UMinho | 2014

Hugo Miguel Melo Giesteira

Development of an integrated platform for the in silico phenotype simulation of microbial strains

Universidade do Minho
Escola de Engenharia

Hugo Miguel Melo Giesteira

Development of an integrated platform for the in silico phenotype simulation of microbial strains

Outubro de 2014

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Hugo Miguel Melo Giesteira

Development of an integrated platform for the in silico phenotype simulation of microbial strains

Dissertação de Mestrado
Mestrado em Bioinformática

Trabalho realizado sob orientação de

Miguel Francisco Almeida Pereira Rocha
Carlos Daniel Machado

Outubro de 2014

Acknowledgements/Agradecimentos

Este trabalho pode ter sido realizado por mim mas tal não seria possível sem um grande grupo de pessoas que me apoio durante este ano. Por essa razão guardo a primeira seção do meu trabalho para agradecer o apoio a todas estas pessoas e dizer que sem elas não teria conseguido chegar até aqui.

Gostaria de agradecer aos meus orientadores, professor Miguel Rocha e doutor Daniel Machado por todos os conselhos e disponibilidade demonstrada desde o início do trabalho. Um especial obrigado ao Paulo Vilaça por todas as horas dispensadas em me orientar, ensinar e apoiar quando situações mais complicadas surgiam e por ter acreditado em mim até ao fim.

Aos meus colegas de mestrado, pelo companheirismo e amizade que vou guardar mesmo depois da final do curso. Aos meus colegas e amigos da SilicoLife, Marco, Hugo, Rafa, Pedro, Barbara, Maia, Simão, Sónia e Joel, por todo o apoio e por conseguirem transformar um local de trabalho num local mais agradável e cheia de boa disposição.

Um grande obrigado a todos os meus restantes amigos que, apesar de não terem percorrido o mesmo caminho académico que eu, sempre me deram conselhos, ajudaram a superar obstáculos muitas vezes colocados por mim mesmo e sempre tiveram palavras amigas que me fizeram puxar para cima e não me deixar desistir dos meus objetivos.

Por fim um especial agradecimento para a minha família, principalmente aos meus pais e à minha irmã, por todas as vezes que lidaram comigo nos momentos menos bons e pela paciência e dedicação pois devo a eles grande parte da pessoa que sou hoje.

A todas estas pessoas digo um enorme obrigado, sem vocês não estaria onde estou nem teria conquistado o que consegui até hoje. Obrigado!

Resumo

Recentes técnicas de sequenciação genómica e abordagens estão a aumentar constantemente, a uma taxa exponencial, os dados biológicos disponíveis. Esta informação está se tornar ainda mais acessível com o desenvolvimento de novas tecnologias de *high-throughout*, fazendo com que a simulação *in silico* de sistemas vivos ou sintéticos se torne mais atrativa. A crescente investigação dos últimos dez anos permitiu o desenvolvimento de modelos matemáticos sofisticados que, por meio de engenharia metabólica, são utilizados numa tentativa de otimizar as funções do organismo, modificando-os geneticamente para produzir compostos de interesse industrial.

Em relação ao uso dos modelos de simulação de fenótipo, métodos como o *Flux Balance Analysis* são utilizados para identificar conjuntos de manipulações genéticas que resultam em estirpes mutantes capazes de produzir compostos desejados. Variações desses métodos têm rapidamente surgido e, em paralelo, um grande número de ferramentas computacionais surgiram com a capacidade de realizar esses métodos. Todas estas ferramentas estão disponíveis para a comunidade e construídas em diferentes sistemas e linguagens. No entanto, nenhuma inclui todos os métodos relevantes, o que resulta numa necessidade de recurso a mais do que uma ferramenta para realizar certas tarefas.

Neste trabalho, uma plataforma que pode integrar todos estes métodos é apresentada com o objetivo de centralizar e integrar os métodos de simulação de fenótipo mais relevantes e também permitir a sua extensão fácil com outros métodos. Para este trabalho, foi realizado um estudo dos métodos e ferramentas mais relevantes de modo a que esta plataforma possa integrar os métodos e ferramentas mais significativas. Esta plataforma é dividida em duas camadas: uma camada de ligação que é responsável pela troca de informação através das ferramentas computacionais e línguas, e uma camada de formulação responsável por executar os métodos dessas ferramentas e também apresentar os seus resultados.

