HUMBOLDT-UNIVERSITÄT ZU BERLIN

MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT I
INSTITUT FÜR BIOLOGIE

Masterarbeit

ZUM ERWERB DES AKADEMISCHEN GRADES
MASTER OF SCIENCE

"Implementation of a simulation environment for the successive integration of mathematical models for cellular processes exemplified for central carbon metabolism in yeast"

"Implementierung einer Simulationsumgebung für die sukzessive Integration von mathematischen Modellen verschiedener Zellprozesse am Beispiel des Zentralstoffwechsels von Hefe"

vorgelegt von
Jens Hahn
geboren am 20.06.1985

betreut von
Edda Klipp
edda.klipp@rz.hu-berlin.de

angefertigt in der Arbeitsgruppe Theoretische Biophysik - tbp

am Institut für Biologie

Berlin, im Dezember 2013

Contents

1. Zusammenfassung **1**

2. Abstract **2**

3. Introduction **3**
3.1. A whole cell modelling approach for *S. cerevisiae* 3
3.2. The whole cell model as a modular entity 5

4. Material & Methods **7**
4.1. The simulation framework 7
4.2. External modules & software tools 8
4.3. ODE solver in the simulation framework 11

5. Results **13**
5.1. The framework for a whole cell modelling approach 13
5.2. A simulation step in the framework 16
5.3. Central carbon metabolism in yeast 29

6. Discussion **39**
6.1. Implementation of a simulation framework 40
6.2. Different approaches to implement a metabolic model 42
6.3. Outlook for the whole cell modelling approach 44

7. Acknowledgements **49**

References **50**

List of Figures **56**

List of Tables **56**

A. Appendix **57**

Eigenständigkeitserklärung **68**

1. Zusammenfassung

Die Bäckerhefe *Saccharomyces cerevisiae* steht seit Jahrzehnten im Zentrum der molekularbiologischen Forschung und macht diesen Organismus zu einem der wohlbekanntesten Eukaryoten unserer Zeit. Dank leichter Kulturbedingungen und der einfachen Verfügbarkeit bietet kaum ein anderer Organismus heute solch eine Bandbreite an experimentellen Daten und ist daher für die mathematische Modellierung besonders interessant. Die große Vielfalt an Informationen und Modellen verschiedenster Zellprozesse, machen diesen Organismus zu einem ausgezeichneten Ausgangspunkt für die Entwicklung des ersten eukaryotischen Ganzzellmodells.

Die Arbeit an einem solchen Modell kann durch eine spezifische Softwareumgebung, angepasset an gängige Standardisierungsformate der Systembiologie und konzipiert für die Simulation von großskaligen Modellen, maßgeblich erleichtert werden.

Die vorliegende Arbeit dokumentiert die ersten Schritte der Implementierung einer solchen Simulationsumgebung für mathematische Modelle und beschäftigt sich mit der Charakterisierung der Möglichkeiten und Limitierungen dieser, insbesondere in Hinsicht auf die Erstellung eines großskaliegen Modells der Hefe *S. cerevisiae*. Die Erstellung eines Modells des Zentralstoffwechsels der Hefe dient hierbei als exemplarische Anwendung.

2. Abstract

Today, the Baker's yeast *Saccharomyces cerevisiae* is one of the best known organisms. The close cellular relationship to other eukaryotic cells and the easy culturing in the lab, makes this organism to a widespread model organism for eukaryotic cells. Hence, a large spectrum of experimental data and mathematical models of various cell processes in yeast are available today. This makes this organism especially interesting for the implementation of large-scale or even whole cell modelling approaches.

A specific simulation environment, adjusted to established standards in the systems biology community and developed to simulate modular large-scale models, can facilitate the work on such a large-scale modelling approach.

In this work, a first version of such a simulation environment for large-scale modelling is presented. The benefits as well as possible limitations and challenges of the underlying concept are discussed. The implementation of a mathematical model describing the central carbon metabolism of the Baker's yeast *S. cerevisiae* demonstrates the application of the simulation framework and its functionality.

3. Introduction

3.1. A whole cell modelling approach for *S. cerevisiae*

Genome sequencing and high-throughput measurements have enabled biological research on a system level [1]. These synergetic approaches try to connect the different parts of a cell, which were investigated only isolated before. *In vitro* measurements of single enzymes to build dynamic models made way for *in vivo* time course measurements of metabolite concentrations on a whole cell level [2]. More and more comprehensive experiments were performed in the last years, investigating the behaviour of the whole system to changing conditions of the environment, e.g. change of carbon sources in the medium [3]. Also the interaction between different cell processes were investigated more closely, an example is given by Wittmann *et al.* [4], showing metabolite concentration changes during the cell cycle in *S. cerevisiae*. Today, many experiments with yeast are performed with synchronised cells to avoid average measurements of cells in different cell cycle stages [5] and to give a more precise picture of the the actual state of the individual cell system.

The given examples are just a few, showing the rise of a more holistic thinking in biological research in the last decades. The availability of these large-scale data sets enable the implementation of computational models with a wider range of scope than just small pathways or single processes. These models allow for the investigation of experimentally unfeasible scenarios and can be used as a reference for the interpretation of experimental results [6]. The implementation process of a whole cell model can also reveal missing links in the understanding of the system and therefore lead to new experimental settings.

Since the complete genome sequencing of *S. cerevisiae* in 1996, yeast is strongly connected to the success of synergetic approaches in systems biology and functional genomics [7]. Yeast is easy to culture and obtain in quantity, it has furthermore a stable haploid and diploid form, allowing for various genetic applications. For about 31%[1] of the protein-encoding genes in yeast (ORF) a robust mammalian homolog could be identified [8]. These aspects of yeast have given rise to a vast amount of experimental data, detailed gene and protein databases, and based on these to a large number of mathematical and computational models. A eukaryotic whole cell modelling approach is most possibly only doable for the Baker's yeast *S. cerevisiae*, no other eukaryotic unicellular organism satisfies better the requirements of available data and detailed knowledge about its cell processes.

[1] p-value 10^{-10}

The work with this large amount of information and computational data was considerably facilitated by the introduction of standards and standardised formats for the exchange of experimental data and mathematical models [9]. Firstly, *Minimum Information Required In the Annotation of Models* (MIRIAM) [10] is to be mentioned, this standard has increased the reusability of computational models by definition of a set of rules and standardised annotations to unambiguously identify model components and ensure the correct use of the certain model. Furthermore, the introduction of *Uniform Resource Identifiers* (URI) allowed for automated database queries to obtain information about the certain model and its components. The second important innovation in systems biology was the introduction of XML-based (Extensible Markup Language) exchange formats for the computational models themselves. Today the *Systems Biology Markup Language* (SBML) [11] is, besides *CellML* [12], the established standard format for computational models in systems biology. A whole cell modelling approach can only be implemented exploiting the given standards to facilitate cooperation between different researchers and to use the possibilities of automated data handling and exchange.

Several large-scale and whole cell projects were established in the last years. The main challenge of a whole cell model is the connection of various different cell processes, mostly described, by different mathematical descriptions, depending on the best suitable description for the given process. Whereas metabolic reactions are mainly described by DAE (differential algebraic equations), ODE (ordinary differential equations), or even FBA (flux balance analysis), other processes are described by stochastic algorithms or Boolean networks, e.g. gene expression [6]. A whole cell model therefore necessitates a specialised software compatible with the certain needs of the model description. One example is the well known *E-CELL* project [13], launched in 1996, this whole cell and multi-cell simulation tool provides the possibility to generate models by using predefined objects and simulation schemes. A similar functionality is provided by the *virtual cell* [14], a web based tool to simulate PDEs (partial differential equations) generated automatically based on predefined submodules which can be linked together. The latest whole cell modelling approach was made by Karr *et al.* [15] who designed a whole cell model of *Mycoplasma genitalium* in MATLAB [16]. This model is especially interesting because it is the first attempt to describe the complete cell at ones and does thereby allow for various mathematical descriptions. A former whole cell approach for this small organism was performed within the E-CELL project, in this approach an essential minimum set of 127 genes of *M. genitalium* were defined. This small whole cell model, the virtual *self-surviving cell* (SSC) [17], consisted of only 495 reaction rules and was able to demonstrate a simplified picture of molecular processes in this prokaryotic cell.

The demands on the simulation environment for our whole cell modelling approach are much higher, yeast has over 6000 genes and various compartments needed to be considered. Therefore, the simulation framework need to be adjusted to a large number of different cell processes. Naturally, a simulation framework for a whole cell model of this size need to be adjusted to the specific needs of large scale computational modelling in systems biology. To guarantee compatibility and the support of the established standards in the systems biology community, the software has to be closely connected to these standards and the related tools.

3.2. The whole cell model as a modular entity

The implementation of a whole cell model is always related to the partitioning of the cell into functional units, e.g. metabolism, gene regulation, and others. This functional separation is not only based on the mentioned utilisation of different mathematical descriptions, nonetheless, this makes the separation in most cases inevitable. Advantages of this separation are also the facilitated understanding and presentation of the units, also the implementation itself benefits from this modularisation. For our modelling approach, these functional units are further partitioned into modules, small computational models that describe single pathways or processes in the certain unit. The different modules can be implemented by different groups of people independently which further facilitates the cooperation and allows for the distribution of effort and expertise [18]. The modules as well as the units themselves can also be adjusted or even exchanged completely without the need of changing the whole model. Furthermore, implemented modules can be reused in other model approaches and existing computational models can be used as building blocks, representing modules for a functional unit or even substitute completely for a certain unit in the whole cell model.

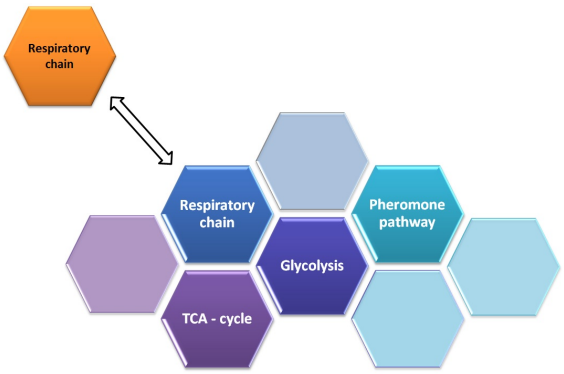


Figure 1: A modular approach

The implementation of the whole cell model using independent modules allows the comfortable exchange and addition of modules. The replacing does also not require a modification of the whole network.

We identified and tested different possible solutions for the implementation process of such a functional unit, exemplified on the central carbon metabolism of yeast. The implementation was also intended to identify shortcomings and limitations of the current version of the simulation environment. Also to characterise required extensions in regard to possible tools helping the implementation of new modules and identify problems in the model architecture, e.g. analysis of model/module properties.

A new large-scale simulation framework

A simulation environment for a whole cell model can be defined mainly by the supported mathematical descriptions of the modules and the method of combining them for simulation. Some tools like the already mentioned *E-CELL* project and the *virtual cell* [14] provide small building blocks to assemble models for simulation. These tools cannot easily be modified and extended in functionality, although some support the utilisation of *SBML* or *CellML* model files. Other tools like the *XS-system* [19] allow the user to define a chemical network, it creates automatically an ODE system with predefined kinetics and simulates this system. A systems biology tool with the closest relation to the simulation framework presented here was introduced by Hernández *et al.* [20], by the name of *M2SL*. This simulation library is written in *C++* and is designed to simulate multi-formalism and heterogeneous models together. Outside the systems biology community, the simulation of modularised multi-formalism systems is also a challenging problem, a general acrchitectural description is the *high level architecture* (HLA) introduced by Kuhl *et al.* [21]. *M2SL* as well as *HLA* provide a function for the interaction of decoupled simulations of particular components. Even though a simulation framework for mathematical models is not comparable to a general purpose architecture, the underlying principle is comparable.

Two basic concepts in regard of the handling of the modules in the simulation framework are apparent. While tools like *E-CELL* and *virtual cell* merge the modules to a large system and solve this newly build large model, other preserve the modularity, which can be provide several advantages, especially for large and complex systems. Firstly, the merging of multi-formalism systems is not easily doable and has to underlie severe limitations, in an independent simulation process, these formalisms can be simulated by specialised solvers and tools. Secondly, the simulation of small modules need less computational effort than the simulation of a large model. Additionally, the use of independent simulations facilitates the parallelisation of the simulation processes considerably, they easily be distributed to different computer cores.

The fundamental component of the simulation environment is the function to merge the modules or coordinate the module simulations, in either way. The quality of this coordination or consolidation function decides about the quality and accuracy of the whole simulation framework.

Despite the availability of simulation environments we decided to implement a new framework for the simulation of the whole cell model. Firstly, to ensure the adjustment to our requirements concerning the model itself and secondly, to guarantee full support of the tools commonly used in the group of theoretical biophysics.

4. Material & Methods

4.1. The simulation framework

The framework of the simulation environment for the whole cell model is written in the programming language *Python* [22]. This language is commonly used in the systems biology community, therefore, several tools for computational modelling are available and can be easily integrated. The plotting of simulation time courses or other numerical values is performed using the package *matplotlib* [23]. Another advantage of this programming language is the readability of *Python* code which improves the reusability for future user. The object-orientation enables the conversion of different model formats to Python module objects. Thereby, the module object can hold all information necessary for the simulation in *Python dictionaries*, a data type referred to as *associative array*. These arrays are indexed by *keys* or names, instead of only numbering the entries in order of the first occurrence, which facilitates the call of different values considerably. A *Python* module object holds the following information:

- Name of the module
- Initial values for all variables
- Parameter values
- Rates of the different reactions
- Equations for all variables
- Annotations for variables and compartments

The import of computational models supports 3 different exchange file formats:

- **SBML:**
  The *Systems Biology Markup Language* [11] is a standardised exchange format for mathematical models based on *XML*. It allows the storage of the chemical reaction network, the stoichiometric matrix, kinetic formulas and numerical values for the dynamic simulation of the model. Several tools in the systems biology community support this file format to simulate, analyse and adjust the computational model. One of these tools is the *libSBML* [24] library, which allows the easy import of data from SBML files into Python, a corresponding Python script was provided by Dr. Thomas Spießer.

- **SBtab:**
  The *SBtab* [25] file format is a unifying spreadsheet format to store different types of information. The use of predefined tables and columns allow for the automated conversion of the stored information into *Python* data types. The *SBtab* parser supports all common spreadsheet formats, e.g. *Excel* files [26]. Furthermore, the tables can automatically read out using web tools to translate them directly into *SBML* or vice versa. *SBtab* was developed by Wolfram Liebermeister *et al.* [25].

- **Python script:** To enable the direct import of the *Python* data structure used by the module objects, the information can be stored directly in a *Python* script. An example for the data structure is given in figure 21, showing the *Lotka-Volterra* predator-prey model [27].

For the unambiguous identification of module variables regardless of the short name used in the certain module, the modules need to be annotated. We decided to use identifiers from the database and ontology of *Chemical Entities of Biological Interest* (ChEBI) [28] for the identification of chemical components in the whole cell model. The component containing compartments are identified by annotations from the *Gene Ontology* project (GO) [29]. These annotations consist of the abbreviation of the database, *ChEBI* or *GO*, respectively, and a unique number corresponding to the certain database entry. These annotations allow also for a web query to obtain additional information about the particular item.

4.2. External modules & software tools

The modularity of the whole cell model allows the utilisation of existing computational models as building blocks. To demonstrate the implementation process of a large-scale model for the whole cell approach, we used different existing models of metabolic processes to create an exemplified large metabolic model for the yeast *S. cerevisiae*.

1. **Glycolysis - Hynne *et al.* 2001** [30]
   This model describes the anaerobe glycolysis in *S. cerevisiae*. The model uses only ordinary differential equations and is fully parametrised. The kinetics are based on the law of mass action but do also exploit more complicated rate laws, e.g. *Ordered Bi Bi* kinetics [31]. The kinetic parameters are given in millimolar and minutes, the simulated time courses show, the model levels out at an oscillating steady state. Additionally to the basic glycolytic reactions of forming pyruvate out of glucose, it consists of reactions for the uptake of extracellular glucose, the formation of glycerol, glycogen (storage), and ethanol, as well as their export over the plasma membrane. Besides, the experimental setting of the underlying continuous-flow experiment required the addition of *cyanide* (CN^-) to the model. The model comprises 22 chemical species in 24 reactions and uses 2 compartments, the extracellular medium and the cytosol. Figure 17 in the appendix of this work shows the reaction network of this model.

2. **Mitochondrial energy metabolism - Nazaret *et al.* 2009** [32]
   This dynamical model is a simplified model of a generalised mitochondria. The given ordinary differential equations represent mainly lumped reactions of the TCA cycle, the respiratory chain and the related membrane proteins, e.g. ATP synthase and adenine nucleotide translocator. The system reaches a stable steady state after a short time of levelling. The model is fully parametrised, the kinetic parameters are given in molar and seconds. The model comprises 10 chemical species and the membrane potential over the inner mitochondrial membrane as variables, it holds 14 reactions. The reaction network of this model is shown in the appendix of this work, Figure 18.

3. **Mitochondrial energy metabolism - Wu *et al.* 2007** [33]
   This ODE model of the mitochondrial energy metabolism is part of the *virtual physiological rat project*, it provides different parameter sets for different experimental settings or different rat tissues, respectively. The kinetic parameters are given in molar and seconds. The rate laws utilise various different kinetics, ranging from *Michalelis-Menten* [34] kinetics to *Ordered Bi Bi* kinetics. The model describes the TCA cycle, several membrane transporters and diffusion processes as well as the respiratory chain. The model holds 62 chemical species and the membrane potential as variables, these are distributed over 3 different compartments, the cytosol (external), the intermembrane space and the mitochondrial matrix. A reaction scheme of this model is shown in Figure 19 in the appendix of this work.

4. **Yeast metabolism - Stanford *et al.* 2013** [35]
   This large dynamic model of yeast metabolism is based on the yeast consensus metabolic network [36]. It comprises ordinary differential equations with kinetic parameters given in the units molar and seconds. The kinetic rates were created automatically using a parameter balancing tool introduced by Lubitz *et al.* in 2010 [37]. The tool created a set of thermodynamically consistent parameters and kinetic rates using *modular rate laws* [38] for the rate equations. The model consists of only 2 compartments, an extracellular and an intracellular part, holding 285 reactions and 295 species as variables. Several anabolic and catabolic processes are presented, besides the glycolysis and the TCA cycle this model also describes a basic biomass function including amino acid and fatty acid synthesis.

5. **Yeast consensus metabolic network - Version 7.0** [36]
   This model is not a dynamic mathematical model, instead a large metabolic network map of *S. cerevisiae* based on the genome sequence and literature information. It holds stoichiometric information of 3498 reactions and 2384 chemical species in 14 Compartments.

Parameter balancing [37]

The *parameter balancing* tool which was used for the parametrising of the metabolic model from Stanford *et al.* was also used for the implementation of a newly implemented metabolic model based on the *yeast consensus model* version 7.0. The tool automatically initialises the reactions using *modular rate laws* [38] and calculates a set of thermodynamic consistent parameters. For the calculation it uses an *SBtab* table, providing rate constants, formation energies and initial concentrations. The structural network information has to be provided in an SBML file. It uses thermodynamic dependencies of the rate constants, e.g. Wegscheider conditions, to estimate unknown parameters and adjusts the given parameters to a global thermodynamically consistent state. The calculation and adjustment of the parameter set employs Bayesian estimation [39]. The adjusted parameters can be downloaded in an *SBtab* file, the re-parametrised model is provided in an *SBML* file. The tool is part of *semanticSBML* [40].

