users to use a large range of different analysis methods, there are always limits to what can be integrated into a user interface without making it overly complicated. Due to these necessary restrictions in the COPASI software, the language bindings are also used frequently by students in our group to e.g. combine different analysis methods in ways not possible with the graphical user interface.

The language bindings as stated above have originally been created for a special purpose and we never thought that so many people would eventually end up using them. Currently the language bindings provide access to most of the functionality in COPASI in exactly the way they are implemented in COPASI. This means that working with the language bindings from languages like Java and Python is not always intuitive. Tuning the language bindings towards providing an interface that feels more natural to users of the different target languages would certainly make it easier for those programmers to use and get used to the language bindings.

Since the work needed for this fine tuning would be different for each target language, this would be a major task.

Another thing that is currently lacking is complete documentation for the COPASI API and how it can be used from the language bindings. We already provide more than 70 pages of documentation for each of the target

languages and we also include a set of examples for each of the different languages. These examples are intended to demonstrate how the language bindings can be used for the implementation of certain tasks, e.g. building models or running time course simulations.

Since the programming interface is very complex, more documentation would be very desirable and together with providing an API that is more tuned towards the target language, would be of great help to developers using the language bindings.

So far the functionality of COPASI can be accessed from C++, Java as well as Python. Currently we are working on extending the set of target languages by providing additional language bindings for Perl[83], R[87] as well as Octave[85].

The SWIG interface files have already been created and Dr. Jürgen Pahle has agreed to test the language bindings for these three new target languages.

As soon as the language bindings for these new target languages have received sufficient testing, they will be released alongside the other language bindings.

Right now we are providing binary files for about a dozen different language/operating system combinations and this number will grow once 64bit binaries for Microsoft Windows and Linux will be made available with the next release of the language bindings. All these packages are build and assembled manually, which is no longer possible once three more target languages have been added. This means that we will have to find a way to build, test and package the language bindings for the different operating systems and target languages automatically. Considering the number of different operating systems and target languages we are supporting, this will involve a significant amount of work.

6.3 Normalizing And Comparing Mathematical Expressions

The implementation on the normalization and comparison of arbitrary mathematical expressions also has been brought to a stage where it starts to be useful. It has for example been used to analyze all mathematical expression in the close to 300 curated models of the BioModels database (release from September of 2010) which only takes around two hours on a modern

processor.

As noted in the corresponding section, in order to complete this analysis, we had to remove one of the BioModels files, otherwise the analysis would stop with an error after running for a very long time. The model file causing this problem is model file number 217.

As a detailed analysis has shown, the problem can be attributed to a single very complex mathematical expression in this model file. During the normalization process products of sums are expanded to sums of products and for some expressions this can lead to an explosion of the resulting terms. This is the case for the afore mentioned expression in BioModels file 217. There the expansion of this single expression leads to such a large amount of terms that the computer eventually runs out of stack memory and the program stops with an error.

This problem is actually not a limit of the normalization algorithm, but it is a combined limit of the C++ programming language used for the implementation and the way we chose to implement the algorithm.

Since the expressions are internally represented as a tree, many of the methods working on the expression tree are implemented as recursions. That means that a method is called on a node and that node calls the same method on all its child nodes. This process continues for all branches of the expression tree until it encounters a node that has no children, a so called leaf node. Each call to the function however reserves a certain small amount of memory on the stack and the total amount of stack memory available to a C++ program is usually limited to a few million bytes. If the tree for an expression gets especially large, the program eventually runs out of space on the stack. This is exactly what happens for this one expression in BioModels file 217.

This means that the C++ programming language is not well suited for algorithms that lead to very deep recursions and this is one of the cases where this leads to an error in the running program.

This problem could potentially be solved in one of several ways. One solution would be to increase the stack size of the program. As a matter of fact, doing this will allow the normalization of the expressions BioModels file 217 to finish without error. Unfortunately, increasing the stack size of a C++ program is highly dependent on the operating system which means that this would have to be done differently for each platform. Another more feasible solution to this problem is to rewrite all the methods that use recursions on the expression tree data structure, which effectively means that large parts of the code have to be reimplemented for this algorithm to work for all possible

expressions.

Another problem with the current implementation is the rather low recognition rate when identifying e.g. kinetic laws. As shown in chapter 5.5, currently about 40% of the expressions analyzed can be identified when compared to existing functions from COPASIs function database.

One potential problem that could cause the low recognition rate lies in the kinetic parameters themselves. Since each kinetic parameter in SBML does not have to be a constant, but can be determined by a so called rule, it would be necessary to expand these rules similar to how we currently expand function calls.

A simple example that demonstrates this is depicted in figure 6.1. Looking at equation 1, one could get the impression that this describes a mass action term for an irreversible reaction with one substrate S . But since SBML allows parameters to be determined by arbitrary mathematical expressions, so called rules, one has to also consider if such a rule exists and what it looks like. This is complicated by the fact that all components in the assignment rule for k can themselves again be determined by assignments.

$$\frac{d[P]}{dt} = k * [S] \quad (6.1)$$

$$k = \frac{V_{max}}{K_m + [S]} \quad (6.2)$$

Figure 6.1: Equation one is the definition of a rate law for a reaction with one substrate S and one product P . Equation 2 determines the value of the kinetic "constant" k and is equivalent to an SBML rule.