Através destas camadas, a plataforma pode executar métodos de qualquer ferramenta disponível construída em qualquer linguagem computacional e fornecer aos investigadores o acesso a todos os métodos em uma única plataforma. Como aplicação prática desta plataforma, foi desenvolvido um *plugin* e integrado no OptFlux, um *software* de código aberto para apoiar as tarefas de engenharia metabólica, e encontra-se disponível em www.optflux.org. Este *plugin* oferece uma ligação com a ferramenta COBRA e executa os métodos de simulação metabólicas mais relevantes presentes no último.

Abstract

Modern sequencing techniques and omics approaches are constantly increasing available biological data at an exponential rate. This information is becoming even more accessible with the development of new high-throughout technologies, making *in silico* simulation of living and synthetic systems more attractive. The growing research in the past decade allowed the development of sophisticated mathematical models that, through Metabolic Engineering, are used in an attempt to optimize organism's functions, genetically modifying them to produce compounds of industrial interest.

Regarding the use of models for phenotype simulation, methods such as Flux Balance Analysis are used to identify sets of genetic manipulations that result in mutant strains capable of producing desired compounds. Variations of these methods have rapidly appeared and in parallel, a great number of computational tools emerged with the capability of performing these methods. All these tools are available for the community and built in different systems and languages. However, none includes all the relevant methods, which results in a need of recurring to more than one tool to accomplish certain tasks.

In this work, a platform that can integrate all these methods is presented with the goal of centralizing and integrating the most relevant existing phenotype simulation methods and also enabling their easy extension with other methods. For this work, a study of the most relevant methods and tools was made so that this platform could integrate the most significant methods and tools. This platform is divided in two layers: a connection layer that is responsible for exchanging information through the computational tools and languages, and a formulation layer responsible for performing the methods from these tools and also presenting the results.

Through these layers, the platform can perform methods from any available tool built in any computational language and provide to the researchers the access to all methods in a single platform. As a practical implementation of this platform, a plugin was developed and integrated in OptFlux, an open-source software to support metabolic engineering tasks, available at www.optflux.org. This plugin provides a connection with the COBRA Toolbox and performs the most relevant metabolic simulation methods present in the latter.

Contents

List of Figures

List of Tables

Acronyms

CLP	Coin-or Linear Programming
CPLEX	ILOG CPLEX Optimization Studio
FBA	Flux Balance Analysis
GLPK	GNU Linear Programming Kit
GSSMs	Genome Scale Metabolic Models
GUI	Graphical User Interface
LP	Linear Programming
ME	Metabolic Engineering
MILP	Mixed Integer Linear Programming
MOMA	Minimization of Metabolic Adjustment
MVC	Model-view-controller
QP	Quadratic Programming
ROOM	Regulatory On-Off Minimization
SBML	Systems Biology Markup Language

Chapter 1

Introduction

1.1. Context and motivation

The field of Metabolic Engineering (ME) studies the optimization of living organisms' metabolic processes, by genetically modifying them to achieve a certain goal, such as increasing production of a desired compound or allowing production of a new product, playing an important role in the biotechnology industry [1]. Over these last few years, computational tools have been used as a way to improve ME methods by computational screening of strains and strain optimization based on computer simulations.

Indeed, in the past decades, biological data increased at an exponential rate, information became even more accessible with the development of new technologies, making *in silico* simulation of living and synthetic systems more attractive [2]. The increase of research in the past years allowed the development of sophisticated mathematical models capable of predicting cellular behavior.

Modelling of cells has been traditionally achieved through the use of dynamic models. However, since these require information typically difficult to determine, their applicability is limited to small-scale systems. As an alternative, recent efforts allowed the development of genome-scale metabolic models (GSMMs) for several organisms, taking advantage on the growing number of sequenced genomes and the availability of Bioinformatics tools for automatic genome annotation [3]. These models allowed the study of the metabolism of different organisms through several methods for phenotype simulation and structural analysis.

Regarding the use of GSMMs for phenotype simulation, algorithms such as Flux Balance Analysis (FBA) [4] or Minimization of Metabolic Adjustment (MOMA) [5] were proposed and quickly became popular. In recent years, the number of methods proposed for this task has been growing rapidly, including variants of the objective function, imposed constraints, optimization methods or inputs (for instance, including different types of omics data).