COPASI - COmplex PAthway SImulator [41]

The creation of an *SBML* model file was facilitated by the software *COPASI* 4.11. This application provides a graphical user interface for creation, simulation and analysis of biochemical models. Furthermore, the import and export of *SBML* files is supported. This tool was used to create the new metabolic model. The use of *COPASI* provided the possibility to test the model on the fly while adding, modifying and removing reactions

and species. Simulations were performed with the integrated ODE solver *LSODA* [42], a solver with an automated switch between stiff (*backward differentiation formula*) [43] and non-stiff (*Adams-Moulton method*) solver methods [44], depending on the behaviour of the system.

4.3. ODE solver in the simulation framework

To test the accuracy and performance of different methods and solvers for the use in the simulation environment, several different ODE solver libraries were integrated. These packages differ not only in the employed solver methods, but also in the processing of the ODE data.

The *SBML ODE Solver Library* (SOSlib) [45] and *libSBMLsim* [46] are both frameworks utilising the *libSBML* library directly to get the data from the *SBML* file of the module. These two are written in *ANSI C* [47] and pass the *Abstract Syntax Trees* (AST), the mathematical representation of the data in *libSBML*, directly to the actual solver routine. This allows a very fast translation of the data into a format that can be used to integrate the ODEs. The *libSBMLsim* library provides additionally a *Python* binding, that facilitated the use in the whole cell modelling simulation environment. It also allows the integration of the equations with 15 different solving methods, among others *Adams-Moulton*, *backward differentiation formula* (BDF), and *Runge-Kutta* [43]. The *SBML solver library* employs the solver *CVODES* [48], a solver from the *SUNDIALS* solver package [49] with additional analysis functions, this solver provides the methods *Adams-Moulton* and *BDF*.

The solver framework *ODEint* [50] and the simulation package *Assimulo 2.4* [51] can use the *Python* module data structure of the whole cell modelling environment. *ODEint* is part of the *scientific Python* package *SciPy* [52] and represents the standard ODE solver for *Python*. The employed solver is *LSODA* [42] from the solver package *ODEPACK* [53], as mentioned in the end of the last chapter, a solver with an automated switch between a solver method for stiff or non-stiff problems, respectively. The last solver framework is *Assimulo*, a *Python* binding for the *SUNDIALS* solver package [49]. We implemented the solver *CVODE* [54], a state of the art solver providing *Adams-Moulton* and *BDF* methods for solving non-stiff or stiff problems, respectively.

Adams-Moulton method for non-stiff problems [44]

The *Adams-Moulton* method is an implicit linear multistep method for the numerical solution of ordinary differential equations. Instead of calculating the current approximated value y_n of a differential equation only by considering the last value y_{n-1} and its derivative (*Euler's method*), a linear multistep method uses a linear combination of the last calculated values and their derivatives to estimate the current value y_n. Compared with the *Runge-Kutta* method, which uses intermediate steps between the previous and the current value for calculation, the advantage of a multistep method is that an increased order of the method can be reached without increasing the number of calculations. Instead, the order is defined by the number of employed previous values for calculation (steps).

The *Adams-Moulton* method can be described by equation (1) with $a_0 = 1$, $a_1 = -1$, $a_2, \cdots, a_m = 0$. The right side of the equation is approximated by polynomial interpolation, requires the use of an iteration algorithm[54]. If $b_0 = 0$ equation (1) describes the *Adams-Bashforth* method, an explicit method, because the value y_n is only present on the left side of the equation. An implicit method employs additionally the derivative of the point to be calculated ($b_0 \neq 0$). Sometimes, an explicit and implicit method are connected to a *predictor-corrector method*, here the explicit method is used for a first approximation, the implicit method for the correction.

$$\sum_{j=0}^{m} a_j \, y_{n-j} = h \sum_{j=0}^{m} b_j \, f(x_{n-j}, y_{n-j}) \tag{1}$$

Backward differentiation formula (BDF) for stiff problems [43]

The backward differentiation formula is also an implicit linear multistep method, it can also be described by equation (1) with $b_1, \cdots, b_m = 0$. Here, the left side of the equation is approximated by polynomial interpolation, instead of the right side. This leads to a change of the stability properties of the *BDF* method and makes it more suitable for stiff problems.

A characteristic property of a stiff ODE system is the unnecessary decrease of the step size for some, mostly explicit, solver methods. This decrease is only based on stability, not on the accuracy of the method [44] and leads to a drop of the efficacy of the employed solver methods.

5. Results

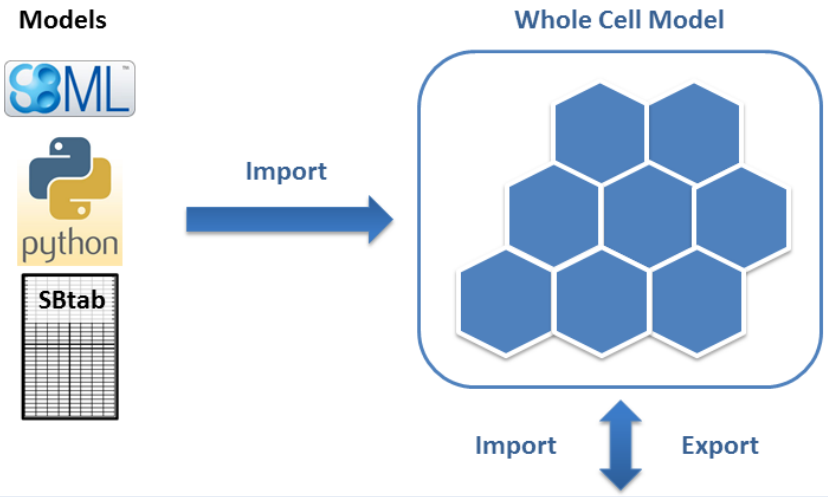
5.1. The framework for a whole cell modelling approach

The whole cell modelling approach is based on the idea of modularity, in regard of the use of small modules as building blocks for the model, as well as during the simulation process. The modules are not connected or merged directly, instead, each module performs an independent or decoupled simulation for a discrete small time interval. Afterwards a reconciliation function reconstructs a consistent cell state, by merging the simulation results. This makes the consolidation function to the crucial component of the simulation framework for the whole cell modelling approach.

Identification of cell components

The recovery of a consistent cell state requires the unambiguous identification of the components in each module, since the consolidation function has to merge the concentration changes resulting from the independent simulations of each module. To ensure this identifiability, we introduced annotations and identifier commonly used in the systems biology community provided by public databases. Figure 2 shows the communication and identification processes of the simulation environment, the modules can be imported using different file formats, the framework automatically creates *Python* module objects from these files and uses the identifier provided in the modules to identify the certain components. This method guarantees the successful match of components in different modules independent from the chosen identifier or short name in the module itself. Furthermore, it allows for the unambiguous identification of components in databases and prevents ambiguity errors due to indistinct naming of compounds in the modules.

A similar procedure is used to identify the localisation of the compounds. The consolidation function does not only need to identify a particular species, but also the localisation of the species in the cell. Each component in the simulation framework is therefore identified by a pair of annotations, one for the species itself and one for the containing compartment. For metabolic compounds the *Chemical Entities of Biological Interest* database (ChEBI) is used, the identification of compartments utilises *Gene Ontology* (GO) terms. If a certain compound has no unambiguous identifier from one of these databases, other standard terms can be used, e.g. *Kyoto Encyclopedia of Genes and Genomes* (KEGG) [55]. To ensure a correct matching for the simulation a visualisation of the coupled species was implemented by Dr. Martin Seeger. Figure 22 in the appendix of this work shows an example for three modules imported and coupled via matching of the components performed by the framework automatically.



!!SBtab	TableType='WCM-Species'	TableName='WCM'	
!WCM-Species	!Species	!Module	!Annotation
WCM_sp_1	atp	Hynne_Glycolysis	ChEBI:15422
WCM_sp_2	adp	Hynne_Glycolysis	ChEBI:16761
WCM_sp_3	ATP	Heiske_RespChain	ChEBI:15422
WCM_sp_4	adenosine triphosphate	Nazaret_TCA	ChEBI:15422

Figure 2: Import and connection of modules

The modules can be imported using a model written as *Python* scripts or use one of the exchange formats *SBML* or *SBtab*, respectively. The matching of the same compounds in different modules is performed automatically using unambiguous annotations for all components involved (e.g. *ChEBI* ids). The table at the bottom of the scheme shows the identifier for adenosine 5'-triphosphate as an example in an *SBtab* table.

The current cell state, all metabolite concentrations and values to describe the condition of the cell, are stored in the *state vector*. The *state vector* has the data structure of a *Python* dictionary, every compound is identified by the particular standard id and the standard annotation of the containing compartment:

E.g.: ATP in the cytosol $\Longrightarrow$ CHEBI:15422 GO:0005829

Models stored in Python module objects

To facilitate the work with several modules imported from different data sources, an object oriented module data structure was introduced by Dr. Martin Seeger. The use of module objects provides a fast and coherent processing of the modules. Instead of a centralised simulation control in the framework, the modules can perform the simulations independently. This facilitates not only the parallelisation of the simulation process, but also the introduction of test and control methods for debugging. The framework serves only as a control instance for the model, the initialisation process and simulations are performed by the module objects themselves, afterwards, the module objects report back to the framework which coordinates the next steps. The user can also work with the module objects directly to perform single module simulations or to observe the modules behaviour during the whole cell simulation. For this reason, each module has its own simulation and plotting methods available, a module state vector holds the actual state of the module object.

The initial cell state The first inconsistent state of the cell is the initial state, most probably, the different modules do not contain the same initial values for matching components. The initialisation of the modules is therefore connected with the declaration of a global initial state. The initial values can be set by the import of an *SBtab* table holding the values for the state vector. If the state vector is set for an initial state, the module state were updated. If no initial state is provided to the framework, the initial values of the state are set by the module values, depending on the import order. In either case, an initialisation step ensures the match of the initial values for the first simulations and therefore a consistent initial state.

5.2. A simulation step in the framework

A crucial parameter of the simulation framework is the discrete time step Δt, it defines the time interval in which the modules are simulated independently from each other. A simulation cycle consists of all modules simulated one after another. After each cycle, the consolidation function merges the results from the modules and updates the state vector. The merging process is based on numerical changes, the modules change the value of the particular component individually, the consolidation function sums up the changes of all modules involved and adds the result to the value in the state vector. The following simulation step is initialised with the new state vector. During the simulation of a module, this module is completely decoupled from the others. Therefore, the values of coupled species in other modules are held constant during the simulation time Δt. This leads to an inconsistent state after a simulation cycle, since the different modules hold divergent numerical values for identical components. After the consolidation step, the cell state is consistent again. Figure 3, shows the scheme of one simulation step t_n to t_{n+1} of the size Δt, exemplified for 3 modules X, Y, and Z. Due to the possible couplings between the modules, the accurate solution of a module is a function of the variables of all 3 modules, even if only a few of the variables in the modules are actually coupled. Instead of solving the function f(**x**, **y**, **z**, t), the independent simulation leads to the solving of a decoupled function, only dependent on the variables of the particular module, denoted as $\tilde{f}$. The changes of the variables due to the other modules are neglected, these module variables are held constant.

Error estimation for the simulation process

An important aspect of the simulation framework is the question of the error being made by using this modular approach, the identification of error sources and the possible reduction of these. To test the simulation framework and to demonstrate the possible error sources, we implemented 2 test systems.

1. **Lotka-Volterra:** predator-prey model [27]
   This well known model is a simple ODE system comprising 2 autonomous first order ordinary differential equations. The 2 strongly coupled species, describe the simplistic behaviour of a predator - prey relationship. The growth of the prey population feeds the predators and leads to an increasing amount of predators. The rise in the predator population lowers the number preys which then decreases also the number of predators due to starvation. The system consists of two terms for each species, a formation and a degradation term, while the formation term for the *prey* species, denoted as **u**, is only dependent on the current amount of

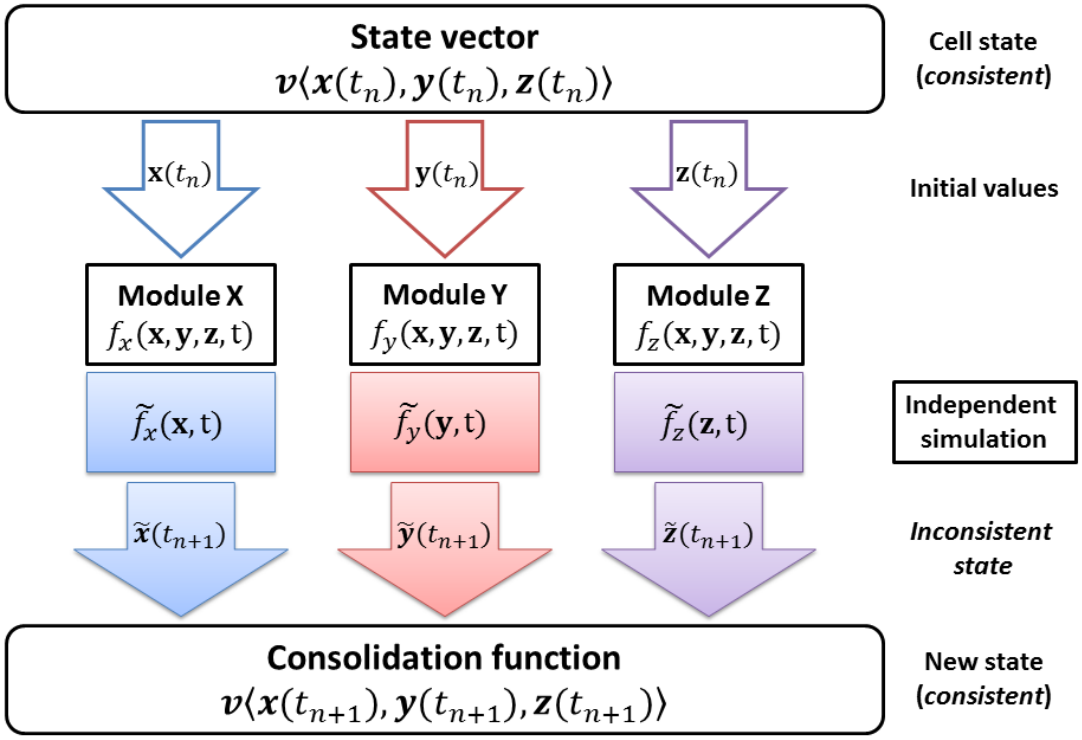


Figure 3: One simulation step

The figure shows a single simulation step, t_n to t_{n+1}, of the simulation framework. The state vector **v** holds the numerical values of the current cell state consisting of the initial values of the three modules X, Y and Z. The vectors **x**, **y** and **z** hold the species in the particular module. Since some species are present in several modules, the module holds a function of all three vectors. The independent simulation is a reduction to the dependency of only one vector, which leads to the inconsistent states $\tilde{x}$, $\tilde{y}$, and $\tilde{z}$, respectively. The consolidation function merges the results and writes the new consistent states to the state vector.

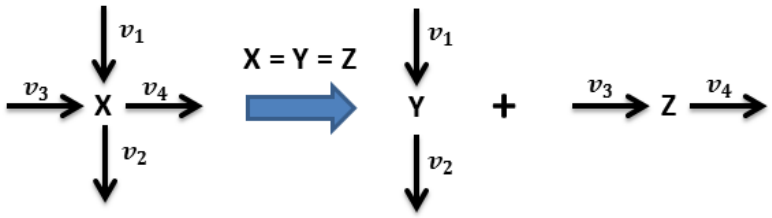


Figure 4: Example of the modularisation of a simple mass action system
The scheme shows the separation of the species x into the species y and z, contained in different modules Y and Z, respectively. Therefore, each module contains different reactions of the species x.

preys, the formation term of the *predator* species, denoted as x, is dependent on the amount of predators and preys. The degradation term is coupled vice versa, the amount of predators increases the degradation rate for the preys. A simulation time course of the system with the given parameters is shown in Figure 5a.

$$\text{Prey}: \quad \frac{\mathrm{d}u}{\mathrm{d}t} = \mathrm{k}_1 \cdot u - \mathrm{k}_2 \cdot u \cdot x$$
$$\text{Predator}: \quad \frac{\mathrm{d}x}{\mathrm{d}t} = \mathrm{k}_3 \cdot u \cdot x - \mathrm{k}_4 \cdot x$$

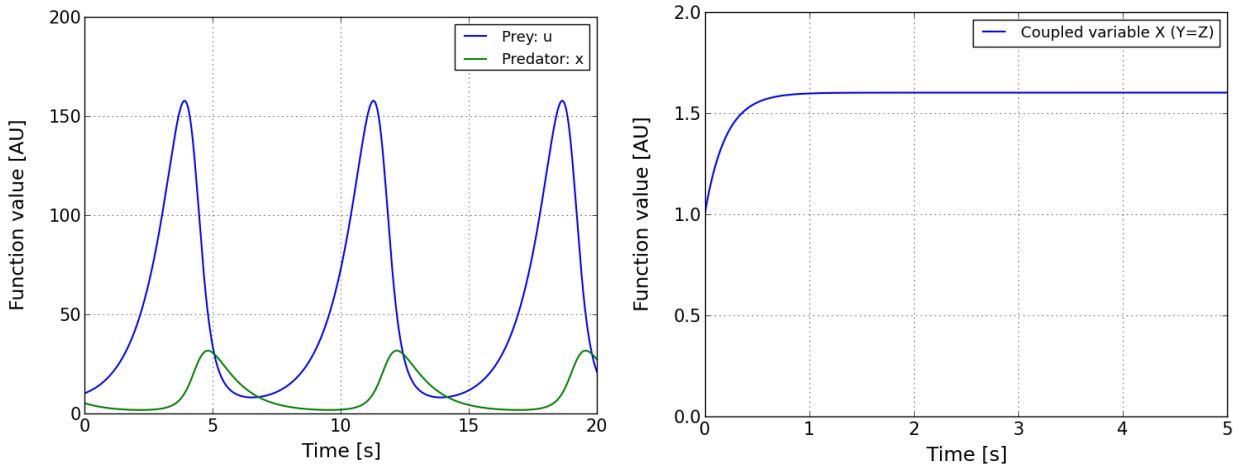
Used parameter values: $\mathrm{k}_1 : 1.0 \quad \mathrm{k}_2 : 0.1 \quad \mathrm{k}_3 : 0.02 \quad \mathrm{k}_4 : 1.0$

2. **Mass action kinetic**

   This example is a simplified system to demonstrate the functionality of the simulation framework. It consists of two species with a constant formation (v_1, v_3) reaction and a concentration dependent (mass action kinetic) degradation reaction (v_2, v_4). It can be used as an example for a species occurring in two modules in the whole cell modelling approach, Figure 4. The species x is occurring in both modules Y and Z, the modular approach leads to the decoupling and splitting of the species x into y and z. Figure 5b shows a simulation time course of the coupled system. The depicted variable shows bounded growth, the value increases with a high rate and reaches then a stable steady state.

$$\text{Module Y}: \quad \frac{\mathrm{d}y}{\mathrm{d}t} = \mathrm{v}_1 - \mathrm{v}_2 = \mathrm{k}_1 - \mathrm{k}_2 \cdot y$$
$$\text{Module Z}: \quad \frac{\mathrm{d}z}{\mathrm{d}t} = \mathrm{v}_3 - \mathrm{v}_4 = \mathrm{k}_3 - \mathrm{k}_4 \cdot z$$

Used parameter values: $\mathrm{k}_1 : 5.0 \quad \mathrm{k}_2 : 3.0 \quad \mathrm{k}_3 : 3.0 \quad \mathrm{k}_4 : 2.0$



(a) Time course: Lotka-Volterra model

(b) Time course: Mass action model

Figure 5: Time courses of the simulated test models

The figure shows the simulated time courses of the presented test systems, used as references for the testing. Both systems were integrated with Assimulo 2.4, Adams-Moulton method.