A solution for this problem is easy to implement, but as further analysis has shown, this does not seem to be the main reason why the algorithm currently only identifies less than half of the kinetic laws.

To identify the real problem, the test program has been slightly modified to produce more information about the size of the expression clusters, especially those with many instances.

From the extended output in figure 6.2 it becomes clear that the kinetic functions that could not be recognized are divided into some very large clusters which contain predominantly mass action type kinetics or Michaelis-Menten type kinetics. This means that the normalization and the comparison

```

...
The expressions that could not be mapped are divided into 733 different expressions.
Expression with more than 20 instances (768): (-1) * G2R * k7r + G2K * R * k7
Expression with more than 20 instances (723): C * Y * a1
Expression with more than 20 instances (643): C * kd
Expression with more than 20 instances (171): (M * V2 * cell)/(K2 + M)
Expression with more than 20 instances (163): Bar1aex * Extracellular * alpha * k1
Expression with more than 20 instances (155): (-1) * PG2 * k25 + G2K * kwee
Expression with more than 20 instances (123): (CELL * Vsp * dClkF_tau1)/(K1 + dClkF_tau1)
Expression with more than 20 instances (68): (C * X * cell * vd)/(C + Kd)
Expression with more than 20 instances (58): (-1) * CC * Cell * k4 + Cell * P2 * T2 * k3
Expression with more than 20 instances (54): (V4 * X * cell)/(K4 + X)
Expression with more than 20 instances (53): (Vs * default * (KI)^(n))/((KI)^(n) + (Pn)^(n))
Expression with more than 20 instances (50): (M * V2)/(K2 + M)
Expression with more than 20 instances (41): (-1) * J12_k2 * Shc_dpEGFR * c1 + J12_k1 * L_dpEGFR * Shc * c1
Expression with more than 20 instances (37): (Mass * R * SPF * kp)/(Kmp + R)
Expression with more than 20 instances (37): (-1) * Cn * compartment_0000002 * k2 + CC * Cell * k1
Expression with more than 20 instances (30): ((-1) * Km * Vmax * c1 * ratio * s174 + Vmax * c1 * ratio
* s130 * s2 + Vmax * c1 * s130 * s2)/(Km)
Expression with more than 20 instances (29): ((-1) * Kms * Vr * compartment_2 * species_15 + Kmp * Vf
* compartment_2 * species_14)/(Kmp * Kms + Kmp * species_14 + Kms * species_15)
Expression with more than 20 instances (26): (CH2FH4 * NADP * Vm * cell)/(CH2FH4 * Km2 + CH2FH4 * NADP
+ Km1 * Km2 + Km1 * NADP)
Expression with more than 20 instances (26): (-1) * kr19 * x25 * x28 + k19 * x27
Expression with more than 20 instances (25): 0.0
Expression with more than 20 instances (23): Glucose * Lysine * compartment * k1a * p1
Expression with more than 20 instances (21): SS_Me * d_k_degr + SS_Me * mu
...

```

Figure 6.2: Excerpt from the extended output from the analysis of the same set of models as has been depicted in figure 5.19. In this output the actual number of expressions in clusters with more than 20 instances can be seen. There are several very large clusters (first 10) with several hundred expression which seem correspond to mass action kinetics and Michaelis-Menten type rate laws. Probably more than 3000 of the unrecognized 4800 expressions consists of mass action and Michaelis-Menten rate laws.

work as expected, but that the normalized expression is not recognized as e.g. a mass action expression.

The fact that they are not recognized lies in the way the expression recognition has been implemented. The kinetic functions in COPASI's kinetic function database expect individual kinetic functions to follow a certain schema. E.g. a mass action kinetic has to consist of the concentrations of all substrates (and all products) multiplied by a kinetic constant. In SBML, the user has more flexibility in specifying kinetic laws and their components which sometimes leads to kinetic expressions that e.g. represent mass action, but in a way that is not compatible with the mass action kinetic term as COPASI expects it.

One very common way these kinetics laws seem to be written in the models from the BioModels database is to use the amounts of the substrates (and products) instead of the concentrations. The conversion factor needed to convert these amounts to concentrations in these cases is implicitly included in the kinetic constants.

This applies equally to most other kinetic function types, so if the kinetic functions in a SBML model are not written in a way that is compatible with the way they are represented in COPASI, these functions will not be recognized by the current implementation.

One way to improve the recognition of kinetic functions would be to use a different standard to compare the normalized expressions against. One obvious candidate for this would be the entries for kinetic functions in the Systems Biology Ontology (SBO). The identifiers for these entries are for example used in SBML to annotate kinetic functions.

Looking at the corresponding entries in the SBO as for example depicted in figure 6.3 it becomes clear that this is not trivial. Each kinetic function in SBO is described by a text and by a number of alternative expressions. However, the expressions don't represent all possible ways of writing that kinetic law in SBML.