Limitations of the solver methods

The used differential equations in the models presented in this work are autonomous ordinary differential equations of first order. The existence and uniqueness of a solution can be assumed since all used functions satisfy the *Lipschitz condition* [44], the equations are globally differentiable and have a bounded derivative. A typical property of chemical reaction kinetics is the occurrence of *stiff* ODE systems [43]. This characteristic can be challenging for explicit methods, e.g. *Adams-Bashforth* method, but in some cases also for the *Adams-Moulton* method. These methods have to use a very small step size while *BDF* or other methods employable for the solving of stiff systems can enlarge the step size and show a much better performance. This decreasing of the step size is here not based on accuracy but on stability of the particular method. A stiff system can be identified by the occurrence of very fast and very slow decaying solution components in a system. A precise definition or test method does not exist. A simulation test can show very quickly whether a method is appropriate to simulate a certain system or not.

The decoupling of the ODE systems is not affecting the solver method, the solver is called with new initial values after each simulation step and solves the system for a short time interval Δt. Inherent solver functions as the step size control or a corrector-predictor are also not affected.

The implemented solver packages employs only the *Adams-Moulton* method or *BDF*, the *SBML* solver were not tested here, because they are limited to *SBML* files and therefore not applicable to all modules. Since *Assimulo 2.4* provided a better interface and more adjustable parameters, we decided to use this package for test simulations. Both of

the two implemented solver methods were able to solve the implemented test systems accurately.

The first possible error source of the simulation environment is the repeated call of the solver. A simulation step in the model framework is consisting of the simulation of each module for a short time step Δt and the following consolidation run. Therefore, to solve a module for a certain time t, the solver has to start $t/\Delta t$-times a new simulation. To test the possible evolution of an error by starting the solver again and again instead of running one continuous simulation for the time t, we calculated and plotted the absolute deviance between the two time courses for the *Lotka-Volterra* model. We simulated the model for 20 seconds with a time step size Δt of 0.1 and 0.01, respectively. We tested the two solver methods as well as different orders for the methods.

The comparison of the simulation time courses showed a 3-times smaller deviation when using the *Adams-Moulton* method, $\Delta t = 0.1$. The error was accumulative and increased with every peak of the *Lotka-Volterra* model. Since a multistep method uses previously calculated points for the approximation of the following point, a frequent restart of the solver was thought to increase the error. The decrease of the time step size Δt was therefore also thought to give rise to a larger deviation. These assumptions could not be confirmed, the reduction of the time step size by the factor 10 has approximately halved the maximum deviation from 0.1 % to 0.05% of the particular species value. A change of the order has not shown any effect on the deviation between the simulations.

Error estimation for the decoupled simulation of modules

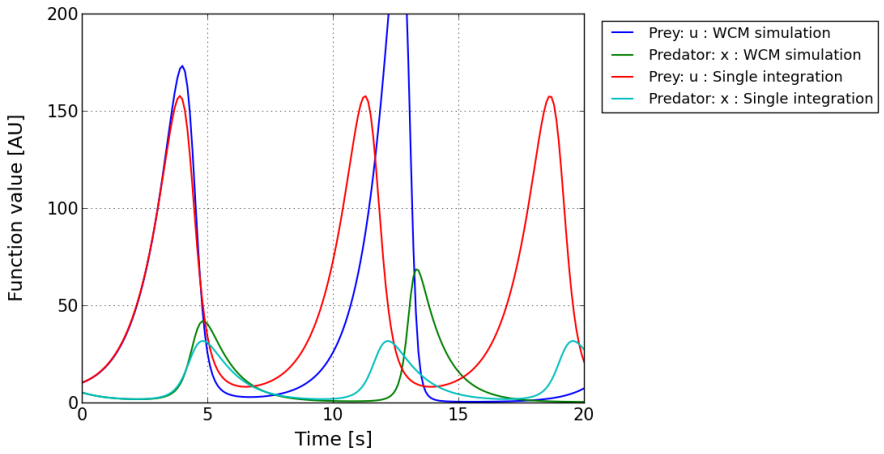
To characterise the error made by the decoupling of the module simulations, we separated each test systems into two modules and compared the simulated time courses with the time courses shown in Figure 5. The modules for the test systems based on mass action kinetic hold one of the equations shown, we provided the same identifier for the species to realise the coupling. Each module of the *Lotka-Volterra* test system had to contain both species, therefore, we introduced the second species as constant. The value of this "constant" species is only changed by the consolidation function of the simulation framework:

Lotka-Volterra test modules:

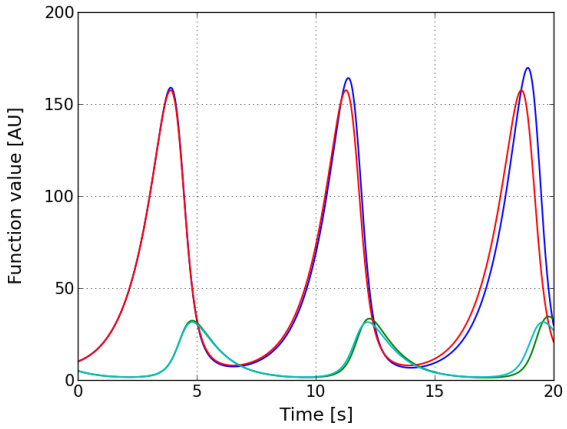
$$\text{Module I}: \quad \frac{\mathrm{d}u}{\mathrm{d}t} = \mathrm{k}_1 \cdot u - \mathrm{k}_2 \cdot u \cdot x$$
$$\frac{\mathrm{d}x}{\mathrm{d}t} = 0$$
$$\text{Module II}: \quad \frac{\mathrm{d}u}{\mathrm{d}t} = 0$$
$$\frac{\mathrm{d}x}{\mathrm{d}t} = \mathrm{k}_3 \cdot u \cdot x - \mathrm{k}_4 \cdot x$$

The simulated time courses for these two modules, representing the *Lotka-Volterra* model, are shown in Figure 6. The time courses are plotted together with the reference time course, which was simulated with the same solver method, *Assimulo 2.4*, *Adams-Moulton* method outside the simulation framework. The modular system shows a clear divergence, the oscillation peaks are shifted and increase over time, Figure 6b. The plots show the simulations with the use of different time step sizes Δt, ranging from 0.1 seconds to 0.001 seconds, the decrease of the step size decreases also the deviation and therefore the error being made by the simulation framework. The *Lotka-Volterra* model represents here a strongly coupled system, the two components are highly dependent on each other, the decoupling leads to a high deviation accumulating over time.

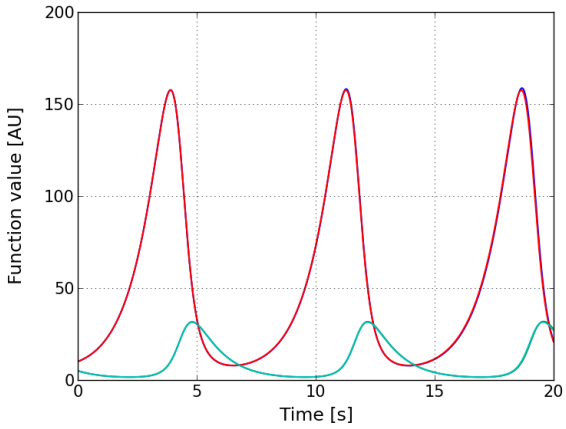
The second test system is based on a species x, which occurs in two different modules with different reactions. During the decoupled simulation, each modules performs its own change of the species concentration. After the simulation of both modules for the given time interval Δt, the consolidation function merges these changes. The choice of the time step size defines the frequency of the consolidation steps. Figure 7 shows the absolute difference between the reference time course, simulated in one equation outside the framework and the test system, separated in two equations and two modules inside the framework. At the beginning of the simulation, the species increases very fast, leading to a high deviation of the solutions. This deviation decreases when the species changing rate decreases and stayed at a constant value when the species reaches its saturation level. Figure 7c shows also an erratic movement of the deviation, the decoupled system shows irregular oscillations of the species function value and was not able to reach a stable steady state. The oscillations have not exceed 0.01% of the species value for all tested time step sizes. The choice of the time step size is also in this example directly proportional to the deviation of the simulation. The reduction of the time step size Δt by the factor 10 reduced the error by approximately the same factor.



(a) Adams-Moulton method, Δt: 0.1



(b) Adams-Moulton method, Δt: 0.01



(c) Adams-Moulton method, Δt: 0.001

Figure 6: Framework test using *Lotka-Volterra*

The *Lotka-Volterra* model represents a highly coupled system. The decoupling of the two species leads to the divergence of the solution from the reference time course. The deviation can be reduced by decreasing the time step size Δt of the simulation framework.

The 2 examples showed the dependency of the error on the time step size Δt. The approach profits from the form of biochemical reaction equations. An ordinary differential equation for a chemical species is based the rates of the corresponding enzymatic reactions. The system showed in Figure 4 is based on 4 reactions v_1 to v_4, the actual mathematical description is based on the chosen rate law. The second example demonstrates the use of mass action kinetics but also other descriptions are possible. The differential equation itself is formed as the sum of rates, here denoted as *v*. The simulation framework does not split the individual reaction, but the reactions were decoupled for a given system:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \mathrm{k}_1 + \mathrm{k}_3 - (\mathrm{k}_2 + \mathrm{k}_4) \cdot x = \mathrm{v}_1 - \mathrm{v}_2 + \mathrm{v}_3 - \mathrm{v}_4$$

The example demonstrates a scenario with 2 different modules, one module (Y) holding the reactions 1 and 2 and one module (Z) containing the reactions 3 and 4. The equation of the species x is therefore split into the equations for the matching species y and z:

$$\frac{\mathrm{d}y}{\mathrm{d}t} = \mathrm{v}_1 - \mathrm{v}_2$$

$$\frac{\mathrm{d}z}{\mathrm{d}t} = \mathrm{v}_3 - \mathrm{v}_4$$

To exemplify the mechanism of the decoupling, we assume the actual solver method to be *explicit Euler* [43]. The error of the solver method can here be neglected, since the error based on the decoupling is several times higher. The mechanisms are still valid for higher-order methods, like the used linear multistep methods.

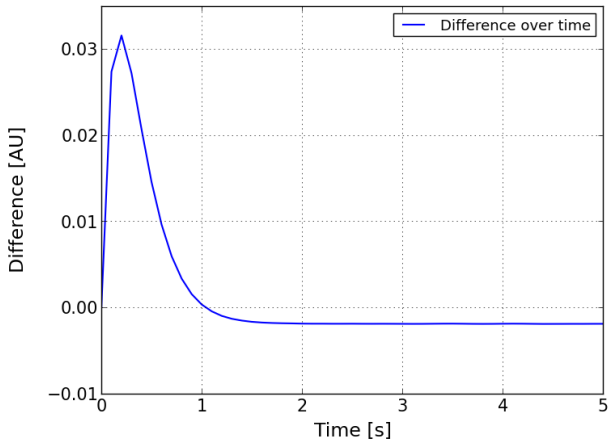
Firstly, we assume the solver step size, usually denoted as h, to have the same size as the step size of the simulation framework Δt. The approximation of the solution for the species x at the discrete time point $t_{n+1} = t_n + \mathrm{h}$ is given as:

$$x_{n+1} = x_n + h \cdot f(t, x(t)) = x_n + h \cdot (\mathrm{v}_1 - \mathrm{v}_2 + \mathrm{v}_3 - \mathrm{v}_4)$$

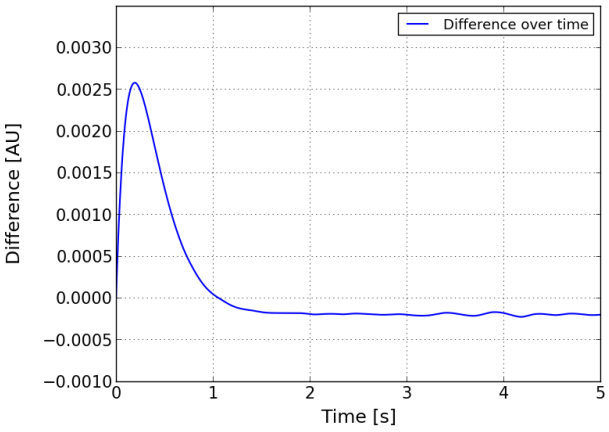
Application of the *Euler method* to y and z leads the following equations for the approximation:

$$y_{n+1} = y_n + h \cdot f(t, y(t)) = y_n + h \cdot (\mathrm{v}_1 - \mathrm{v}_2)$$

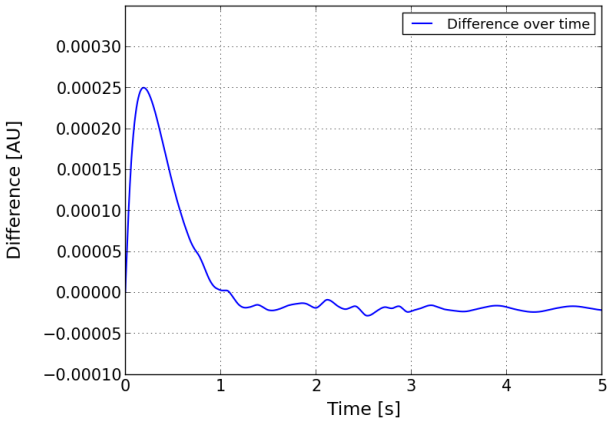
$$z_{n+1} = z_n + h \cdot f(t, z(t)) = z_n + h \cdot (\mathrm{v}_4 - \mathrm{v}_4)$$



(a) Adams-Moulton method, Δt: 0.1



(b) Adams-Moulton method, Δt: 0.01



(c) Adams-Moulton method, Δt: 0.001

Figure 7: Framework test using mass action kinetics

The simplified mass action test system was separated into 2 modules. The figures show the differences between the modular simulations and the reference time course. The deviation was increased by the fast movement of the species and decreases with the changing rate of the species. The modular simulation shows also irregular oscillations of the solution when the reference has reached its saturation level. The reduction of the time step size Δt reduced the deviation by approximately the same factor.

The consolidation function guarantees a consistent state for the species x before the start of a simulation cycle, the initial values are therefore identical: $y_n = z_n = x_n$. After the uncoupled simulation, the changes of x will be summed up by the consolidation function and added to the value in the state vector:

$$\Delta y = y_{n+1} - y_n = h \cdot (\mathrm{v}_1 - \mathrm{v}_2)$$

$$\Delta z = z_{n+1} - z_n = h \cdot (\mathrm{v}_3 - \mathrm{v}_4)$$

$$\Longrightarrow \Delta x = x_n + \Delta y + \Delta z = x_n + h \cdot (\mathrm{v}_1 - \mathrm{v}_2 + \mathrm{v}_3 - \mathrm{v}_4)$$

The last line is identical to the application of the *Euler method* to approximate the value of x at time point t_{n+1}. The decoupling of the modules does not lead to an error in this artificial and simplified scenario, independent from the actual description of the rates v. If the solver takes 2 steps in one simulation step, the second approximation for the values of y_{n+2} and z_{n+2} are dependent on the unconsolidated values y_{n+1} and z_{n+1}. The change of the value made by the uncoupled module is not considered for the approximation. The module is assumed to be constant. Figure 8 shows an exemplified scheme of the behaviour of two different module solutions and their trajectories. For the simulation of the uncoupled solution $\tilde{y}$, the changes of the module Z are not considered, the module is assumed to be constant $\overline{z}$. The difference between the coupled solution $\hat{z}$ and the constant solution $\overline{z}$ leads to the error of the uncoupled simulation of the 2 modules. The same error occurs while the simulation of module Z in regard of module Y, after the two simulations, the consolidation function calculates a consistent solution for the next simulation step. The scheme can be transferred to larger modules and multidimensional trajectories. Of course, the usual case is a scenario, in which the solver makes several or, dependent on the type of the ODE system and the time step size of the framework, several hundred steps during one simulation step of the framework. A precise formula for the error of the decoupled simulations cannot be given in this work, still, as the example already showed, the error is indeed dependent on the mathematical rate descriptions in the decoupled system and the step size Δt of the simulation framework.

For a given time step size Δt, the error of the decoupled simulations is also dependent on the definition of the interface between two modules. The choice of the coupling variables can have a large effect on the error of the simulation. To demonstrate this effect, we separated the glycolysis model by Hynne *et al.* [30] into 2 modules and simulated them together. To show the effect of the coupling variables, we used 2 different species as interfaces between these two modules. The simulation were, as the examples before, performed with *Assimulo 2.4*, *Adams-Moulton* method. Figure 9 shows

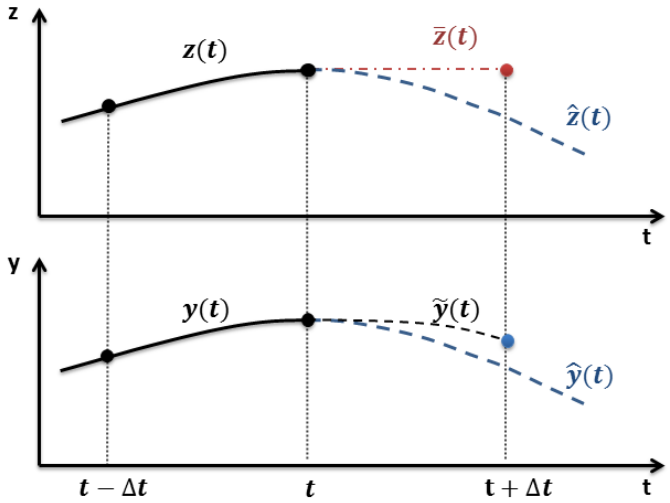


Figure 8: Simulation scheme of a module

During the decoupled simulation of a module Y, the changes caused by the equations in module Z are neglected. The module trajectory is assumed to be constant $\bar{z}$, this leads to the approximated or decoupled solution $\tilde{y}$. The same method is used for the simulation of module Z. After the simulation of both modules, the consolidation function calculates a consistent solution based on the results of the decoupled simulations of the two modules. The difference between the constant solution $\bar{z}$ and the theoretical coupled solution is responsible for the error of the method used by the simulation framework.