The SBO term described by entry SBO:0000054 shows that the kinetic law for an irreversible mass action reaction with two substrates should follow the scheme $k * R1 * R2$ where k is the kinetic constant and $R1$ and $R2$ are substrate quantities. In SBML the expression for this reaction could be written in many different ways, e.g. $k * R1/C1 * R2$ or $k * R1/C1 * R2/C2$ where k is the kinetic constant, $R1$ and $R2$ are the amounts or concentrations of substrates and $C1$ and $C2$ represent the volumes of compartments. How the expression has to be written in SBML depends on several things, e.g. whether the *hasOnlySubstanceUnits* flag has been set on one or more species participating in the reaction. The two examples given above represent only

a subset of the many different ways in which this kinetic law can be written in SBML. So even using all expressions for a kinetic law specified in the Systems Biology Ontology as a standard for the comparison would not solve the problem.

The only way to really determine if a given expression corresponds to a certain SBO term is to analyze the meaning of the textual description and to compare it to a given expression. This is nothing a computer can do automatically. For this, the textual descriptions of these SBO terms have to be converted to a set of rules in the form of program source code which can then be used to identify the expressions. This entails a lot of work and unfortunately we currently do not have the resources to pursue this.

http://www.ebi.ac.uk/sbo/main/browse.jsp?sbold=SBO:0000054

Term: SBO:0000054

Name
mass action rate law for second order irreversible reactions, two reactants, continuous scheme

Definition
 Reaction scheme where the products are created from the reactants and the change of a product quantity is proportional to the product of reactant activities. The reaction scheme does not include any reverse process that creates the reactants from the products. The change of a product quantity is proportional to the product of two reactant quantities. It is to be used in a reaction modelled using a continuous framework.

MathML

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
<semantics definitionURL="http://biomodels.net/SBO/#SBO:0000062">
  <lambda>
    <bvar><ci definitionURL="http://biomodels.net/SBO/#SBO:0000036">k</ci></bvar>
    <bvar><ci definitionURL="http://biomodels.net/SBO/#SBO:0000509">R1</ci></bvar>
    <bvar><ci definitionURL="http://biomodels.net/SBO/#SBO:0000509">R2</ci></bvar>
    <apply>
      <times/>
      <ci>k</ci>
    </apply>
  </lambda>
</semantics>
</math>
```

Rendered equation
 $\lambda(k, R_1, R_2) = k \times R_1 \times R_2$

Miscellaneous
 Date of creation:
 10 March 2006, 10:28
 Date of last modification:
 04 February 2010, 14:00

Parent(s)
[SBO:0000053](#) mass action rate law for second order irreversible reactions, two reactants (is a)
[SBO:0000163](#) mass action rate law for irreversible reactions, continuous scheme (is a)

Children
 This term has no child.

History [+]

Figure 6.3: Systems Biology Ontology term SBO:0000054, which describes the irreversible mass action term for a two substrate reaction for a reaction in a continuous simulation framework.

What the results of analyzing the model from the BioModels database also showed is that, although the complete analysis only takes a few hours, the analysis of certain expressions take several minutes to finish. Since users can

not be expected to wait 15 minutes for a file to be analyzed when they import it into COPASI, we need to improve the overall speed of the implementation before we can consider using this code in an official release of COPASI.

One obvious way to achieve this would be to distribute the normalization task over all processors available in the computer. Since modern desktop computers contain between 2 and several dozen processors, this should already provide a significant speed improvement.

In summary it can be said that the actual normalization and comparison works well, only the way we implemented the identification of the kinetic functions is too naïve and has to be improved. Due to the flexibility of the SBML format and the way the Systems Biology Ontology encodes the descriptions of kinetic functions, this is no trivial task and we don't expect to be able to come up with a solution in the immediate future. Since this problem could potentially be of interest to other developers handling SBML model files, this problem might be solved in a concerted effort distributing the work over several groups.

Acknowledgments

Inertia is the resistance of any physical object to a change in its state of motion or rest, or the tendency of an object to resist any change in its motion. It is proportional to an object's mass.

Wikipedia

OK. Now that most has been said or rather written, we come to the really important part, meaning all the people without whom this work would never have happened.

There are actually two contenders for first place in my list and in the end I decided to put them in chronological order.

So first of all, I would like to thank my parents and grandparents. Since the (two) people reading this probably posses knowledge about biology and biological processes, it should be clear that I would not be here without them.

I know it must have been awful to have a kid that thinks he knows everything better than his parents, but luckily you didn't think about giving me up for adoption. (At least I assume you didn't.)

Next in line for taking blame for this work is Prof. Dr. Ursula Kummer. Without her constant pushing and nagging about me finally writing this thesis, it would surely never have happened. And as a sign of how persistent she can be, it has to be noted that it took her close to ten years to finally push me this far. ☺

This shows certain parallels to characters from Greek mythology and can serve as proof that the physical principle of inertia is correct.

Without her constant support and input and the many valuable discussions, none of this work would probably have come to be. "Thank you so

much Ursula."

This also proves that no good deed goes unpunished, and she actually had to read this thesis.

(And to whoever else has to read this thesis: It is Ursulas' fault, so complaints should be directed at: `ursula.kummer at returntosendermail.de`.)

Now that the major blame has been distributed where due, there is a whole cloud of other people that were essential to this work in one way or other.

My gratitude also goes to Dr. Rebecca Wade for agreeing to referee this thesis. Since Rebecca is a very kind person and never did anything bad to me, there really isn't any good reason why she got punished with this task. I guess she was just in the wrong place at the wrong time. Sorry.

Another round of thanks goes to Prof. Dr. Ursula Klingmüller and Prof. Dr. Victor Sourjik who have kindly agreed to act as examiners at my defense. I really hope I am not going to waste your time. (That was actually a broad hint! ☺)

Further, I would like to thank my colleagues Dr. Stefan Hoops, Dr. Jürgen Pahle, Dr. Ursula Rost, Dr. Sven Sahle as well as Prof. Dr. Katja Wegner. They have played a major role in many of the projects described in this thesis. They also always provided me with lots of good advice when I ran out of ideas, which probably was quite often. Especially Dr. Sven Sahle contributed code and lots of useful feedback to many of the implementations described. (As a general rule one can say that the things that work have been contributed by him and I did all the rest.)