3 different simulations of the model by Hynne *et al.*, the plotted time courses show the absolute differences between the reference time course and the related test time course over time. The first plot 9a shows the absolute deviation for a separated model with a time step size of 0.01 seconds. The separation of the model is performed into an upper and lower part of the glycolysis, see Figure 17 for the complete reaction network of the model, the number in brackets refer to the particular reaction number in the network scheme. The first test scenario (system I) is implemented with the species *glyceraldehyde 3-phosphate* as interface between the two modules. All reactions following this species, starting with the *glyceraldehyde 3-phosphate dehydrogenase* reaction (8), were moved into the lower glycolysis module. Since the *ATP* producing reactions are also located in the lower part, the *ATP* consume reaction (23) was also transferred into this module. The same 2 modules were also simulated with a time step size of 0.1 s, a 10-times larger step size (test system II). For the last test, we moved the interface of the two modules to *1,3-bisphosphoglycerate*, the lower glycolysis module starts here with the *lumped phosphoenol pyruvate production* (9), one reaction less than in the example before. We simulated the modules again with the smaller step size of 0.01 s.
Since the glycolysis model is a linear model with strongly coupled reactions, a relatively large error was expected. The plots in Figure 9 confirm this expectation of large absolute

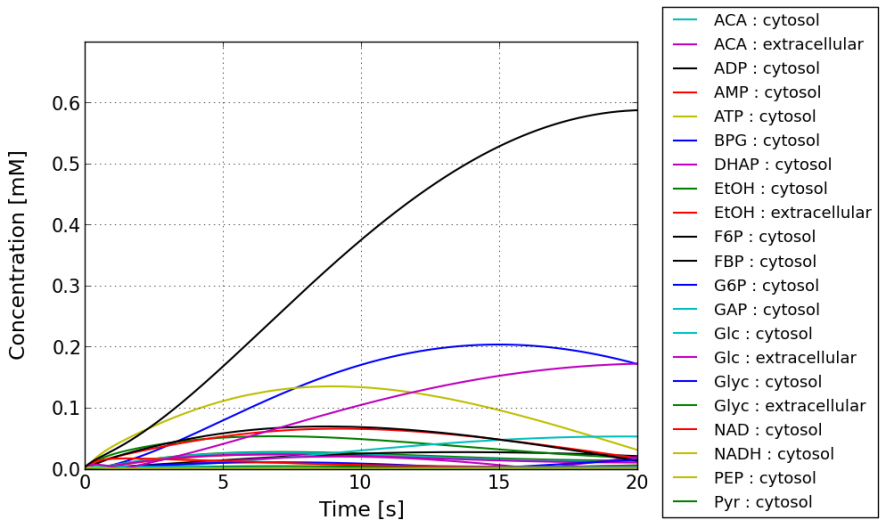
deviations. The difference between the time courses in 9a and 9b also show an expected behaviour, the enlargement of the step size Δt leads to an increase of the error. The time course show a considerably larger distribution as the time course in 9a, also confirmed by the average deviation expressed in percentage, depicted in table 1. The deviation was calculated as the absolute difference between the species concentration after 20 seconds of simulation and the value from the connected simulation as reference. The deviation is expressed in percentage of the particular species concentration from the reference time course. The relative deviation of the species after 20 seconds of simulation is considerably higher, also the average deviation of the whole system. The error of the simulation of the first test system shows that the species involved in the separation, here *fructose 1,6-bisphosphate* and *dihydroxyacetone phosphate*, show the largest relative deviation from the reference time course. The two species of each system with the highest relative deviation are listed, the median is shown because of the better stability in regard of outliers. The species in vicinity of the interface are likely to have a larger deviation than others, since the introduction of the module interface leads to perturbations of the coupled species concentrations. The third test simulation shows a completely different behaviour. Considering the discussions of the examples before, the 10-times smaller simulation step size should have lead to a far more precise simulation compared to the second test system. The contrary behaviour can be observed, the relative deviation of the system to the reference is several times higher and far more species show a large deviation. The plot in Figure 9c as well as the listed values in table 1 verify that the separation of the glycolysis model into the given modules is unemployable. This also shows, that the choice of the interface between two models can be crucial for the successful simulation of a large-scale model with this simulation environment.

Table 1: Separation tests glycolysis model from Hynne *et al.*

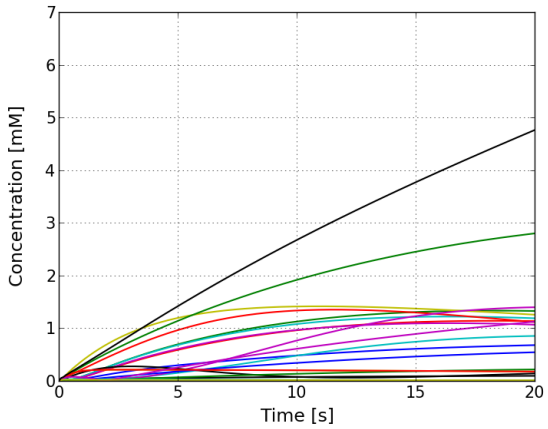
Test system	Δt	Interface	Max. deviation	Species	Average	Median	Figure
I	0.01 s	GAP	12.0 % 5.7 %	FBP DHAP	2.11 %	0.86 %	9a
II	0.1 s	GAP	315.5 % 97.3 %	AMP FBP	48.19 %	27.45 %	9b
III	0.01 s	BPG	2009.5 % 631.4 %	BPG AMP	194.28 %	67.85 %	9c

The values verify the quality of the 3 test systems. The two species with the largest relative deviation to the reference time course after 20 seconds of simulation are listed. Except of AMP, the species denoted here are in direct vicinity of the interface between the separated modules. Figure 9 shows the corresponding time courses of the absolute difference over time.

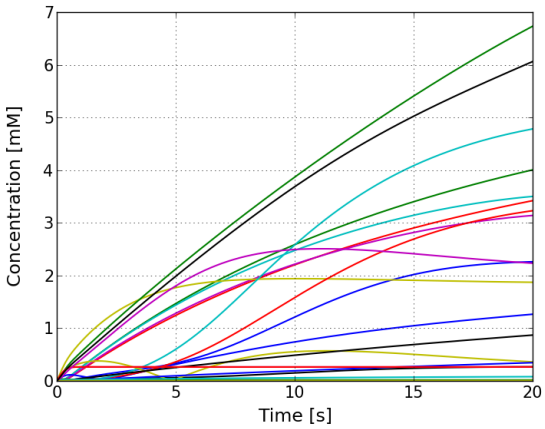
Abbreviations BPG: 1,3-bisphosphoglycerate, DHAP: dihydroxyacetone phosphate, FBP: fructose 1,6-phosphate, GAP: glyceraldehyde 3-phosphate.



(a) Test system I
Absolute difference. Interface: GAP. Δt: 0.01



(b) Test system II
Absolute difference.
Interface: GAP. Δt: 0.1



(c) Test system III
Absolute difference.
Interface: BPG. Δt: 0.01

Figure 9: Simulation tests using the model by Hynne *et al.* from 2001 [30]
Shown is the absolute difference over time between the reference time courses of the continuous glycolysis model and the test model consisting of two modules, representing the upper and lower glycolysis, respectively. Figure 9a shows the simulation for the modules separated at *glyceraldehyde 3-phosphate* (GAP), simulated with a time step size Δt of 0.01 seconds. The plot 9b shows the simulation for the same system with a time step size of 0.1 seconds. The last figure, 9c, represents the modules with a different interface. Here, *1,3-bisphosphoglycerate* serves as connection between the two modules. Although this module was simulated with a smaller step size of 0.01 s, this model shows the largest error of the three test systems.

Abbreviations ACA: acetaldehyde, BPG: 1,3-bisphosphoglycerate, DHAP: dihydroxyacetone phosphate, EtOH: ethanol, F6P: fructose 6-phosphate, FBP: fructose 1,6-phosphate, G6P: glucose 6-phosphate, GAP: glyceraldehyde 3-phosphate, Glc: glucose, Glyc: glycerol, PEP: phosphoenol pyruvate, Pyr: pyruvate.

5.3. Central carbon metabolism in yeast

The central carbon metabolism is a part of one of the functional units for the whole cell model of yeast. Therefore, it can serve as an example for the implementation process and as indicator for limitations and missing functions of the simulation framework. We tested different possible methods to create a functional entity for the whole cell model in regard of the usability, required effort and possible error sources.

Definition of the functional unit

A crucial and formative part of the characterisation process is the definition of interfaces between different functional units. Concerning the metabolic model, we identified the main outputs of the metabolic unit to be biomass (anabolism) and energy (catabolism) [56]. The technical implementation of these interfaces is highly dependent on the architecture and granularity of the other functional units. We limit the central carbon metabolism in this test approach therefore to catabolic reactions and the production of *ATP* and *NADH*, representing the storage molecules for free energy [56] and therefore the energy yield of the model. Biomass production as lumped reaction or any other biosynthesis reaction of biomolecules are indicated, but not fully implemented in this test model. The main inputs for this unit were assumed to be nutrients from the extracellular medium, enzyme concentrations based on the gene expression and transcription of the metabolic proteins, and general cell dependent parameters. These parameters can be influenced by environmental changes, but also by the cell division stage or simply the growth of the cell, e.g. size changes and related dilution of molecule concentrations. Since we do not consider any other functional units beside the metabolic unit for our test, we omitted these inputs. Only the uptake and export of metabolites and nutrients from the extracellular medium are implemented.

After the characterisation and definition of the main inputs and outputs of the functional unit, the technical implementation of the modules begins. We tested 3 different methods in regard of their usability to gain a working metabolic model for the simulation environment. After the characterisation of the interfaces for the functional unit, this unit is still represented by a black box in the large-scale model. This black box can be filled by a single equation or a small set of lumped equations just to connect the inputs and outputs of other units. The use of single equations as placeholders can be used for testing purposes, but represents also the lowest attention to detail of a functional unit. The definition of the granularity of single modules and functional units is an important aspect for the whole cell modelling approach. The modularity of the model allows for the facilitated exchange of modules and therefore for an easy change of granularity concerning the whole cell approach. Therefore, also the representation of a module by a single

equation can be understand as a reasonable implementation. Regardless of the granularity and the definition of interfaces is the choice of the mathematical description of the different modules. Although the modularity of the approach allows for different mathematical descriptions in general, we decided to start with the utilisation of ODE-based modules only. Since ordinary differential equations are a commonly used mathematical descriptions for enzymatic reaction networks [6], this is not a limitation of the model for the central carbon metabolism.

Definition of the modules

We identified 3 possible ways to implement a functional unit as model for the central carbon metabolism. The representation of the module by a single lumped equation is omitted, since this representation is only reasonable in a scenario consisting of several functional units. We defined a set of 3 main pathways to be the basis of this model: the glycolysis, the tricarboxlylic acid (TCA) cycle and to some extend the oxidative phosphorylation (respiratory chain). Figure 11 shows a scheme of the central metabolism in yeast, containing the pathways we want to be represented in the test modules. These pathways are spread over 4 different compartments, the extracellular medium, the cytosol, the mitochondrial matrix, and the intermembrane space, between the inner and outer mitochondrial membranes. For simplicity, the oxidative phosphorylation can technically be seen as a transport reaction. Therefore, we extended the module with the addition of a general transport module. Besides the merging of a few reactions we wanted to keep the attention to detail as high as possible. After the definition of the interfaces, the mathematical description and the granularity of the different modules, we started the implementation with 3 different approaches:

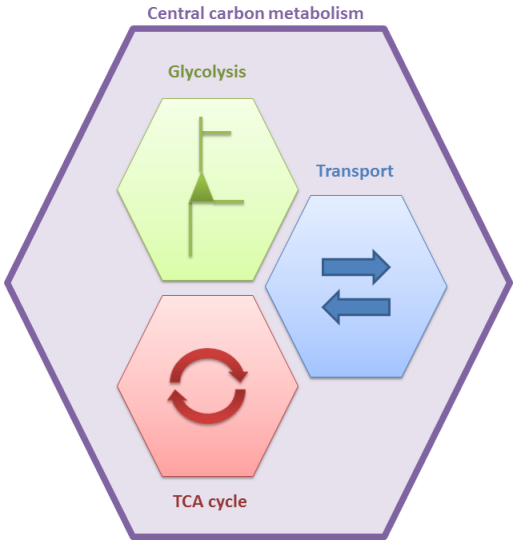


Figure 10: Scheme of the model

The model of the central carbon metabolism is represented by a set of 3 modules: the glycolysis, the TCA cycle and a general transport module, including the oxidative phosphorylation.

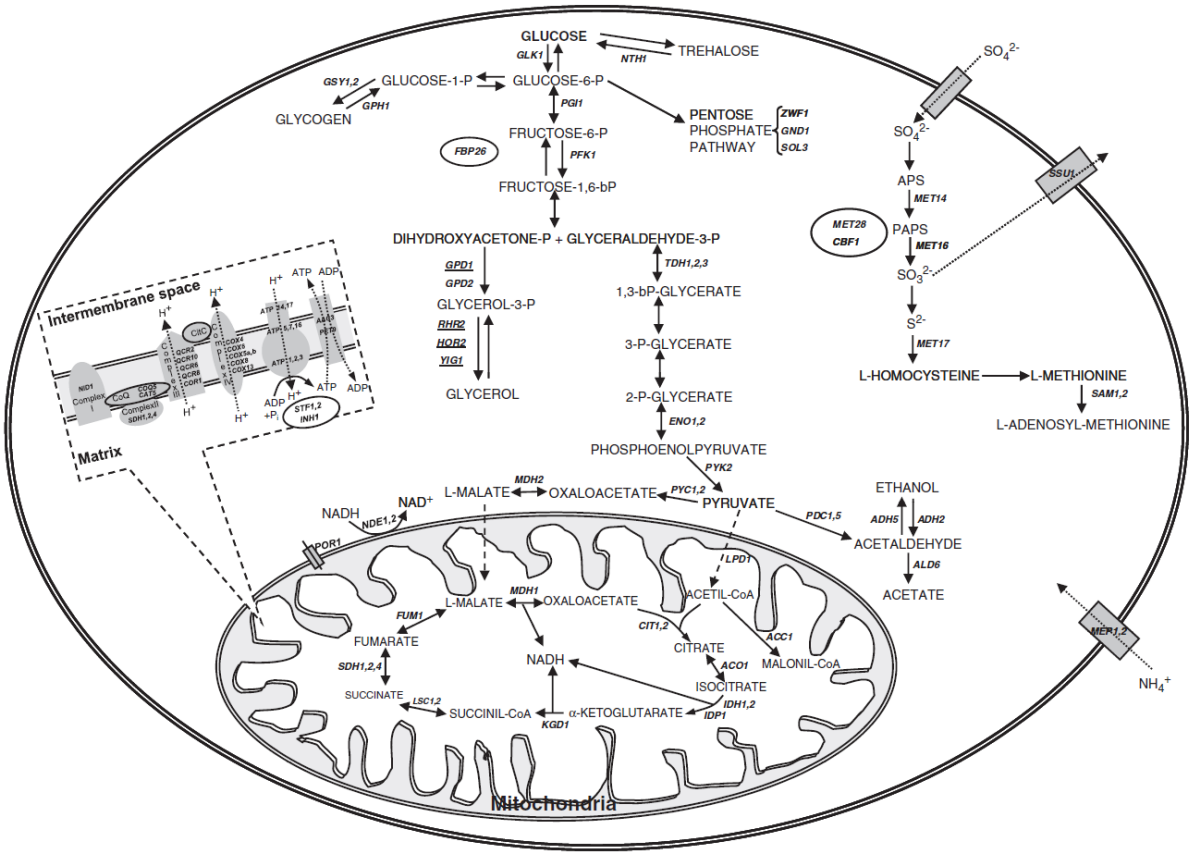


Figure 11: Scheme of the central metabolism in yeast

Figure is taken from Jiménez-Martí *et al. 2011* [57]. The figure shows a scheme of the main pathways in the central metabolism of the Baker's yeast *Saccharomyces cerevisiae*. We defined our model of the central carbon metabolism the following modules: The glycolysis module, consisting of the uptake of glucose over the plasma membrane and the subsequent degradation to pyruvate. This module also includes the formation of glycerol and glycogen. The tricarboxylic acid (TCA) cycle module, consisting of all reactions localised in the mitochondrial matrix. The transport module, containing the oxidative phosphorylation (respiratory chain) on the inner mitochondrial membrane, the production of ethanol and acetate in the cytosol and all transport reactions over the mitochondrial membranes and the plasma membrane.

An existing large-scale model

Besides the availability of computational models describing single pathways and small functional parts, also some large-scale models are present in the systems biology community. These large-scale models could substitute a whole functional unit in the whole cell model. However, such a large-scale model has to satisfy far more requirements than a small model being used as a module. The aim of the model itself, the inputs and outputs of the model, as well as the architecture of the model have to match. The modification and adjustment of a continuous large-scale model is in most cases a time consuming and tedious process.

We tested this approach with a larger model (285 reactions) representing the carbon metabolism by Stanford *et al.* [35]. This model is based on the *consensus yeast metabolic network* [36]. Glucose was assumed to be the only energy source and a "thermodynamically feasible, stationary flux distribution matching the flux data" [36] was defined. The model only contains reactions which showed an active-flux in this flux distribution, all reactions without an active-flux were deleted. Afterwards it was automatically provided with rate laws and parametrised using the *parameter balancing* tool [37]. The model also contains only 2 compartments, the extracellular and the intracellular medium and is lacking the oxidative phosphorylation. To use this model as a feasible functional unit for the whole cell approach several modifications would have to be performed:

- Identification of species and reactions
- Introduction of compartments and corresponding transporters
- Localisation of species and reactions
- Addition of missing reactions to enable the model to describe different experimental conditions and use different energy sources
- Re-parametrising the modified model
- Partitioning of the model into modules to regain the advantages of the modular approach

Max Schelker wrote a new specialised *Python* program to enable the framework to import this particular *SBML* file and to allow for the identification of species and reactions. Together with some members of the group of theoretical biophysics it was possible to identify the reactions and the corresponding genes and enzymes by hand. We were able to localise the reactions using the *Saccharomyces Gene Database* (SGD) [58] and to identify most of the missing reactions and transporters. Although this model can be

simulated using the whole cell model framework, it is still very limited because it cannot profit from the main advantages of a modular approach. After adding all necessary components, adjusting the architecture and splitting the model into modules, it would still need to be re-parametrised to regain the functionality. Since the model provides a biological feasible biomass function to model biosynthesis based on the used metabolites, it could nevertheless be used as a substitute for the metabolic unit.

This example reveals again the advantages of a modular whole cell approach, the use of small modules can guarantee a comprehensible and easily modifiable large-scale model. Without the need of modifications, the partitioning of a large-scale model into modules would be superior to other methods, since this method guarantees consistency and compatibility of the modules. However, without a large-scale model satisfying the given requirements, the adjustment of smaller models could appear to be more feasible to gain a functional unit more compatible with the requirements of the simulation framework.

Merging small models to a large-scale model

The second test implementation we performed was the utilisation of smaller computational models as building blocks for the functional unit. We found a glycolysis model by Hynne *et al.* from 2001 [30] and two mitochondrial models from 2007 (Wu *et al.* [33]), and 2009 (Nazaret *et al.* [32]). The mitochondrial models were not specified to yeast, the model by Wu *et al.* represents a rat tissue mitochondrion, the model by Nazaret *et al.* a generalised mitochondrion. We prepared both mitochondrial models to test the exchangeability of the modules. Since the modules showing different granularities, we also wanted to investigate the best suitable module for the given task. We identified several modification and adaptation steps to employ the models in the simulation framework and to use them as building blocks for a model of the central carbon metabolism.