Also thanks to Jocelyn Faberman for proofreading parts of my thesis and making valuable suggestions. I know if I had let you have your way with this thesis, I might have been close to retirement by the time it would have been finished, but I would surely have gotten the Pulitzer price. Unfortunately I am a bit short on time, so we will never know. Likewise can be said for Dr. Ursula Rost. Whenever she finished reading some part of this thesis, a scary grin appeared out of thin air, reminiscent of the cheshire cat from Jim Carols "Alice in Wonderland", and I knew I had to brace myself.

Actually I would like to thank all my colleagues here in Heidelberg because without them, this place just wouldn't be the same. And the reason I am

enjoy working here is to a large part due to them.

This is really difficult because there are so many people that I owe thanks and I know that I will probably forget more than one.

A big thanks also goes to the SBML community as a whole and to Frank Bergmann who has provided me with lots of feedback regarding the SBML Layout and Render Extension. I think he is the only one who ever completely read the specification (including me).

Here, I would like to single out Prof. Dr. Akira Funahashi and Dr. Akiya Jouraku who provided me with valuable feedback regarding the Java language bindings.

I am also much obliged to Dr. Frederico Pinna for letting me borrow his model for the use case in this thesis. By patiently answering all of my stupid questions, he is also helping me in refreshing my biological knowledge again.

Some special thanks is due to Dr. Donald Knuth who by inventing the superb text typesetting system \TeX has saved me from writing this thesis with Microsoft Word. Thanks Dr. Knuth, you have probably saved the rest of what is left of my sanity.

Last but by no means least¹, big big thanks to my wife Anika who had to fill in for me² on an awful lot of occasions during the last few months and I could never have managed this without her. She was also a great help in writing this thesis and when she finally managed to read the complete introduction without falling asleep on the floor, I knew I was finally getting somewhere.

I also want to thank my three kids Felicia, Melina and Timon for making sure our local ice cream parlor did not go out of business while I was tied up, You did good, but now I am taking over again.

Since I have neither a cat nor a dog, the acknowledgments end here. And if I should ever get an Academy Award, I can hopefully reuse this text.

¹I had to say this because she is standing beside me with the rolling pin.

²Yes, those who know us both probably will have a hard time picturing this.

Bibliography

- [1] GOTTSCHALK, A., Prof. Carl Neuberg, *Nature* **178** (1956) 722.
- [2] Székessy-Hermann, V., Friedrich Wöhler synthesized urea 150 years ago, *Orv Hetil* **119** (1978) 3073.
- [3] Schmitz, R., [the beginnings of organic chemistry and today's neo-vitalism. on the 100th anniversary of Friedrich wöhler's death on 23 september 1982], *Sudhoffs Arch Z Wissenschaftsgesch Beih* (1984) 105.
- [4] Ramberg, P. J., The death of vitalism and the birth of organic chemistry: Wohler's urea synthesis and the disciplinary identity of organic chemistry, *Ambix* **47** (2000) 170.
- [5] Websters dictionary entry for "vitalism", <http://www.merriam-webster.com/dictionary/vitalism>.
- [6] Wikipedia entry for "vitalism", <http://en.wikipedia.org/wiki/Vitalism>.
- [7] Kinne-Saffran, E. and Kinne, R. K., Vitalism and synthesis of urea. From Friedrich Wöhler to Hans A. Krebs, *Am. J. Nephrol.* **19** (1999) 290.
- [8] Dahm, R., Discovering DNA: Friedrich Miescher and the early years of nucleic acid research, *Hum. Genet.* **122** (2008) 565.
- [9] Tipson, R., Phoebus Aaron Theodor Levene, 1869-1940, *Adv. Carbohydr. Chem.* **12** (1957) 1.
- [10] Gabryelska, M. M., Szymański, M., and Barciszewski, J., [dna: from miescher to venter and beyond], *Postepy Biochem.* **55** (2009) 342.
- [11] WATSON, J. D. and CRICK, F. H., Molecular structure of nucleic acids; a structure for deoxyribose nucleic acid, *Nature* **171** (1953) 737.
- [12] Crick, F., Central dogma of molecular biology, *Nature* **227** (1970) 561.

- [13] Khorana, H. G., Polynucleotide synthesis and the genetic code, *Fed. Proc.* **24** (1965) 1473.
- [14] Sanger, F. et al., Nucleotide sequence of bacteriophage phi x174 dna, *Nature* **265** (1977) 687.
- [15] Various, The human genome, *Nature* **409** (2001) 745.
- [16] Various, The human genome, *Science* **291** (2001) 1145.
- [17] Various, Double helix at 50, *Nature* **422** (2003) 787.
- [18] Venter, J. C. et al., The sequence of the human genome, *Science* **291** (2001) 1304.
- [19] Goodman, N., Biological data becomes computer literate: new advances in bioinformatics, *Curr. Opin. Biotechnol.* **13** (2002) 68.
- [20] Ng, P. C. and Kirkness, E. F., Whole genome sequencing, *Methods Mol. Biol.* **628** (2010) 215.
- [21] Fang, F. C. and Casadevall, A., Reductionistic and holistic science, *Infect Immun* **79** (2011) 1401.
- [22] Bhalla, U. S. and Iyengar, R., Emergent properties of networks of biological signaling pathways, *Science* **283** (1999) 381.
- [23] Aon, M. A., Cortassa, S., and Lloyd, D., Chaotic dynamics and fractal space in biochemistry: simplicity underlies complexity, *Cell. Biol. Int.* **24** (2000) 581.
- [24] Ross, J. and Arkin, A. P., Complex systems: from chemistry to systems biology, *Proc. Natl. Acad. Sci. U.S.A.* **106** (2009) 6433.
- [25] Kitano, H., Computational systems biology, *Nature* **420** (2002) 206.
- [26] Ideker, T., Galitski, T., and Hood, L., A new approach to decoding life: systems biology, *Annu. Rev. Genomics Hum. Genet.* **2** (2001) 343.
- [27] Cassman, M., Barriers to progress in systems biology, *Nature* **438** (2005) 1079.
- [28] Bruggeman, F. and Westerhoff, H., The nature of systems biology, *Trends Microbiol.* **15** (2007) 45.

- [29] Mendes, P., Gepasi: a software package for modelling the dynamics, steady states and control of biochemical and other systems, *Comput. Appl. Biosci.* **9** (1993) 563.
- [30] Gepasi web page,
<http://www.gepasi.org>.
- [31] Kitano, H., Funahashi, A., Matsuoka, Y., and Oda, K., Using process diagrams for the graphical representation of biological networks, *Nat. Biotechnol.* **23** (2005) 961.
- [32] Lloyd C, Halstead M, N. P., CellML: its future, present and past, *Prog. Biophys. Mol. Biol.* **85** (2004) 433.
- [33] Garny, A. et al., CellML and associated tools and techniques, *Philos. Transact. A. Math. Phys. Eng. Sci.* **366** (2008) 3017.
- [34] CellML web page, <http://www.cellml.org/>.
- [35] CellML case studies, <http://www.cellml.org/community/case-studies>.
- [36] Hedley, W., Nelson, M., Nielsen, P., Bullivant, D., and Hunter, P., XML Languages for Describing Biological Models, *Proceedings of the Physiological Society of New Zealand* **19** (2000).
- [37] Bullivant, D., Hedley, W., Hunter, P., Nelson, M., and Nielsen, P., Languages for the definition and exchange of biological models, *Proceedings of the Physiological Society of New Zealand* **20** (2001).
- [38] Demir, E. et al., The BioPAX community standard for pathway data sharing, *Nat. Biotechnol.* **28** (2010) 935.
- [39] BioPAX web page, <http://www.biopax.org/>.
- [40] Hucka, M. et al., The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models, *Bioinformatics* **1** (2003) 524.
- [41] SBML homepage, <http://www.sbml.org/>, 2006.
- [42] Le Novère, N. et al., The Systems Biology Graphical Notation, *Nature Biotechnology* **27** (2009) 735.
- [43] Electrical circuit diagram package for tikz,
<http://www.texample.net/tikz/examples/circuit-decorations/>.

- [44] Degtyarenko, K. et al., ChEBI: a database and ontology for chemical entities of biological interest, *Nucleic Acids Res.* **36** (2008) 344.
- [45] Degtyarenko, K., Hastings, J., de Matos, P., and Ennis, M., ChEBI: an open bioinformatics and cheminformatics resource, *Curr Protoc Bioinformatics* **14.9** (2009) 344.
- [46] Virtual Liver Network web page, <http://www.virtual-liver.de/>.
- [47] Kountouras, J., Boura, P., and Lygidakis, N., Liver regeneration after hepatectomy, *Hepatogastroenterology* **48** (2001) 556.
- [48] Fausto, N., Campbell, J., and Riehle, K., Liver regeneration, *Hepatology* **43** (2006) 45.
- [49] Michalopoulos, G., Liver regeneration, *J. Cell Physiol.* **213** (2007) 286.
- [50] Michalopoulos, G., Liver regeneration after partial hepatectomy: critical analysis of mechanistic dilemmas, *Am. J. Pathol.* **176** (2010) 2.
- [51] Pan, D., Hippo signaling in organ size control, *Genes Dev.* **21** (2007) 886.
- [52] Saucedo, L. and Edgar, B., Filling out the hippo pathway, *Nature Reviews Molecular Cell Biology* **8** (2007) 613.
- [53] Zhang, L., Yue, T., and Jiang, J., Hippo signaling pathway and organ size control, *Fly* **3** (2009) 68.
- [54] Halder, G. and Johnson, R., Hippo signaling: growth control and beyond, *Development* **138** (2011) 9.
- [55] Single unix specification,
http://www.unix.org/what_is_unix/single_unix_specification.html.
- [56] Oracle solaris,
<http://www.oracle.com/solaris/index.html>.
- [57] HP-UX,
<http://en.wikipedia.org/wiki/HP-UX>.
- [58] IBM AIX,
<http://www-03.ibm.com/systems/power/software/aix/index.html>.
- [59] BSD,
<http://www.bsd.org>.

- [60] GNU/Linux web page,
<http://www.gnu.org/>.
- [61] Wikipedia entry for the the x window system,
http://en.wikipedia.org/wiki/X_Window_System.
- [62] Debian linux home page,
<http://www.debian.org>.
- [63] Ubuntu linux home page,
<http://www.ubuntu.com/>.
- [64] Mac OS X,
<http://www.apple.com/macosx/what-is-macosx/>.
- [65] FreeBSD home page,
<http://www.freebsd.org/>.
- [66] NetBSD home page,
<http://www.netbsd.org/>.
- [67] Microsoft Windows,
<http://www.microsoft.com/windows/>.
- [68] Stroustrup, B., *The C++ Programming Language*, Addison Wesley, 2004.
- [69] Hoops, S. et al., COPASI—a COMplex PATHway SIMulator, *Bioinformatics* **22** (2006) 3067.
- [70] Bornstein, B., Keating, S., Jouraku, A., and Hucka, M., LibSBML: An API library for SBML, *Bioinformatics* **26** (2008) 880.
- [71] GNU Compiler Collection,
<http://gcc.gnu.org/>.
- [72] Microsoft visual studio express edition web page,
<http://www.microsoft.com/express/>.
- [73] Intel compiler web page,
<http://software.intel.com/en-us/articles/intel-compilers/>.
- [74] Oracle solaris studio web page,
<http://www.oracle.com/technetwork/server-storage/solarisstudio/overview/index.html>.

- [75] Java programming language wikipedia entry,
[http://en.wikipedia.org/wiki/Java_\(programming_language\)](http://en.wikipedia.org/wiki/Java_(programming_language)).
- [76] BioUML web page,
<http://www.biouml.org/>, 2011.
- [77] JSBML project web page,
<http://sourceforge.net/projects/jsbml/files/jsbml/0.8-b1/>.
- [78] Python history & license,
<http://docs.python.org/license.html>.
- [79] Cock, P. et al., Biopython: freely available python tools for computational molecular biology and bioinformatics, *Bioinformatics* **25** (2009) 1422.
- [80] Olivier, B., Rohwer, J., and Hofmeyr, J., Modelling cellular processes with python and scipy, *Mol. Biol. Rep.* **29** (2002) 249.