- Annotation of species and compartments
- Creating valid module files for the framework
- Equalising the units for concentration and time
- Adaptation of the modules to *S. cerevisiae*
- Addition/removal of species and reactions to build valid interfaces between the modules
- Re-parametrising the modified modules

While the glycolysis model by Hynne *et al.* and the mitochondrial model by Nazaret *et al.* provided a valid *SBML* file, the mitochondrial model by Wu *et al.* was only available as *MATLAB* [16] file and had to be translated by hand into a *Python* script. Additionally, it was not annotated and the other modules needed to be curated as well, since different annotations were used for matching species. During the modification of the modules, we encountered several algebraic equations in the model files, since the provided solver is not able to work with algebraic equations, we had substitute the representing variables with the algebraic formulas in the differential equations. The algebraic equations were mainly used for mass conservation, some algebraic equations were therefore eliminated by the addition of the corresponding species into the modules. A more technical modification process was based on the units of the models, while one module used *minutes* and *millimolar*, the others used *seconds* and *molar*. We adjusted the kinetic parameters to *millimolar* as the global unit for concentrations and divided the differential equations of one module by 60 to adjust the module to seconds as the global unit for the time. After this adjustment, the modules were technically operational, the framework was able to import and initialise the models correctly. The second part of the adaptation process concerned the biological aspect of the modules. The model by Hynne *et al.* is closely connected to an experimental scenario, therefore, cyanide (CN^-) was occurring in the model. We removed the species and the corresponding reactions, this was easily doable, since the species was only present at one edge of the model and outside of the cell, as can be seen in Figure 17 in the appendix of this work. Also the mitochondrial models had to be changed, they were not representing a yeast mitochondrion correctly. Firstly, complex I of the respiratory chain in *S. cerevisiae* is substituted by an internal (*NDI1*) and an external (*NDE2*) *NADH dehydrogenase* [59], [58]. These enzymes are not involved in the formation and maintenance of the proton gradient over the internal mitochondrial membrane, since they are lacking a proton pump. We therefore removed the complete respiratory chain from the models and Matthias Reis set up a new module concentrating on the connection of glycolysis in the cytosol and the TCA cycle in the mitochondrial matrix. The scheme of this connecting model can be seen in Figure 20 in the appendix of this work.

After the modification and adaptation of the models to the technical requirements of the simulation framework and to the biological aspects of *S. cerevisiae*, the new module still needed to be parametrised. Therefore, a feasible time course simulation could not be performed. We tried to identify a working parameter set by hand, but this was not successful in the end. In either way, the whole metabolic model should be re-parametrised to ensure the depiction of a consistent cell condition and to improve the communications

between the modules. The approach to use computational models as building blocks for the whole cell modelling was successful so far. Nevertheless, several considerable modifications had to be performed, considering the inherent limitations of the original models and the need of a re-parametrisation, we wanted to test the implementation of the metabolic model from scratch to compare this approach with the previous examples.

A new implementation

The implementation of a specified model adjusted for the needs of the simulation framework can profit from the advantages of the previous approaches. It can guarantee the connectivity and compatibility of a large connected model as well as the flexibility and specificity of a set of modules. The model can be created with the defined interfaces as main focus, provides an individual granularity and ensures compatibility with the simulation environment. An implementation of a model of this size is nonetheless a large effort and a time consuming process, but in this case, the model can benefit directly from previous attempts, since the metabolic model by Stanford *et la.* [35] presented a way to parametrise a model based on the yeast consensus metabolic network [36] and the definition and extend of the single pathways for the modules could be taken from the definition of the small models.

Indeed, we managed to map the glycolysis model by Hynne *et al.* and the TCA cycle model by Wu *et al.* to the latest yeast consensus model, version 7.0 [60]. The metabolic network can be assumed to contain all relevant information about the reactions and the corresponding metabolites and ensures actuality [60]. We used a graphical representation of the *SBML* file to identify additional transporters and reactions for all related species in the model. With the stoichiometric information of the consensus model, we implemented the complete model with 60 species and 59 reactions in *COPASI 4.11*. After the implementation of the reaction network we exported the model into an *SBML* file and used the *parameter balancing* tool to add kinetic reactions and to parametrise the model. The same rate constants, equilibrium constants and initial values for the parametrising of the metabolic model from Stanford *et la.* were used for the parametrising of our new model. The values were taken from BioModels [61] and eQuilibrator [62]. We extended the list with several entries for the added reactions and species in the new model from the same sources. We achieved a first parametrised version of the new metabolic model, using *modular rate laws* [38] as rate description, a generalised formalism for reversible reactions automatically introduced by the parameter balancing tool.

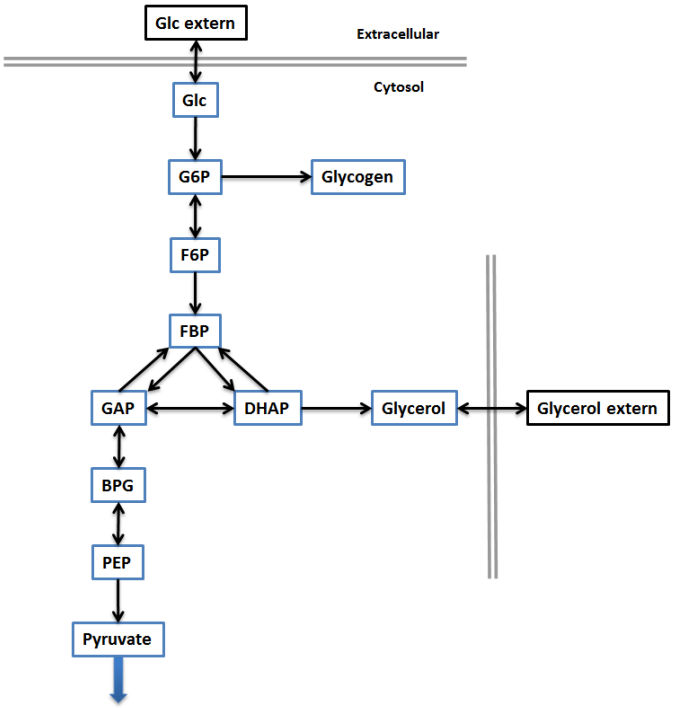


Figure 12: Scheme of the glycolysis module

The figure shows the simplified reaction network of the upper cytosolic pathway. The scheme neglects co-factors and the ATP-based reactions *adenylate kinase* and a reaction modelling the energy consumption of the cell. Furthermore, the membrane transport of protons and carbon dioxide are omitted. The blue arrow at the bottom of the scheme represents the connection to the adjacent transport module. The reactions and species of this module are listed in the tables 2 and 5, respectively.

Abbreviations: BPG: 1,3-bisphosphoglycerate, DHAP: dihydroxyacetone phosphate, F6P: fructose 6-phosphate, FBP: fructose 1,6-phosphate, G6P: glucose 6-phosphate, GAP: glyceraldehyde 3-phosphate, Glc: glucose, PEP: phosphoenol pyruvate

After the parametrising of the model, we separated the model into 3 modules as described before. The first module, representing the glycolysis, starts with external glucose leading to the compound pyruvate. It is mainly a representation of the glycolysis model introduced by Hynne *et al.* from 2001. The model describes the metabolic conversion in the glycolysis and two additional branches, the storage of glucose 6-phosphate in glycogen and the production of glycerol starting at dihydroxyacetone phosphate. Figure 12 shows the simplyfied reaction network scheme.

The mitochondrial part of the model represents the TCA cycle and holds all reactions of the model which take place in the mitochondrial matrix. To simplify the model and reduce the amount of compartments needed, the localisation of ubiquinone and ubiquinol was assumed to be in the mitochondrial matrix to reduce the number of compartments. The oxidative phosphorylation was split into the transport module and the TCA cycle module, since the *succinate dehydrogenase* (complex II) is also part of the TCA cycle, this reaction was transferred to this module to close the TCA cycle [56]. Figure 13 shows the network scheme of this module.

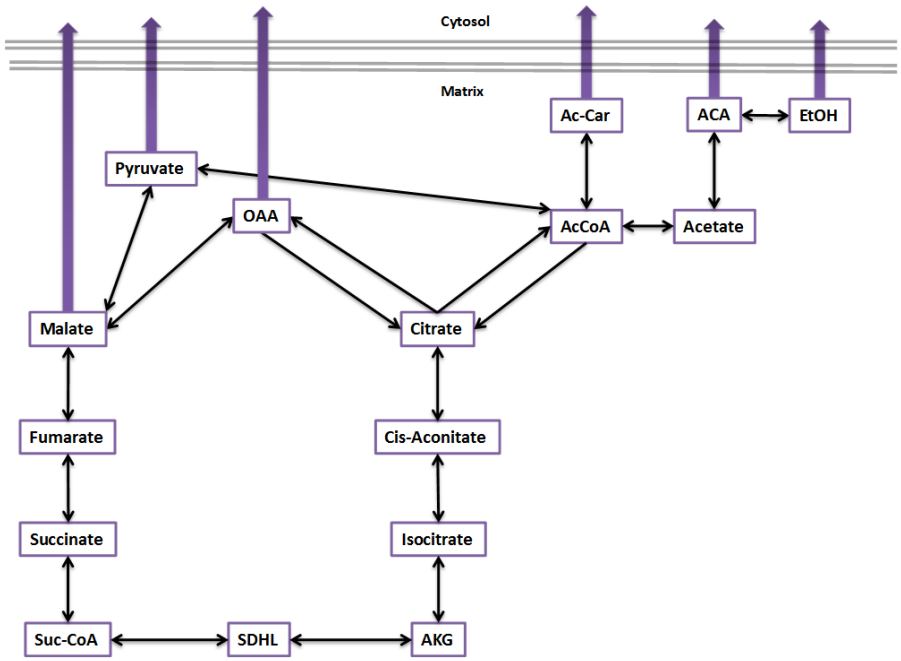


Figure 13: Scheme of the TCA cycle module

The network scheme depicts the reactions of the mitochondrial matrix module. This module holds all reactions localised in the mitochondrial matrix, additionally, the *succinate dehydrogenase* (complex II) was transferred to this module to close the TCA cycle. Co-factors are omitted in this scheme.
The reactions and species of this module are listed in the tables 3 and 6, respectively.

Abbreviations:

ACA: acetaldehyde, Ac-Car: acetyl-carnitine, AcCoA: acetyl-CoA, AKG: α-ketoglutarate, EtOH: ethanol, OAA: oxaloacetate, SDHL: S-succinyl-dihydrolipoamide, Suc-CoA: succinyl-CoA

The connection between the 2 modules is the transport module. Firstly, it holds the cytosolic reactions of pyruvate to ethanol, secondly the transporters over the plasma membrane, and thirdly the mitochondrial transport reactions, including the respiratory chain. To simplify the model and to reduce the number of compartments, the mitochondrial intermembrane space was omitted. This decision was encouraged by the increased permeability of the outer membrane [56]. A further assumption was made concerning the respiratory chain, which was modelled without considering the membrane potential over the inner mitochondrial membrane. Figure 14 shows the schematic network of the module.

The reactions and species of the 3 modules are shown in the tables 2 to 4, and 5 to 7 in the appendix of this work.

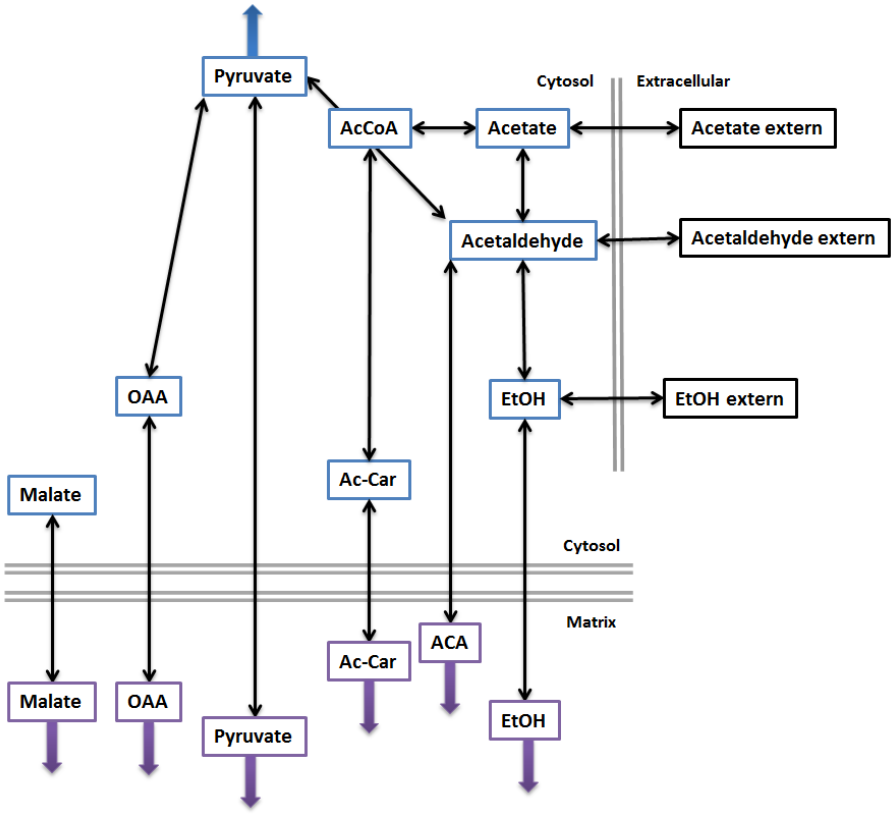


Figure 14: Scheme of the transport module

The transport module consists of the transport reactions between the cytosol and the mitochondrial matrix or the extracellular medium, respectively. It does not show co-factors and co-factor transporters, also the mitochondrial *adenylate kinase* and the reactions of the oxidative phosphorylation (respiratory chain) are neglected in this figure.

The reactions and species of this module are listed in the tables 4 and 7, respectively.

Abbreviations: ACA: acetaldehyde, Ac-Car: acetyl-carnitine, AcCoA: acetyl-CoA, EtOH: intracellular ethanol, OAA: oxaloacetate, Pyr: pyruvate.

To provide a constant energy source, extracellular glucose was set as constant, also the concentrations of protons, carbon dioxide, acetaldehyde, acetate, ethanol, pyruvate, and glycerol in the extracellular medium. These components serve as sinks and sources outside the cell to balance the network. Malate in the cytosol is the only constant species inside the cell and serves as a sink/source for the TCA cycle. This species could be used as a precursor of biomass in the model.

Despite the parametrisation based on literature values and the employment of the *yeast consensus network* to build the model, we were not able to gain a feasible time course simulation, not with the connected *SBML* file, nor with the modular model. The model was not able to stabilise the ATP/ADP and the NADH/NAD ratios. We tested several possible error sources, changed the number of reactions and species in the model as well as changed the number of given values for the parameter balancing. Even the manually adjustment of rate parameters of the main reactions, involved in the formation and degradation of these species, were not able to change these ratios considerably. It is assumed, that the parameters of transport reaction cannot be gained using the parameter balancing tool and that these reactions disturb the successful parametrising of the model.

6. Discussion

The presented simulation environment for a whole cell modelling approach and the implementation of a metabolic model representing the central carbon metabolism in *Saccharomyces cerevisiae* seem to be unrelated at a first glance. On closer inspection, however, the strong connection between the model and the framework in this approach becomes clear. The given examples showed, that this simulation environment is not meant to be another highly accurate simulation environment for differential equation-based simulations. Instead, this framework is a helping tool for the implementation and simulation of a whole cell modelling approach and exchanged some of its accuracy for compatibility and functionality as a modelling tool specialised for this purpose. The implementation of the central carbon metabolism exemplified different possible methods to create a working model for the use in this whole cell modelling approach and for the simulation in the specified environment. Even if we are not able to present time course simulations, we were able to demonstrate the advantages and limitations of our approach in regard of the model implementation as well as in regard of the simulation framework.

6.1. Implementation of a simulation framework

This early version of the simulation environment demonstrated successfully the underlying concept of this approach and its working principle. The function of the framework is to control and coordinate the module simulations, matching of species, administrating the current cell state, and processing the simulation data. We were able to show the benefit of utilising standardised annotations to identify module components unambiguously, the use of *ChEBI* ids was functional for metabolic models, still, some metabolites do not provide such an identifier and most components have several ids, depending on the current charge. ATP for example, provides more than 8 different *ChEBI* identifiers. This makes the control instances of the framework and visualisation tools for the models and the contained species inevitable. The text-based output to visualise the species matching (table 22) by Dr. Martin Seeger is one example of a small tool, that can increase the functionality and usability of the whole framework. Another example is the provided validator to check if a Python-based module can be initialised by the framework, these validation tools help the successful implementation and connection of modules and need to be extended. When a set of modules was successfully imported and initialised, the framework has shown no instability or unforeseen problems. The matching of species and the consolidation of simulation results work in this first version absolutely reliable. Initialisation of modules is facilitated by the clearly structured format of the *Python* scripts for storing module information, also the use of *Python* dictionaries supports the representation of modules in *Python* scripts. The *SBtab* format is a powerful tool to provide numerical data and parameter sets for the simulations. So far, the only applications of *SBtab* are the import and export of modules and the provision of the initial cell state. Every kind of data in the simulation framework can be exported using *SBtab* tables. A web parser for *SBtab* files into *SBML* files, provided by *semanticSBML* [40], turns *SBtab* into the only exchange format for modules to create *SBML* files.

Error estimation and reduction

The emergence of numerical errors during the simulations in the modelling framework cannot be avoided and is an inherent property of the simulation environment, due to the decoupled simulations of modules. We were able to demonstrate, that the error can be influenced by the definition of the module interfaces. Strongly coupled species with a high changing rate like the species in the *Lotka-Volterra* test model, leading more likely to a large numerical error, which can only be reduced by the decrease of the time step size Δt. Also the second example using the test system based on simple mass action kinetics confirmed this observation. Therefore, the error can already be reduced due to a optimised implementation of the modules. The coupling of species with slow components

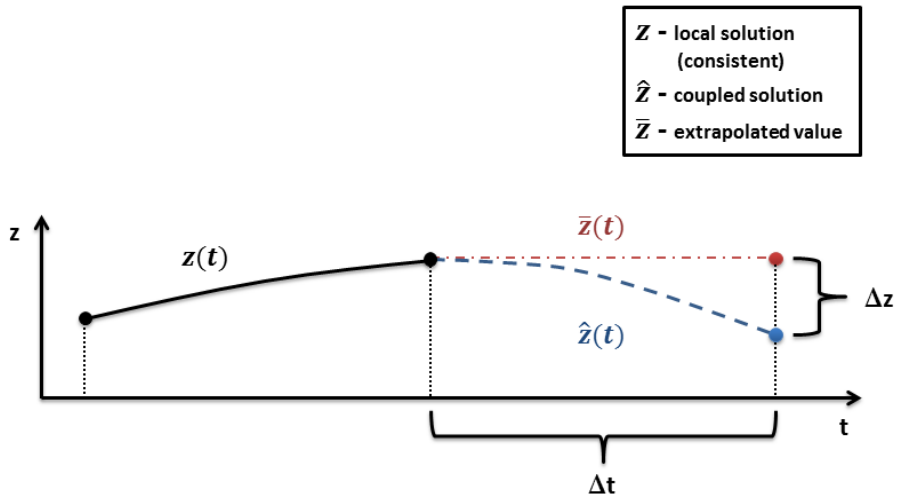


Figure 15: Visualised error

The module Z held constant during the decoupled simulation of other modules. The scheme shows the dependency of the error on the time step size Δt and the changing rate of the species z. The deviation Δz of the species is the error source of the decoupled simulations. The precise numerical value of the resulting error is dependent on the formalism of the rate equation as well as on the number of modules containing the particular species.

can reduce the error and increase the stability of the simulations considerably. Figure 15 shows the dependency of the simulation error on the time step size of the simulation and changing rate of the decoupled species. The deviation Δz is direct proportional to the actual error made during the simulation of this species, its precise numerical effect is thereby dependent on the formalism of the rate equations and the number of modules containing the particular species.

The test simulations using the glycolysis model by Hynne *et al.* demonstrated the effect of differently defined interfaces. A shift by one reaction of the interface between the test modules lead already to a considerably higher error of the simulation. It has to be assumed, that a higher sensitivity of the reactions in regard of disturbances in metabolite concentrations are responsible for this effect. The implementation of analysis tools to investigate modules in regard of optimised interfaces could facilitate the implementation of models and reduce the simulation errors.