- [81] Krause, F. et al., Annotation and merging of SBML models with semanticSBML, *Bioinformatics* **26** (2010) 421.
- [82] Schulz, M., Bakker, B., and Klipp, E., Tide: a software for the systematic scanning of drug targets in kinetic network models, *BMC Bioinformatics* **10** (2009) 344.
- [83] Perl web page,
<http://www.perl.org/>.
- [84] Stajich, J. et al., The bioperl toolkit: Perl modules for the life sciences, *Genome Res.* **12** (2002) 1611.
- [85] GNU Octave web page,
<http://www.gnu.org/software/octave/>.
- [86] Matlab web page,
<http://www.mathworks.com/products/matlab/>.
- [87] R language web page,
<http://www.r-project.org/>.
- [88] Wikipedia entry for s programming language,
[http://en.wikipedia.org/wiki/S_\(programming_language\)](http://en.wikipedia.org/wiki/S_(programming_language)).
- [89] Beck, K., *Test Driven Development: By Example*, Addison-Wesley Professional, 2002.

- [90] Check unit testing framework web page,
<http://check.sourceforge.net/>.
- [91] GNU Lesser General Public License web page,
<http://www.gnu.org/licenses/lgpl.html>.
- [92] CPPUNIT web page,
http://sourceforge.net/apps/mediawiki/cppunit/index.php?title=Main_Page.
- [93] JUnit web page,
<http://junit.sourceforge.net/>.
- [94] Eclipse public license web page,
<http://www.eclipse.org/legal/epl-v10.html>.
- [95] GNU debugger web page,
<http://www.gnu.org/software/gdb/>.
- [96] Documentation to the DBX debugger,
<http://developers.sun.com/sunstudio/overview/topics/debugging.jsp>.
- [97] Wikipedia entry for the Intel debugger,
http://en.wikipedia.org/wiki/Intel_Debugger.
- [98] Microsoft debugger for windows web page,
<http://msdn.microsoft.com/en-us/windows/hardware/gg463009.aspx>.
- [99] Valgrind web page,
<http://valgrind.org/>.
- [100] Qt framework web page,
<http://qt.nokia.com/>, 2011.
- [101] Qt widget for technical applications,
<http://qwt.sourceforge.net/>.
- [102] Qwtplot3d web page,
<http://qwtplot3d.sourceforge.net/>.
- [103] OpenGL web page,
<http://www.opengl.org/>.
- [104] XML specification web page,
<http://www.w3.org/XML/>.

- [105] Expat XML parser web page,
<http://expat.sourceforge.net/>.
- [106] Xerces XML parser web page,
<http://xerces.apache.org/xerces-c/>.
- [107] Gnome XML parser web page,
<http://xmlsoft.org/>.
- [108] XML schema specification web page,
<http://www.w3.org/XML/Schema.html>.
- [109] RELAX NG web page,
<http://www.relaxng.org/>.
- [110] Resource description framework web page,
<http://www.w3.org/RDF/>.
- [111] Kanehisa, M. and Goto, S., KEGG: kyoto encyclopedia of genes and genomes, *Nucleic Acids Res.* **28** (2000) 27.
- [112] Croft, D. et al., Reactome: a database of reactions, pathways and biological processes, *Nucleic Acids Res.* **39** (2011) D691.
- [113] SBML Level 2 Version 4 specification,
<http://precedings.nature.com/documents/2715/version/1>.
- [114] Raptor RDF Syntax Library web page,
<http://librdf.org/raptor/>.
- [115] Jing RELAX NG validator web page,
<http://www.thaiopensource.com/relaxng/jing.html>.
- [116] Hucka, M. and Finney, A., Systems Biology Markup Language: Level 2 and beyond, *Biochemical Society Transactions* **31** (2003) 1472.
- [117] SBML WIKI,
http://sbml.org/SBML_Software_Guide, 2010.
- [118] Firefox browser web page,
<http://www.mozilla.com/en-US/firefox/>, 2011.
- [119] Safari browser web page,
<http://www.apple.com/safari/>, 2011.

- [120] Opera browser web page,
<http://www.opera.com/>, 2011.
- [121] Batik SVG library web page,
<http://xmlgraphics.apache.org/batik/>, 2011.
- [122] rsvg library web page,
<http://librsvg.sourceforge.net/>, 2011.
- [123] Inkscape SVG drawing program web page,
<http://inkscape.org/>.
- [124] Gimp web page,
<http://www.gimp.org/>, 2011.
- [125] Sauro, H. et al., Next generation simulation tools: the systems biology workbench and biospice integration, *OMICS* **7** (2003) 355.
- [126] Simplified Wrapper and Interface Generator web page,
<http://www.swig.org/>.
- [127] SCons software build system web page,
<http://www.scons.org/>.
- [128] CMake web page,
<http://www.cmake.org/>.
- [129] SBML supporting software web page,
http://sbml.org/SBML_Software_Guide, 2011.
- [130] libsbgn web page,
<http://libsbgn.sourceforge.net>, 2011.
- [131] Voet, D. and Voet, J., *Biochemistry*, Wiley, 3rd edition, 2004.
- [132] GraphML web page,
<http://graphml.graphdrawing.org/>, 2011.
- [133] SVG specification web page,
<http://www.w3.org/Graphics/SVG/>, 2011.
- [134] 8th SBML Forum Meeting,
http://sbml.org/Events/Workshops/The_8th_SBML_Forum_Meeting.
- [135] check unit testing framework web page,
<http://check.sourceforge.net>, 2011.

- [136] XSLT specification web page,
<http://www.w3.org/TR/xslt>, 2011.
- [137] Cairo rendering library web page,
<http://www.cairographics.org>, 2011.
- [138] SBML Level 3 extension proposals on sbml.org,
http://sbml.org/Community/Wiki/SBML_Level_3_Proposals, 2011.
- [139] SBML spatial extension proposal on sbml.org,
http://sbml.org/Community/Wiki/SBML_Level_3_Proposals/...
↔ Spatial_Diffusion, 2011.
- [140] model composition proposal on sbml.org,
http://sbml.org/Community/Wiki/SBML_Level_3_Proposals/...
↔ Hierarchical_Model_Composition, 2011.
- [141] SBML Level 3 development process,
http://sbml.org/Documents/SBML_Development_Process/...
↔ SBML_Development_Process_for_SBML_Level_3, 2011.
- [142] SBML Layout proposal on sbml.org,
http://sbml.org/Community/Wiki/SBML_Level_3_Proposals/Layout,
2011.
- [143] Glucose SVG image on Wikipedia,
http://commons.wikimedia.org/wiki/File:Glucose_structure.svg,
2011.
- [144] SBML Render Extension documentation,
<http://otto.bioquant.uni-heidelberg.de/bcb/sbml/>, 2007.
- [145] SBML Level 3 packages,
http://sbml.org/Community/Wiki/SBML_Level_3_Core/Package_mechanism,
2011.
- [146] Presentation of layout support in CellDesigner,
<http://sbml.org/images/2/26/Osana-celldesigner-layout.pdf>, 2005.
- [147] CellDesigner 4, towards CellDesigner 5,
<http://www.ebi.ac.uk/biomodels/meetings/2ndTrainingCamp/CellDesigner.pdf>.
- [148] SBW layout viewer,
<http://www.sys-bio.org/Layout/>, 2006.

- [149] Deckard, A., Bergmann, F., and Sauro, H., Supporting the SBML layout extension, *Bioinformatics* **22** (2006) 2966.
- [150] SBMLLayout Library web page,
<http://sbmllayout.sourceforge.net/>, 2011.
- [151] Shen, S., Bergmann, F., and Sauro, H., SBML2TikZ: supporting the SBML render extension in LaTeX, *Bioinformatics* **26** (2010) 2794.
- [152] Latex web page,
<http://www.latex-project.org/>, 2011.
- [153] PGF/TIKZ web page,
<http://sourceforge.net/projects/pgf/>, 2011.
- [154] Villéger, A., Pettifer, S., and Kell, D., Arcadia: a visualization tool for metabolic pathways, *Bioinformatics* **26** (2010) 1470.
- [155] Dräger, A., Hassis, N., Supper, J., Schröder, A., and A, Z., SBMLsqueezer: a CellDesigner plug-in to generate kinetic rate equations for biochemical networks, *BMC Syst. Biol.* **2** (2008) 39.
- [156] COPASI web page,
<http://www.copasi.org>.
- [157] C++ ISO standard,
<http://www.open-std.org/jtc1/sc22/wg21/docs/standards#14882>.
- [158] Berkeley Madonna,
<http://www.berkeleymadonna.com/>.
- [159] XPP-Auth,
<http://www.math.pitt.edu/~bard/xpp/xpp.html>.
- [160] Li, C. et al., BioModels Database: An enhanced, curated and annotated resource for published quantitative kinetic models, *BMC Syst. Biol.* **4** (2010) 92.
- [161] SED web page,
<http://www.gnu.org/software/sed/>.
- [162] Shapiro, B., Hucka, M., Finney, A., and Doyle, J., MathSBML: a package for manipulating SBML-based biological models, *Bioinformatics* **20** (2004) 2829.

- [163] SBML test suite,
http://sbml.org/Software/SBML_Test_Suite.
- [164] SBML test suite distribution,
<http://sourceforge.net/projects/sbml/files/test-suite/>.
- [165] SBML online test suite,
http://sbml.org/Facilities/Online_SBML_Test_Suite.
- [166] Bergmann, F. and Sauro, H., Comparing simulation results of SBML capable simulators, *Bioinformatics* **24** (2008) 1963.
- [167] SBML simulator comparison web page,
<http://www.sys-bio.org/sbwWiki/compare>.
- [168] Le Novère, N. et al., BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems, *Nucleic Acids Research* **34** (2006) D689.
- [169] Evans, T., Gillespie, C., and Wilkinson, D., The SBML discrete stochastic models test suite, *Bioinformatics* **24** (2008) 285.
- [170] Le Novère, N. et al., Minimum information requested in the annotation of biochemical models (MIRIAM), *Nat. Biotechnol.* **23** (2005) 1509.
- [171] MIRIAM web page,
<http://biomodels.net/miriam/>.
- [172] Rost, U. and Kummer, U., Visualisation of biochemical network simulations with SimWiz, *Syst. Biol. (Stevenage)* **1** (2004) 184.
- [173] Heinrich, R. and Schuster, S., The modelling of metabolic systems. structure, control and optimality, *Biosystems* **47** (1998) 61.
- [174] Machné, R. et al., The SBML ODE Solver Library: a native API for symbolic and fast numerical analysis of reaction networks, *Bioinformatics* **22** (2006) 1406.
- [175] Systems Biology Software Infrastructure,
<http://www.sbsi.ed.ac.uk/index.html>.
- [176] JlibSEDML web page,
<http://ntcnp.org/twiki/bin/view/VCell/JlibSEDML>.

- [177] Schilling, C., Schuster, S., Palsson, B., and Heinrich, R., Metabolic pathway analysis: basic concepts and scientific applications in the post-genomic era, *Biotechnology Progress* **15** (1999) 296.
- [178] SBML Level 2 Version 1 specification,
<http://sbml.org/Documents/Specifications/...>
↔ [All_Releases_and_Versions_of_SBML_Level_2](http://sbml.org/Documents/Specifications/...).
- [179] SBML Level 2 Version 2 specification,
<http://sbml.org/Documents/Specifications/...>
↔ [All_Releases_and_Versions_of_SBML_Level_2](http://sbml.org/Documents/Specifications/...).
- [180] SBML Level 2 Version 3 specification,
<http://precedings.nature.com/documents/58/version/2>.
- [181] SBML Level 3 Version 1 specification,
<http://precedings.nature.com/documents/4959/version/1>.
- [182] definition of the canonical disjunctive normal form for logical expressions from Wolfram Research,
<http://mathworld.wolfram.com/DisjunctiveNormalForm.html>, 2011.
- [183] definition of the canonical disjunctive normal form for logical expressions from wikipedia,
http://en.wikipedia.org/wiki/Disjunctive_normal_form, 2011.
- [184] Ashburner, M. et al., Gene ontology: tool for the unification of biology. The Gene Ontology Consortium, *Nat. Genet.* **25** (2000) 25.
- [185] gene ontology web page, <http://amigo.geneontology.org>.
- [186] Automatic layout proposal for google summer of code 2011,
<http://rumo.biologie.hu-berlin.de/gsoc/>.
- [187] mplayer/mencoder video player/encode web page,
<http://www.mplayerhq.hu>.
- [188] ffmpeg video/audio encoding library web page,
<http://www.ffmpeg.org>.
- [189] Apple Quicktime software web page,
<http://www.apple.com/quicktime/>.
- [190] Adobe Premiere software web page,
<http://www.adobe.com/products/premiere/>.

- [191] Herrgård, M. J. et al., A consensus yeast metabolic network reconstruction obtained from a community approach to systems biology, *Nat. Biotechnol.* **26** (2008) 1155.
- [192] Yeast whole genome model from biomodels database,
<http://www.ebi.ac.uk/biomodels-main/MODEL0072364382>.