The reduction of the time step size Δt is at the moment connected with an increase of the simulation time by approximately the same factor. Simulations with a time step size lower 0.01 seconds are therefore aggravated, the simulation of the glycolysis model by Hynne *et al.* for a time interval of 120 minutes (7200 seconds) can take several

hours (Intel quad-core 4 x 2.8 GHz). This is based on the unoptimised implementation of this first version of the framework. It has to be assumed, that an optimisation of the simulation environment and the related increase of the computational efficacy reduce the simulation time by the factor 1000 or even more. Due to different experiments with functions and calculation methods several redundant processes emerged, especially concerning the consolidation function. The exchange of information between the framework and the solvers is also not efficiently implemented. In every simulation step of the framework the solvers have to reload the whole ODE system, instead of only changing the initial values.

We also compared different solver methods to identify possible influences on the simulation process. As already mentioned, the advantage of a solver method has to be tested dependent on the module and the representing ODE system. It was assumed, that the use different orders of the solver methods can influence the accuracy of the simulation results. The reason is the start of the ODE solver in every simulation step of the framework. Since the initial values change after each consolidation step, the solver cannot use previous simulation results for the approximation of the next data point when starting a new simulation. Therefore, the solver can only use a first order method to calculate the first step, a second order method for the second step and so on. An increase of the error could still not be confirmed, no deviation between the simulation results could be identified. It has to be assumed, that our test modules were to simple to detect a deviation between the methods using different orders. The implementation of more complex test modules is necessary to identify the error sources and limiting steps reliably.

6.2. Different approaches to implement a metabolic model

We presented 3 different methods to implement a functional unit for the whole cell modelling approach, the use of a single equation to substitute for the whole unit is omitted at this point. All 3 methods could be identified as possible successful approaches, dependent on the available models and in either way connected with the requirement of intensive care. The last and decisive step, the simulation of the different modules and the predictive power in regard of experimental results could not be performed. The main reason was the absence of a reliable parameter set. This showed again the necessity of a parameter estimation tool to reliably parametrise the different modules. Nevertheless, several advantages and limitations of the different methods could be identified.

The decision to use a large model and split it afterwards into modules has two major advantages. Firstly, the model uses the same granularity, which ensures a high compatibility of the modules. Furthermore, the modules are consistent in regard of any model property, e.g. units and underlying assumptions. Secondly, the parametrising can be performed with a consistent and connected module, instead of estimating parameters only for a single module, or for a merged set of individual modules. If a large-scale model for a certain cell process is available, the use is at an advantage compared to the other methods. The model architecture of the presented model by Stanford *et al.* was not compatible with the requirements for the simulation framework, which made the adjustment of the complete model necessary. The introduction of compartments, additional species and reactions results in the need of a re-parametrising of the whole model.

The merging of small modules is much easier but can lead to severe problems, due to inconsistent interfaces, varying cell conditions or simply different underlying assumptions. We provided two different set of modules, using a different model of the TCA cycle. The model by Nazaret *et al.* is a very simplistic model of the mitochondria, but provided a better interface with the glycolysis model of Hynne *et al.* as the TCA cycle model by Wu *et al.*. The latter was to detailed to be connected easily to the glycolysis module, the module was based on the export of several mitochondrial metabolites, as seen in Figure 19. The connection of the glycolysis module with any of the TCA cycle modules as well as the adaptation of the modules to yeast, required the creation of a new module, containing the reactions between the modules and connecting the related species. After the introduction of the new module, at least this new module had to be parametrised. A set of modules using the TCA cycle model of Nazaret *et al.* was connected successfully and could be used for re-parametrising.

The creation of a new model was facilitated in this case by the availability of the *yeast consensus metabolic network* as a basis for the reaction network. The new implementation of a whole model can be reasonable, especially to adapt the model to defined interfaces. Here, the parameter balancing tool was able to set up kinetic equations for the model and provided a set of parameters, still, it was not possible to simulate the model. The reasons could be the already mentioned difficulties to estimate kinetic parameters for transport reactions in a thermodynamic-based parameter balancing tool, since the energy of formation for "products" and "educts" is identical. It is also possible that the set of experimental rate constants to calculate the parameters was just too small or incorrect. It can also not ruled out, that the reaction network is incompatible with the expectation of a stable steady state, due to the absence of crucial reactions, even if these reactions could not be identified.

In the end only an experimental data set could decide for the best suitable model for our approach. For the moment, all 3 possibilities have lead to a model that need to be re-parametrised. In regard of the defined interfaces for the metabolic functional unit, the model based on the work by Stanford *et al.* has the advantage of a fully integrated biomass function.

6.3. Outlook for the whole cell modelling approach

Optimisation and error reduction

This first version of the simulation framework was able to demonstrate key features and the basic functionality of a simulation environment for large-scale modelling. The main issue is without question the size of the numerical error made by performing decoupled simulations of the different modules. The best possible interface definition presumed, the error can in principle be reduced by decreasing the simulation step size Δt of the framework. As discussed before, the optimisation of the simulation framework can considerably reduce the simulation time and therefore enable the use of time step sizes below 0.01 seconds. The implementation of the solver library *SOSlib* [45] shows a far more efficient simulation, with a runtime in the range of seconds even for smaller time step sizes. This library employs the same solver *CVODES* [48] and solver method *Adams-Moulton* or *BDF* [44], respectively. Therefore, this library could serve as an example for an optimised solver integration. A difference between this library and our solver integration is the programming language, both solver methods are implemented in *C*, but *SOSlib* also uses *C* for the framework of the solver. The highest computational efficacy would be achieved by changing the programming language to a faster one, for example *C* or even *FORTRAN*. Since this would decrease the reusability and modifiability of the framework dramatically, this is only a consideration if the computation time after optimisation of the framework does not drop considerably.

Another possibility to tackle the error of the simulations is to reduce the error source directly. While simulating a module, the other modules are held constant, this leads to a discrepancy because the actual change of the species due to reactions in other modules is not considered during simulation. Δz in Figure 15 depicts this discrepancy. Several methods could be tested to reduce this error. The partitioning of the modules could be performed with the changing rates as criterion, to ensure, that only the smallest changing rates are involved in the coupling of modules. This partitioning would highly increase the effort to implement and integrate the modules and would require a preprocessing analysis of the whole model to identify the different rates.

A different approach is not to change the modules, but to modify the solver method. The solver uses 2 simulations in every step to estimate the error for the *adaptive step size control*. This means, if the difference (error) between the two solutions is larger than a predefined value, the step size has to be reduced to stay in the range of the predefined error tolerance. In principle, a similar mechanism could be used to adjust the step size of the model and to reduce the error of the simulations. A first attempt could be implemented quite easily, the consolidation function in the current version of the framework sums up the changes made by the different modules on a coupled species. If the size of a change made by a single module is larger than a certain threshold, the simulation cycle could be restarted again with a smaller step size Δt. This approach could possibly reduce the error, since it is directly proportional to the changing rate of a single module, also this would need an optimised framework to implement. Another, more complex approach is the use of a fast but inaccurate solver to extrapolate the changes of coupled species for the simulation of a module. This approach could only be realised, if the framework can be optimised dramatically, since this approach would double the number of simulation steps per cycle.

To optimise the performance of the simulation framework we also tested the advantage of the utilisation of different time step sizes for individual modules. The idea was to reduce the computational effort, some modules could be simulated for a longer time step, while others need to be updated more frequently. This would especially be useful for a modularisation by rate, where species with a similar changing rate are stored in the same module[63]. Several software environments were implemented for this *multirate time stepping* [64] approach, for example the *MUltiscale MUltiphysics Software Environment* (MUSE) [65], an astrophysical simulation environment that exploits the existence of modules with different time scales. Modules with a slower time scale can be updated less frequent than modules with a faster time scale, therefore, the software works more efficient. The approach is based on the already mentioned partitioning of the modules dependent on the rates of the reactions. We decided to skip the implementation of this method, since we do not assume to have modules with such a weak coupling, that the update frequency could be reduced to a considerable level. Without this requirement, the necessary effort to implement this approach would not be justified by the savings in computational power.

Use of different mathematical descriptions

To enable the framework to utilise modules with different mathematical descriptions, Dr. Martin Seeger performed first tests with a second vector in the framework. The flag vec-

tor stores information about a discrete cell state, e.g. cell cycle phase. The consolidation function can test the model in every simulation step for a certain condition, by passing the constraints, an initial value can be changed or even the whole parameter set for a module can be substituted to describe the new state. The scheme of this approach is visualised in Figure 16, showing a simulation step of the whole cell model with two state vectors, a continuous and a discrete one. It also shows the implementation of different mathematical descriptions. The use of loosely connected modules would also allow for the utilisation of different kinds of mathematical formulas, e.g. stochastic differential equations, rule based modelling, or Boolean description. These descriptions can be more suitable to certain cell processes as ordinary differential equations [6].

The utilisation of different mathematical descriptions requires also the modification of the state vector, since some of the modules would need the current molecule number instead of the concentrations of the cell components, e.g. a stochastic module would most probably calculate in molecule numbers, instead of concentrations. We implemented a version of the framework with an integrated unit conversion and successfully tested the utilisation of modules using different units. The unit conversion function uses the actual compartment volume to recalculate the amount of molecules by a given concentration. The state vector and all initial values were represented in molecule numbers, whereas the modules used still molar concentration and even different prefixes, e.g. milli- or micro-. The import of a unit conversion function has improved the work with different modules and enabled a facilitated realisation of the dilution of concentrations based on cell growth. On the other hand, it has increased the effort needed for the implementation of modules (the units must be set and pronounced additionally) and the difficult integration into the framework has lowered the computational efficacy drastically. The simulation time was increased by the factor of 4 to 10, depending on the number of species, nonetheless, after optimising the main framework, the integration of this function should again be considered.

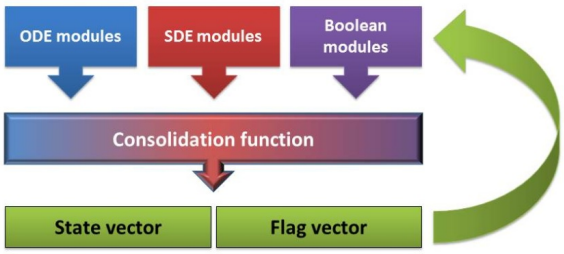


Figure 16: Simulation step

Illustration of a step in the simulation process of the whole cell model. The modules are simulated independently, afterwards the consolidation function merges the results of the modules and writes them to the state vector. Other entries, e.g. Boolean and state flags are controlled in the flag vector. The next simulation step is performed using the new entries from the state and flag vector as initial values or new cell state, respectively.

Integration of tools and analysis methods

The planned extensions and improvements of the modules and the framework for the next months concern especially a closer connection to *SBML*. The import of *SBML* models should be extended to a full support of this format as a general exchange format for the whole cell model. At the moment all information of a module could be stored in an *SBML* file and would allow for the use of external tools to analyse, simulate and modify the modules more easily. The connection to *semanticSBML* [40] tools and functions could also be facilitate. The annotation, parameter balancing and merging of *SBML* models could increase the number of available models very fast. At the moment only *SBtab* serves as an output format for the use of the tools from *semanticSBML*.

Another important extension is the implementation of more test methods to ensure a correctly working simulation environment. This could facilitate the modification of functions and methods in the framework considerably, since the test methods could guarantee the functionality of the simulation environment. Also further tools for visualisation of the modules and the whole model are needed to ensure the framework to be a powerful and productive environment for a whole cell model.

In the long run, a reliable tool for parameter estimation is inevitable [34]. As the model implementation of the metabolic model has shown, the parametrisation of a module is at the moment very difficult to perform reliably. Also experimental datasets will be needed to define a consistent cell state under given environmental conditions and create a model with physiological behaviour and predicting power.

Besides tools for facilitating the implementation of modules, the single modules as well as the whole model should be able to analyse. However, it is a long way to go before the implementation of analysis tools for sensitivity or steady state analysis [34] is beneficial. Most probably the architecture of the framework will have to be changed noticeably compared to the current first version.

Many tasks are still open concerning the successful implementation of a whole cell model or at least a powerful and reliable simulation software for this approach. Nonetheless, we were able to identify and effectively tackle some of the challenges that occurred during the last 6 months. We were able to create a working software environment for the simulation of modularised computational models including assisting tools for modification and visualisation of these modules. The software furthermore supports commonly used and accepted standards in the systems biology community. It can be extended and modified easily due to the use of a user-friendly and wide spread programming language. The presented implementation approaches demonstrated different possibilities to gain a metabolic model for the yeast *S. cerevisiae*. These examples showed the needs and possibilities of a module created for the use in this software.
In the next months the software as well as the model has to be optimised and adapted to gain a more reliable and precise working version of this whole cell modelling approach.

7. Acknowledgements

I would like to express my deep gratitude to Prof. Dr. Dr. h.c. Edda Klipp, my research supervisor, for their patient guidance, enthusiastic encouragement and useful critiques. I would also like to thank Dr. Marcus Krantz, for his advice and assistance. I like to thank Dr. Martin Seeger for the brilliant collaboration, his way to solve problems and the encouragement during the last months. My grateful thanks are also extended to Timo Lubitz for helping me with this work and the parameter balancing tool, to Katja Tummler, Claudia Beck, Max Schelker, and David Jesinghaus for helping me with the metabolism, to Ivo Maintz for preparing my computer, to Prof. Dr. Casten Hartmann and Iurii Kozhan from the FU Berlin for taking time for me and my questions.

I also like to thank Dr. Thomas Spießer, Dr. Clemens Kühn, Friedemann Uschner, Dr. Jannis Uhlendorf, Dr. Magdalena Rother, Katharina Albers, Dr. Marvin Schulz, Dr. Max Flöttmann, Dr. Gabriele Schreiber, Lotte Teufel, Aouefa Amoussouvi, Wolfgang Giese, Björn Goldenbogen, Ulrike Münzner, Matthias Reis, Stephan Adler, and Dr. Samuel Drulhe for willingly participating on the whole cell model workshop.

My thanks to Phillipp Schmidt and Falko Krause for helping me with the Python program and Prof. Dr. Hermann-Georg Holzhütter for agreeing to evaluate this work.

Finally, I wish to thank my parents, Jeanette Werner, and Aurora Rodriguez for their support and encouragement throughout my study.

References

[1] Hiroaki Kitano. Systems biology: A brief overview. *Science*, 295(5560):1662–1664, 2002.

[2] Uwe Theobald, Werner Mailinger, Michael Baltes, Manfred Rizzi, and Matthias Reuss. In vivo analysis of metabolic dynamics in saccharomyces cerevisiae: I. experimental observations. *Biotechnology and Bioengineering*, 55(2):305–316, 1997.

[3] J. M. Buescher, W. Liebermeister, M. Jules, M. Uhr, J. Muntel, and others. Global network reorganization during dynamic adaptations of bacillus subtilis metabolism. *Science*, 335(6072):1099–1103, 2012.

[4] Christoph Wittmann, Michael Hans, Wouter A. van Winden, et al. Dynamics of intracellular metabolites of glycolysis and TCA cycle during cell-cycle-related oscillation insaccharomyces cerevisiae. *Biotechnology and Bioengineering*, 89(7):839–847, 2005.

[5] Arkadi Manukyan, Lesley Abraham, Huzefa Dungrawala, and Brandt L. Schneider. Synchronization of yeast. In Gaspar Banfalvi, editor, *Cell Cycle Synchronization*, volume 761 of *Methods in Molecular Biology*, pages 173–200. Humana Press, Totowa and NJ, 2011.

[6] Kouichi Takahashi, Katsuyuki Yugi, Kenta Hashimoto, et al. Computational challenges in cell simulation: a software engineering approach. *IEEE Intelligent Systems*, 17(5):64–71, 2002.

[7] David Botstein and Gerald R. Fink. Yeast: An experimental organism for 21st century biology. *Genetics*, 189(3):695–704, 2011.

[8] D. Botstein. Genetics: Yeast as a model organism. *Science*, 277(5330):1259–1260, 1997.

[9] Christoph Wierling, Ralf Herwig, and Hans Lehrach. Resources, standards and tools for systems biology. *Briefings in functional genomics & proteomics*, 6(3):240–251, 2007.

[10] Nicolas Le Novère, Andrew Finney, Michael Hucka, et al. Minimum information requested in the annotation of biochemical models (MIRIAM). *Nature Biotechnology*, 23(12):1509–1515, 2005.

[11] Michael Hucka, Andrew Finney, Herbert M. Sauro, et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.

[12] A. A. Cuellar, C. M. Lloyd, P. F. Nielsen, D. P. Bullivant, D. P. Nickerson, and P. J. Hunter. An overview of cellml 1.1, a biological model description language. *SIMULATION*, 79(12):740–747, 2003.

[13] Masaru Tomita, Kenta Hashimoto, Kouichi Takahashi, et al. E-CELL: software environment for whole-cell simulation. *Bioinformatics*, 15(1):72–84, 1999.

[14] James C. Schaff and Leslie M. Loew, editors. *The virtual cell*, volume 4, 1999.

[15] Jonathan R. Karr, Jayodita C. Sanghvi, Derek N. Macklin, et al. A whole-cell computational model predicts phenotype from genotype. *Cell*, 150(2):389–401, 2012.

[16] MATLAB. *version 7.11.0 (R2010b)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[17] M. Tomita. Whole-cell simulation: a grand challenge of the 21st century. *Trends in biotechnology*, 19(6):205–210, 2001.

[18] James P. J. Hetherington, Ian David Lockhart Bogle, Peter Saffrey, et al. Addressing the challenges of multiscale model management in systems biology. *Computers & Chemical Engineering*, 31(8):962–979, 2007.

[19] Marco Antoniotti, Alberto Policriti, Nadia Ugel, and Bud Mishra. Model building and model checking for biochemical processes. *Cell Biochemistry and Biophysics*, 38:2003, 2003.

[20] Alfredo I. Hernández, Virginie Le Rolle, Antoine Defontaine, and Guy Carrault. A multiformalism and multiresolution modelling environment: application to the cardiovascular system and its regulation. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 367(1908):4923–4940, 2009.

[21] F. Kuhl, R. Weatherly, and J. Dahmann. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Prentice Hall, 2000.

[22] Python. http://www.python.org/.

[23] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[24] Benjamin J. Bornstein, Sarah M. Keating, Akiya Jouraku, and Michael Hucka. LibSBML: an API library for SBML. *Bioinformatics*, 24(6):880–881, 2008.

[25] Wolfram Liebermeister, Timo Lubitz, and Jens Hahn. SBtab specification. unpublished.

[26] Microsoft. Microsoft Excel. Redmond, Washington: Microsoft, 2010. Computer Software.

[27] A.J. Lotka. *Elements of Physical Biology*. Williams & Wilkins Company, 1925.

[28] Janna Hastings, Paula de Matos, Adriano Dekker, et al. The ChEBI reference database and ontology for biologically relevant chemistry: enhancements for 2013. *Nucleic Acids Research*, 41(D1):D456–D463, 2013.

[29] Michael Ashburner, Catherine A. Ball, Judith A. Blake, et al. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nature genetics*, 25(1):25–29, 2000.

[30] Finn Hynne, Sune Danø, and Preben Graae Sørensen. Full-scale model of glycolysis in saccharomyces cerevisiae. *Biophysical chemistry*, 94(1-2):121–163, 2001.

[31] W.W Cleland. The kinetics of enzyme-catalyzed reactions with two or more substrates or products. *Biochimica et Biophysica Acta (BBA) - Specialized Section on Enzymological Subjects*, 67:104–137, 1963.

[32] Christine Nazaret, Margit Heiske, Kevin Thurley, and Jean-Pierre Mazat. Mitochondrial energetic metabolism: A simplified model of TCA cycle with ATP production. *Journal of Theoretical Biology*, 258(3):455–464, 2009.

[33] Fan Wu, Feng Yang, Kalyan C. Vinnakota, and Daniel A. Beard. Computer modeling of mitochondrial tricarboxylic acid cycle, oxidative phosphorylation, metabolite transport, and electrophysiology. *Journal of Biological Chemistry*, 282(34):24525–24537, 2007.

[34] Edda Klipp, Wolfram Liebermeister, Christoph Wierling, et al. *Systems Biology - A Textbook*. Wiley-Blackwell, 2009.

[35] Natalie J. Stanford, Timo Lubitz, Kieran Smallbone, et al. Systematic construction of kinetic models from genome-scale metabolic networks. *PLoS ONE*, 8(11):e79195, 2013.

[36] Markus J. Herrgård, Neil Swainston, Paul Dobson, et al. A consensus yeast metabolic network reconstruction obtained from a community approach to systems biology. *Nature Biotechnology*, 26(10):1155–1160, 2008.

[37] Timo Lubitz, Marvin Schulz, Edda Klipp, and Wolfram Liebermeister. Parameter balancing in kinetic models of cell metabolism †. *The Journal of Physical Chemistry B*, 114(49):16298–16303, 2010.

[38] Wolfram Liebermeister, Jannis Uhlendorf, and Edda Klipp. Modular rate laws for enzymatic reactions: thermodynamics, elasticities and implementation. *Bioinformatics*, 26(12):1528–1534, 2010.

[39] Andrew Gelman. *Bayesian data analysis*, volume [60] of *Texts in statistical science series*. Chapman & Hall/CRC, Boca Raton and Fla. [u.a.], 2004.

[40] Falko Krause, Jannis Uhlendorf, Timo Lubitz, et al. Annotation and merging of SBML models with semanticSBML. *Bioinformatics*, 26(3):421–422, 2010.

[41] Stefan Hoops, Sven Sahle, Ralph Gauges, et al. COPASI–a COmplex PAthway SImulator. *Bioinformatics*, 22(24):3067–3074, 2006.

[42] Linda R. Petzold. Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. *SIAM Journal on Scientific and Statistical Computing*, 4(1):136–148, 1983.

[43] Karl Strehmel, Rüdiger Weiner, and Helmut Podhaisky. *Numerik gewöhnlicher Differentialgleichungen: Nichtsteife, steife und differential-algebraische Gleichungen*. Vieweg+Teubner Verlag, 2 edition, 2012.

[44] John C. Butcher. *Numerical Methods for Ordinary Differential Equations*. Wiley, J, 2003.

[45] Rainer Machné, Andrew Finney, Stefan Müller, et al. The SBML ODE Solver Library: a native api for symbolic and fast numerical analysis of reaction networks. *Bioinformatics*, 22(11):1406–1407, 2006.

[46] Hiromu Takizawa, Kazushige Nakamura, Akito Tabira, et al. LibSBMLSim: a reference implementation of fully functional sbml simulator. *Bioinformatics*, 29(11):1474–1476, 2013.

[47] ISO. The ANSI C standard (C99). Technical Report WG14 N1124, ISO/IEC, 1999.

[48] Radu Serban and Alan C Hindmarsh. Cvodes, the sensitivity-enabled ode solver in sundials. In *Proceedings of the 5th International Conference on Multibody Systems, Nonlinear Dynamics and Control, Long Beach, CA*, 2005.

[49] Alan C. Hindmarsh, Peter N. Brown, Keith E. Grant, et al. SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers. *ACM Transactions on Mathematical Software*, 31(3):363–396, 2005.

[50] ODEint (SciPy). http://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html.

[51] Assimulo 2.4. http://www.assimulo.org/.

[52] Travis Oliphant, Pearu Peterson, and Eric Jones. Scipy: Open source scientific tools for Python. *http://www.scipy.org/*, 2001.

[53] Alan C. Hindmarsh and Robert S. Stepleman. Odepack, a systematized collection of ode solvers. *IMACS Transactions on Scientific Computation*, 1:55–64, 1983.

[54] Scott D Cohen and Alan C Hindmarsh. Cvode, a stiff/nonstiff ode solver in c. *Computers in physics*, 10(2):138–143, 1996.

[55] Minoru Kanehisa, Susumu Goto, Yoko Sato, et al. KEGG for integration and interpretation of large-scale molecular data sets. *Nucleic Acids Research*, 40(D1):D109–D114, 2011.

[56] Donald Voet. *Biochemistry*. Wiley, Hoboken and NJ, 2011.

[57] E. Jiménez-Martí, A. Zuzuarregui, M. Gomar-Alba, D. Gutiérrez, C. Gil, and M. del Olmo. Molecular response of saccharomyces cerevisiae wine and laboratory strains to high sugar stress conditions. *International Journal of Food Microbiology*, 145(1):211–220, 2011.

[58] James Michael Cherry, Eurie L. Hong, Craig Amundsen, et al. Saccharomyces Genome Database: the genomics resource of budding yeast. *Nucleic Acids Research*, 40(D1):D700–D705, 2012.

[59] Fernando Rodrigues, Paula Ludovico, and Cecília Leão. Sugar metabolism in yeasts: an overview of aerobic and anaerobic glucose catabolism. In Gábor Péter and Carlos Rosa, editors, *Biodiversity and Ecophysiology of Yeasts*, The Yeast Handbook, pages 101–121. Springer-Verlag, Berlin/Heidelberg, 2006.

[60] Hnin W. Aung, Susan A. Henry, and Larry P. Walker. Revising the representation of fatty acid, glycerolipid, and glycerophospholipid metabolism in the consensus model of yeast metabolism. *Industrial Biotechnology*, 9(4):215–228, 2013.

[61] Vijayalakshmi Chelliah, Camille Laibe, and Nicolas Le Novère. Biomodels database: A repository of mathematical models of biological processes. *Methods Mol Biol*, 1021:189–199, 2013.

[62] Avi Flamholz, Elad Noor, Arren Bar-Even, and Ron Milo. equilibrator - the biochemical thermodynamics calculator. *Nucleic Acids Research*, 40(D1):D770–D775, 2012.

[63] Charles William Gear and D. R. Wells. Multirate linear multistep methods. *BIT*, 24(4):484–502, 1984.

[64] Jan Frederick Andrus. Numerical solution of systems of ordinary differential equations separated into subsystems. *SIAM Journal on Numerical Analysis*, 16(4):605–611, 1979.

[65] Simon Portegies Zwart, Steve McMillan, Stefan Harfst, et al. A multiphysics and multiscale software environment for modeling astrophysical systems. *New Astronomy*, 14(4):369–378, 2009.

List of Figures

1. A modular approach . . . 5
2. Import and connection of modules . . . 14
3. One simulation step . . . 17
4. Example of the modularisation of a simple mass action system . . . 18
5. Time courses of the simulated test models . . . 19
6. Framework test using *Lotka-Volterra* . . . 22
7. Framework test using mass action kinetics . . . 24
8. Simulation scheme of a module . . . 26
9. Simulation tests using the model by Hynne *et al.* from 2001 [30] . . . 28
10. Scheme of the model . . . 30
11. Scheme of the central metabolism in yeast . . . 31
12. Scheme of the glycolysis module . . . 36
13. Scheme of the TCA cycle module . . . 37
14. Scheme of the transport module . . . 38
15. Visualised error . . . 41
16. Simulation step . . . 46
17. Reaction network taken from Hynne *et al.* 2001 . . . 63
18. Reaction network taken from Nazaret *et al.* 2009 . . . 64
19. Reaction network taken from Wu *et al.* 2007 . . . 65
20. Transport module by Matthias Reis . . . 66
21. Example for the module representation in a Python script . . . 66
22. Test output - matching species . . . 67

List of Tables

1. Separation tests glycolysis model from Hynne *et al.* . . . 27
2. Reactions in the glycolysis module . . . 57
3. Reactions in the TCA cycle module . . . 58
4. Reactions in the transport module . . . 59
5. Species in the glycolysis module . . . 60
6. Species in the transport module . . . 61
7. Species in the mitochondrial module . . . 62

A. Appendix

Table 2: Reactions in the glycolysis module

Reaction	Formula
Glycolysis	
Adenylate kinase (cyt)	ATP + AMP $\rightleftharpoons$ 2 ADP
Energy consumption	ATP $\rightarrow$ ADP + P_i
Fructose-bisphosphate aldolase	FBP $\rightleftharpoons$ DHAP + GAP
Glucose transporter	GLC_{ext} $\rightleftharpoons$ GLC
Glucose-6-phosphate isomerase	G6P $\rightleftharpoons$ F6P
Glyceraldehyde-3-phosphate dehydrogenase	GAP + NAD $\rightleftharpoons$ BPG + NADH + H
Glycerol production (lumped)	DHAP + NADH + H $\rightarrow$ Glyc + NAD
Glycerol transport	Glyc + H $\rightleftharpoons$ $Glyc_{ext}$ + H_{ext}
Glycogen production (storage/lumped)	G6P + ATP $\rightarrow$ Glyco + ADP
Hexokinase (D-glucose)	GLC + ATP $\rightarrow$ G6P + ADP + H
Phosphoenolpyruvate production (lumped)	BPG + ADP $\rightleftharpoons$ PEP + ATP
Phosphofructokinase	F6P + ATP $\rightarrow$ FBP + ADP
Pyruvate kinase	PEP + ADP $\rightarrow$ PYR + ATP
Triosephosphate isomerase	GAP $\rightleftharpoons$ DHAP

Cytosolic species without index

Table 3: Reactions in the TCA cycle module

Reaction	Formula
Tricarboxylic acid cycle	
AKG DH (dihydroliponamide)	$SDL_{mit} \rightleftharpoons sucCoA_{mit}$
AKG DH (liponamide)	$AKG_{mit} + H_{mit} \rightleftharpoons SDL_{mit}$
Cis-aconitate(3-) to isocitrate	$cisAcon_{mit} \rightleftharpoons isoCIT_{mit}$
Citrate synthase	$AcCoA_{mit} + OAA_{mit} \rightleftharpoons CIT_{mit}$
Citrate to cis-aconitate(3-)	$CIT_{mit} \rightleftharpoons cisAcon_{mit}$
Fumarase	$FUM_{mit} \rightleftharpoons MAL_{mit}$
Isocitrate DH	$isoCIT_{mit} + NAD_{mit} \rightleftharpoons AKG_{mit} + CO2_{mit} + NADH_{mit} + H_{mit}$
Malate DH	$MAL_{mit} + NAD_{mit} \rightleftharpoons H_{mit} + NADH_{mit} + OAA_{mit}$
Pyruvate DH	$NAD_{mit} + PYR_{mit} \rightleftharpoons AcCoA_{mit} + CO2_{mit} + NADH_{mit}$
Succinate DH	$SUC_{mit} + Q_{mit} \rightleftharpoons FUM_{mit} + QH2_{mit}$
Succinyl-CoA ligase	$ADP_{mit} + P_{i,mit} + sucCoA_{mit} \rightleftharpoons ATP_{mit} + SUC_{mit}$
Additional reactions	
AcetylCoA synthetase (mit)	$AcCoA_{mit} + AMP_{mit} + 2\ P_{i,mit} \rightleftharpoons ACE_{mit} + ATP_{mit}$
Acetaldehyde DH (mit)	$ACA_{mit} + NAD_{mit} \rightleftharpoons ACE_{mit} + NADH_{mit} + H_{mit}$
Adenylate kinase (mit)	$ATP_{mit} + AMP_{mit} \rightleftharpoons 2\ ADP_{mit}$
Alcohol DH (mit)	$ACA_{mit} + NADH_{mit} + H_{mit} \rightleftharpoons etOH_{mit} + NAD_{mit}$
Carnitine O-acetyltransferase (mit)	$AcCar_{mit} \rightleftharpoons AcCoA_{mit}$
Malic enyzme	$MAL_{mit} + NAD_{mit} \rightleftharpoons CO2_{mit} + NADH_{mit} + PYR_{mit}$

Cytosolic species without index

Table 4: Reactions in the transport module

Reaction	Formula
Mitochondrial transport reactions	
Acetaldehyde transport (mit)	$ACA \rightleftharpoons ACA_{mit}$
ADP/ATP transporter	$ATP + ADP_{mit} \rightleftharpoons ATP_{mit} + ADP$
Carnitine O-acetyltransferase (cyt)	$AcCoA \rightleftharpoons AcCar$
CO2 diffusion (mit)	$CO2 \rightleftharpoons CO2_{mit}$
EtOH diffusion (mit)	$etOH \rightleftharpoons etOH_{mit}$
H^+ diffusion (mit)	$H \rightleftharpoons H_{mit}$
Malate transport	$MAL + P_i \rightleftharpoons MAL_{mit} + P_{i,mit}$
NAD transporter	$AMP_{mit} + NAD \rightleftharpoons AMP + NAD_{mit}$
O-acetylcarnitine transport	$AcCar \rightleftharpoons AcCar_{mit}$
Oxaloacetate transporter	$OAA + H \rightleftharpoons OAA_{mit} + H_{mit}$
Phosphate transporter	$P_i + H \rightleftharpoons P_{i,mit} + H_{mit}$
Pyruvate transporter (mit)	$PYR + H \rightleftharpoons PYR_{mit} + H_{mit}$
Cytosolic transport reactions	
Acetaldehyde transport (cyt)	$ACA \rightleftharpoons ACA_{ext}$
Acetate transport (cyt)	$ACE \rightleftharpoons ACE_{ext}$
CO2 diffusion (cyt)	$CO2 \rightleftharpoons CO2_{ext}$
EtOH diffusion (cyt)	$etOH \rightleftharpoons etOH_{ext}$
H^+ diffusion (cyt)	$H \rightleftharpoons H_{ext}$
Pyruvate transporter (cyt)	$PYR + H \rightleftharpoons PYR_{ext} + H_{ext}$
Oxidative phosphorylation	
Complex III	$QH2_{mit} + 2\ H_{mit} \rightarrow Q_{mit} + 4\ H_{mit}$
Complex IV	$8\ H_{mit} \rightarrow 4\ H$
Complex V	$ADP_{mit} + P_{i,mit} + 4\ H \rightleftharpoons ATP_{mit} + 4\ H_{mit}$
NADH DH (outer)	$H + NADH + Q_{mit} \rightarrow NAD + QH2_{mit}$
NADH DH (inner)	$H_{mit} + NADH_{mit} + Q_{mit} \rightarrow NAD_{mit} + QH2_{mit}$
Additional reactions	
Acetaldehyde DH (cyt)	$ACA + NAD \rightleftharpoons ACE + NADH + H$
Acetyl-CoA synthetase (cyt)	$AcCoA + AMP + 2\ P_i \rightleftharpoons ACE + ATP$
Alcohol DH (cyt)	$ACA + NADH + H \rightleftharpoons etOH + NAD$
Pyruvate carboxylase	$ATP + PYR \rightleftharpoons ADP + P_i + OAA$
Pyruvate decarboxylation	$PYR + H \rightleftharpoons ACA + CO2$

Cytosolic species without index

Table 5: Species in the glycolysis module

Abbreviation	Name	Comp.	ChEBI
ADP	adenosine 5'-triphosphate	cyt	30616
AMP	adenosine 5'-monophosphate	cyt	16027
ATP	adenosine 5'-triphosphate	cyt	30616
BPG	1,3-bisphosphoglycerate	cyt	22902[2]
DHAP	dihydroxyacetone phosphate	cyt	57642
FBP	fructose 1,6-bisphosphate	cyt	40595
F6P	fructose 6-phosphate	cyt	15946
GLC	D-glucose	cyt	4167
GLC_{ext}	external D-glucose	cyt	4167
G6P	glucose 6-phosphate	cyt	4170
GAP	glyceraldehyde 3-phosphate	cyt	17138
Glyc	glycerol	cyt	17754
$Glyc_{ext}$	external glycerol	cyt	17754
Glyco	glycogen	cyt	28087
H	hydron	cyt	15378
H_{ext}	external hydron	cyt	15378
NAD^+	nicotinamide adenine dinucleotide (ox.)	cyt	15846
NADH	nicotinamide adenine dinucleotide (red.)	cyt	16908
PEP	phosphoenolpyruvate	cyt	44897
P_i	inorganic phosphate	cyt	18367
PYR	pyruvate	cyt	15361

[1] ChEBI of bisphosphoglyceric acid

Table 6: Species in the transport module

Abbreviation	Name	Comp.	ChEBI
ACA_{mit}	acetaldehyde	mit	15343
ACE_{mit}	acetate	mit	30089
$AcCar_{mit}$	O-acetyl-L-carnitine	mit	57589
$AcCoA_{mit}$	acetyl-CoA	mit	15351
ADP_{mit}	adenosine 5'-triphosphate	mit	30616
ATP_{mit}	adenosine 5'-triphosphate	mit	30616
$cisAcon_{mit}$	cis-aconitate	mit	32805
CIT_{mit}	citrate	mit	16947
$CO2_{mit}$	carbon dioxide	mit	16526
$etOH_{mit}$	ethanol	mit	16236
FUM_{mit}	fumarate	mit	18012
H_{mit}	hydron	mit	15378
$isoCIT_{mit}$	isocitrate	mit	30887
MAL_{mit}	malate	mit	15595
NAD^+	nicotinamide adenine dinucleotide (ox.)	mit	15846
$NADH_{mit}$	nicotinamide adenine dinucleotide (red.)	mit	16908
OAA_{mit}	oxaloacetate	mit	30744
$P_{i,mit}$	inorganic phosphate	mit	18367
PYR_{mit}	pyruvate	mit	15361
Q_{mit}	ubiquinone	mit	16389
$QH2_{mit}$	ubiquinol	mit	17976
SUC_{mit}	succinate	mit	15741
$SucCoA_{mit}$	succinyl-CoA	mit	15380
$SDHL_{mit}$	S-succinyl-dihydrolipoamide	mit	439425[2]
AKG_{mit}	α-ketoglutarate[3]	mit	16810

[2] PubChem instead of ChEBI

[3] also 2-oxoglutarate

Table 7: Species in the mitochondrial module

Abbreviation	Name	Comp.	ChEBI
ACA_{cyt}	acetaldehyde	cyt	15343
ACA_{ext}	acetaldehyde	ext	15343
ACA_{mit}	acetaldehyde	mit	15343
ACE_{cyt}	acetate	cyt	30089
ACE_{ext}	acetate	ext	30089
ACE_{mit}	acetate	mit	30089
$AcCar_{cyt}$	O-acetyl-L-carnitine	cyt	57589
$AcCar_{mit}$	O-acetyl-L-carnitine	mit	57589
$AcCoA_{cyt}$	acetyl-CoA	cyt	15351
$AcCoA_{mit}$	acetyl-CoA	mit	15351
ADP	adenosine 5'-triphosphate	cyt	30616
ADP_{mit}	adenosine 5'-triphosphate	mit	30616
AMP	adenosine 5'-monophosphate	cyt	16027
AMP_{mit}	adenosine 5'-monophosphate	mit	16027
ATP	adenosine 5'-triphosphate	cyt	30616
ATP_{mit}	adenosine 5'-triphosphate	mit	30616
$CO2_{cyt}$	carbon dioxide	cyt	16526
$CO2_{ext}$	carbon dioxide	ext	16526
$CO2_{mit}$	carbon dioxide	mit	16526
$etOH_{cyt}$	ethanol	cyt	16236
$etOH_{ext}$	ethanol	ext	16236
$etOH_{mit}$	ethanol	mit	16236
H	hydron	cyt	15378
H_{ext}	external hydron	cyt	15378
H_{mit}	hydron	mit	15378
MAL_{cyt}	malate	cyt	15595
MAL_{mit}	malate	mit	15595
NAD^+	nicotinamide adenine dinucleotide (ox.)	cyt	15846
NAD^+	nicotinamide adenine dinucleotide (ox.)	mit	15846
NADH	nicotinamide adenine dinucleotide (red.)	cyt	16908
$NADH_{mit}$	nicotinamide adenine dinucleotide (red.)	mit	16908
OAA_{cyt}	oxaloacetate	cyt	30744
OAA_{mit}	oxaloacetate	mit	30744
P_i	inorganic phosphate	cyt	18367
$P_{i,mit}$	inorganic phosphate	mit	18367
PYR_{cyt}	pyruvate	cyt	15361
PYR_{ext}	pyruvate	ext	15361
PYR_{mit}	pyruvate	mit	15361

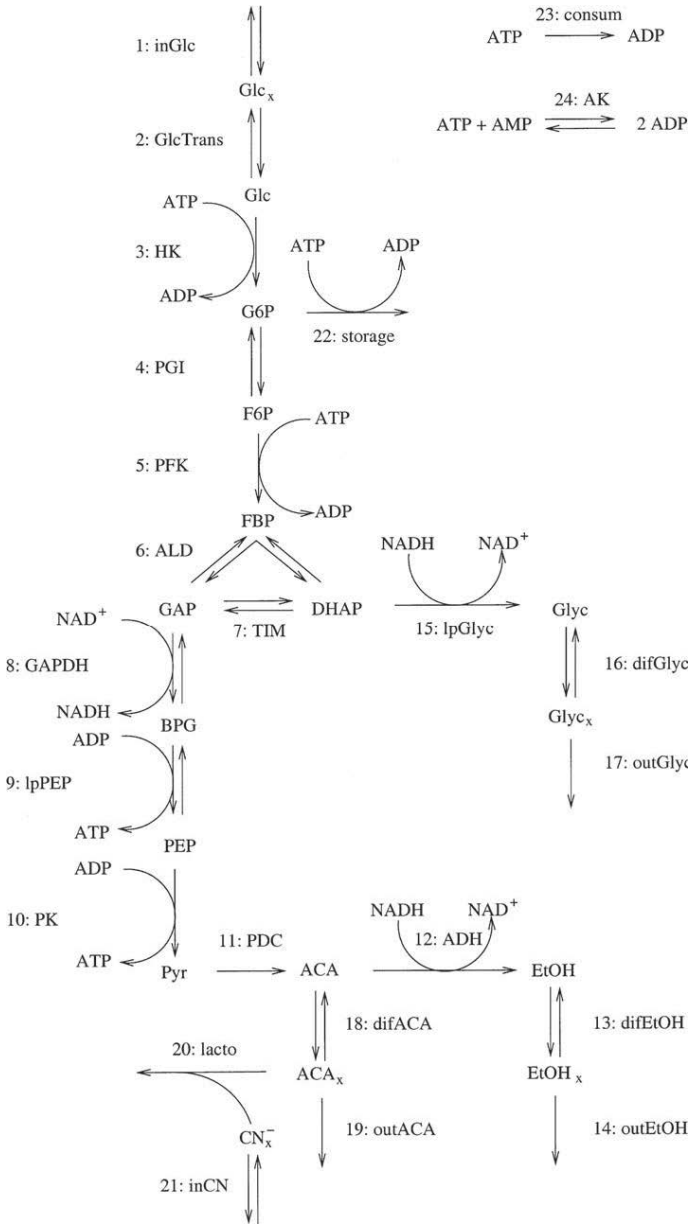


Figure 17: Reaction network taken from Hynne *et al.* 2001

The figure shows the reaction network of the computational model taken from Hynne *et al.* 2001, [30]. The model was used as a module representing the glycolysis for the implementation of the central carbon metabolism model and as a test module for the simulation environment.

Abbreviations:

Enzymes: ADH: alcohol dehydrogenase, AK: adenylate kinase, ALD: aldolase, ENO: enolase, G3PDH: glycerol 3-phosphate dehydrogenase, GAPDH: glyceraldehyde 3-phosphate dehydrogenase, HK: hexokinase, PDC: pyruvate carboxylase, PFK: phosphofructokinase-1, PGI: phosphoglucoisomerase, PGK: phosphoglycerate kinase, PGM: phophoglycerate mutase, PK: pyruvate kinase, TIM: triosephosphate isomerase. **Metabolites:** ACA: intracellular acetaldehyde, ACA_x: extracellular acetaldehyde, DHAP: dihydroxyacetone phosphate, BPG: 1,3-bisphosphoglycerate, EtOH: intracellular ethanol, $EtOH_x$: extracellular ethanol, F6P: fructose 6-phosphate, FBP: fructose 1,6-phosphate, G6P: glucose 6-phosphate, GAP: glyceraldehyde 3-phosphate, Glc: intracellular glucose, Glc_x: extracellular glucose, Glyc: intracellular glycerol, $Glyc_x$: extracellular glycerol, PEP: phosphoenol pyruvate, Pyr: pyruvate.

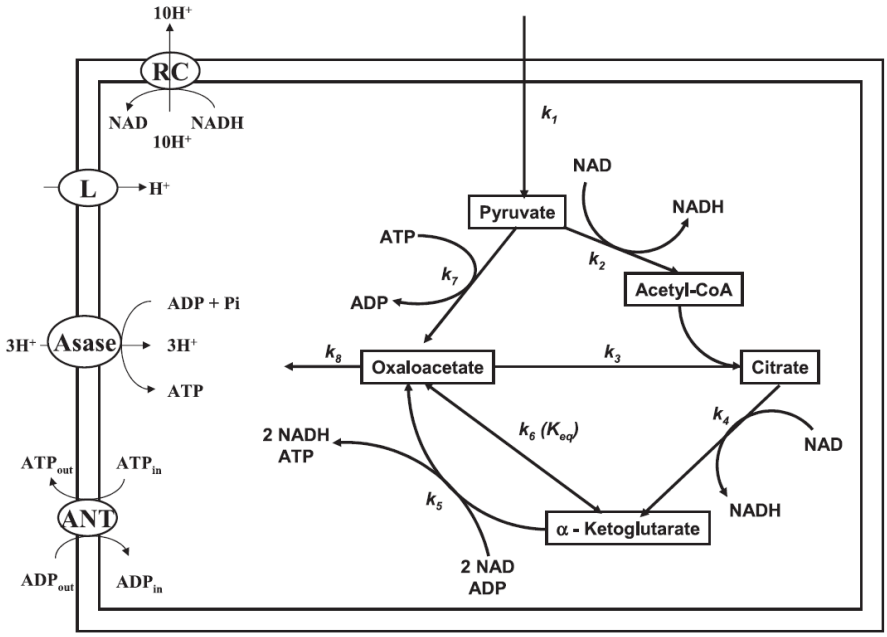


Figure 18: Reaction network taken from Nazaret *et al.* 2009

The figure shows the reaction network of the computational model taken from Nazaret *et al.* 2009, [32]. The model was utilised as a module representing the TCA cycle for the central carbon metabolism model.

Abbreviations:

ANT: adenine nucleotide translocase, Asase: ATP synthase, L: proton leak, RC: respiratory chain

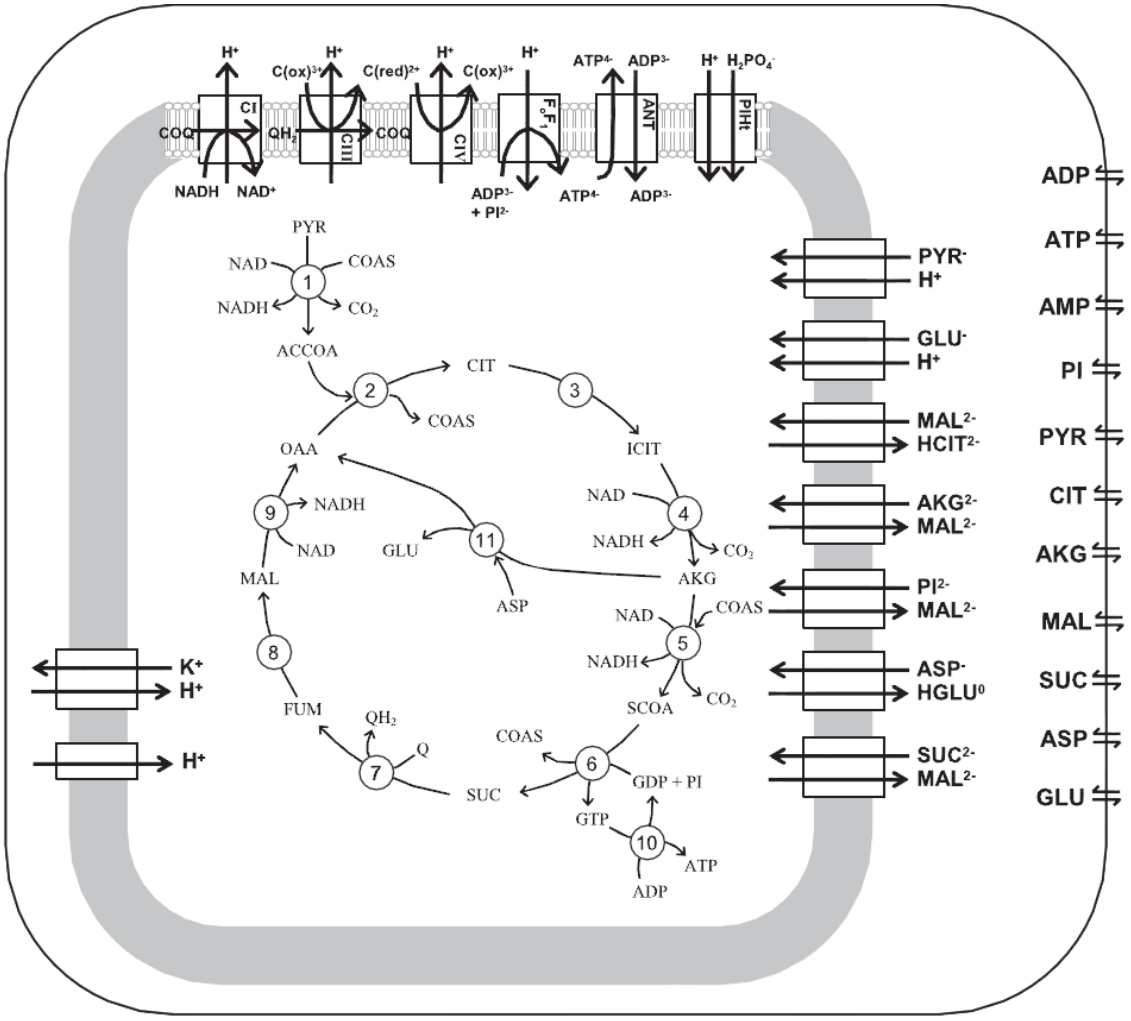


Figure 19: Reaction network taken from Wu *et al.* 2007

The figure shows the reaction network of the computational model taken from Wu *et al.* 2007, [33]. The model was used as a module representing the TCA cycle for the central carbon metabolism.

Reactions:

1: pyruvate dehydrogenase, 2: citrate synthase, 3: aconitase, 4: isocitrate dehydrogenase, 5: α-ketoglutarate dehydrogenase, 6: succinyl-CoA synthetase, 7: succinate dehydrogenase, 8: fumarase, 9: malate dehydrogenase, 10: nucleoside diphosphokinase, 11: glutamate oxaloacetate transaminase.

Abbreviations:

ACCOA: acetyl-CoA, AKG: α-ketoglutarate, ANT: adenine nucleotide translocase, ASP: asparagine acid, C(ox): oxidised cytochrome C, C(red): reduced cytochrome C, CIT: citrate, COAS: CoA-SH, COQ: oxidised ubiquinol, FUM: fumarate, GLU: glutamate, ICIT: isocitrate, MAL: malate, OAA: oxaloacetate, PI: inorganic phosphate, PYR: pyruvate, QH_2: reduced ubiquinol, SCOA: succinyl-CoA, SUC: succinate

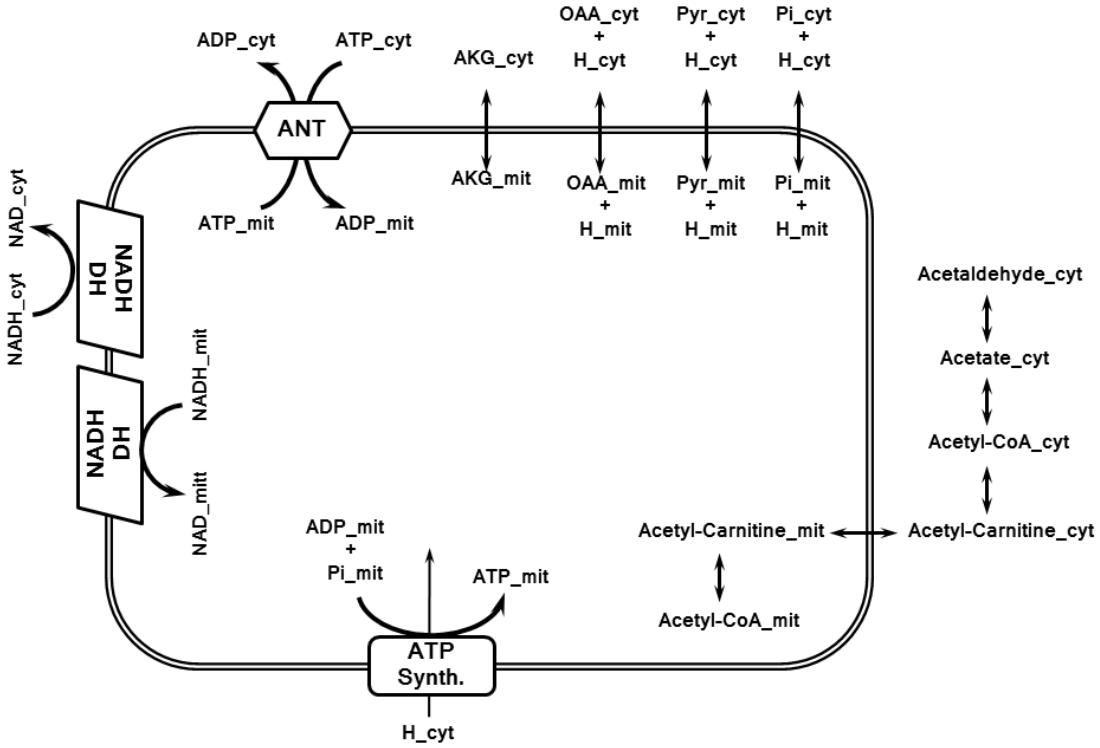


Figure 20: Transport module by Matthias Reis

The figure shows the reaction network of the transport module implemented by Matthias Reis. The module was used to connect the glycolysis and the TCA cycle modules in the central carbon metabolism model. **Abbreviations:**

AKG: α-ketoglutarate, ANT: adenine nucleotide translocase, OAA: oxaloacetate, Pi: inorganic phosphate, PYR: pyruvate

```
module = {}
module['name'] = 'Lotka_dict'
module['vars'] = ['x', 'y']
module['initvars'] = {'x': 10.0, 'y': 5.0}
module['pars'] = ['k1', 'k2', 'k3', 'k4']
module['initpars'] = {'k1': 1.0, 'k2': 0.1, 'k3': 0.02, 'k4': 1.0}
module['rates'] = {'v1': '(k1*x)', 'v2': '(k2*x*y)', 'v3': '(k3*x*y)', 'v4': '(k4*y)'}
module['odes'] = {'x': module['rates']['v1'] - module['rates']['v2'],
   'y': module['rates']['v3'] - module['rates']['v4']}
module['sp_annotations'] = {'x': 'CHEBI:16761', 'y': 'CHEBI:15422'}
module['sp_compartment'] = {'x': 'cytosol', 'y': 'cytosol'}
module['com_annotations'] = {'cytosol': 'GO:0005829'}
```

Figure 21: Example for the module representation in a Python script

The figure shows the representation of the *Lotka-Volterra* model [27] in a Python script. The model representation is based on the Python dictionary data structure and holds all information necessary for simulation.

annotation	compartment	tca_nazaret	glycolysis_hynne	trpMito_reis
CHEBI:15343	GO:0005576	.	ACAX	.
CHEBI:15343	GO:0005829	.	ACA	Acetaldehyde_cyt
CHEBI:15351	GO:0005759	facCoA	.	.
CHEBI:15351	GO:0005829	.	.	Acetyl_CoA_cyt
CHEBI:15378	GO:0005759	fh	.	H_0_mit
CHEBI:15378	GO:0005829	.	.	H_0_cyt
CHEBI:15422	GO:0005759	fatp	.	ATP_mit
CHEBI:15422	GO:0005829	.	ATP	ATP_cyt
CHEBI:15846	GO:0005759	fnad	.	NAD_0_mit
CHEBI:15846	GO:0005829	.	NAD	NAD_0_cyt
CHEBI:15954	GO:0005829	.	G6P	.
CHEBI:15960	GO:0005759	faccar	.	Acetyl_carnitine_mit
CHEBI:15960	GO:0005829	.	.	Acetyl_carnitine_cyt
CHEBI:16001	GO:0005829	.	BPG	.
CHEBI:16027	GO:0005829	.	AMP	.
CHEBI:16108	GO:0005829	.	DHAP	.
CHEBI:16236	GO:0005576	.	EtOHX	.
CHEBI:16236	GO:0005829	.	EtOH	.
CHEBI:16761	GO:0005759	fadp	.	ADP_mit
CHEBI:16761	GO:0005829	.	ADP	ADP_cyt
CHEBI:16905	GO:0005829	.	FBP	.
CHEBI:16908	GO:0005759	fnadh	.	NADH_0_mit
CHEBI:16908	GO:0005829	.	NADH	NADH_0_cyt
CHEBI:16947	GO:0005759	fcit	.	.
CHEBI:17234	GO:0005576	.	GlcX	.
CHEBI:17234	GO:0005829	.	Glc	.
CHEBI:17754	GO:0005576	.	GlycX	.
CHEBI:17754	GO:0005829	.	Glyc	.
CHEBI:18021	GO:0005829	.	PEP	.
CHEBI:18367	GO:0005759	.	.	Pi_0_mit
CHEBI:18367	GO:0005829	.	P	Pi_0_cyt
CHEBI:20935	GO:0005829	.	F6P	.
CHEBI:29052	GO:0005829	.	GAP	.
CHEBI:30089	GO:0005829	.	.	Acetate_cyt
CHEBI:30744	GO:0005759	foaa	.	Oxaloacetate_mit
CHEBI:30744	GO:0005829	.	.	Oxaloacetate_cyt
CHEBI:30915	GO:0005759	fkg	.	TwoOxoglutarate_mit
CHEBI:30915	GO:0005829	.	.	TwoOxoglutarate_cyt
CHEBI:32816	GO:0005759	fpyr	.	Pyr_mit
CHEBI:32816	GO:0005829	.	Pyr	Pyr_cyt

Figure 22: Test output - matching species

The figure shows the test output of a function written by Dr. Martin Seeger. The output shows the matching of species from 3 different modules, representing the glycolysis, the TCA cycle, and the transport reactions between the 2 pathways.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsgbehörde vorgelegen.

(Ort, Datum) (Unterschrift)