Also, in parallel, a number of computational tools and integrated platforms have been proposed for ME tasks. Notable examples are the COBRA toolbox [6], which works over MATLAB (a more recent version in Python has also been proposed) or CellNetAnalyser [7], just to name a few. Within the host group for this thesis, the OptFlux ME platform has also been developed over the past few years [8] aiming to be a reference tool in the field.

While all these tools are available for the community, none includes all the relevant methods, impairing their straightforward comparison for a given task. In many cases, methods are proposed in a publication being tested with few case studies, while their thorough validation is hard to achieve and typically not done.

1.2. Goals

The main goal of this work will be the design and development of an integrated platform for *in silico* phenotype simulation of microbial strains. This platform should integrate the most relevant existing simulation methods from different platforms built in different computational languages. This platform should also enable its easy extension with other methods and tools that may be developed. The developed platform will be integrated with the OptFlux workbench for ME, developed by the host group of this work.

This work implies the following scientific/technological objectives:

- Review the existent phenotype simulation methods and the ME computational tools that implement them;
- Select and make a detailed study of the most relevant methods and tools;

- Design and implement these methods within an integrated computational platform that will also allow to integrate different ME platforms written over different systems and programming languages;
- Develop a user-friendly interface, in the form of a plugin for OptFlux that allows the use of the developed platform and all implemented phenotype simulation methods;
- Validate the performance and accuracy of the developed methods with selected case studies.

1.3. Structure of the document

This document is organized in the following way:

Chapter 2

Metabolic engineering: concepts and methods

An introduction to the Metabolic Engineering field and along with it a brief presentation of the genome-scale models used in the area. Some phenotype prediction algorithms will be presented that use these models, together with a review of some relevant ME tools.

Chapter 3

Core computational frameworks analysis

Introduce the frameworks OptFlux and COBRA toolbox. Explain how the model is created and used in both tools. Describe the structure of the constraint-based methods in these tools.

Chapter 4

Results and Development

Explained how the implementation of the platform was performed and also explain the development of the OptFlux plugin. Validate the plugin through case studies.

Chapter 5

Conclusions and future work

The summary of the work accomplished for this thesis and the future improvements and implementations.

Chapter 2

Metabolic engineering: concepts and methods

2.1. Background

Nowadays, chemical and pharmaceutical industries are strongly exploiting the metabolic capabilities of different microorganisms for the production of valuable compounds. Indeed, there is an increasing trend to replace chemical production processes with biotechnological methods based on microbial fermentations. Typically, microorganisms evolve to be adapted to their natural habitats and this implies requirements distinct from the industrial ones. So, to obtain a desired phenotype for industrial purposes, it is often needed to rearrange the genotype of the organism, to create so called cell factories. The recent development of genetic engineering techniques to make targeted changes over the genomes, made the reproduction of metabolic potentials of many different microorganisms possible. The field that handles the definition of the desired metabolic changes towards industrial goals is referred as Metabolic Engineering (ME) [9].

ME provides an integrated approach towards the conception of new cell factories by providing rational conception procedures and valuable mathematical and experimental tools. Mathematical models or *in silico* (computational) models play a vital role for phenotypic analysis, being used for the design of optimal metabolic network structures. Recurring to these models, it is possible to design improved metabolic networks by performing changes over the genotype of a selected host microorganism. Afterwards, an experimental evaluation can be done to check the results of the performed changes validating *in silico* predictions [10].

The nature of cellular metabolism and regulation is not always easy to understand and normally raises difficulties. Therefore, an analysis of the metabolism as a whole is usually required. The increasing amount of complete sequenced genomes for a large number of microorganisms, together with powerful Bioinformatics tools for genome annotation, provided the opportunity to develop genome-scale metabolic models (GSMMs). Also, these advances led to the creation of a large number of databases with experimental data, offering plenty information from different microorganisms. On the other hand, a question arises, how to extract relevant information from these large repositories of data to correctly use them to develop efficient industrial processes. Therefore, the main challenge of ME of this post-genomic era is to extend its design methodologies to include biological data on the whole-genome scale. The first stage of this challenge involves the reconstruction of genome-scale models and to study how the consistency and exactness of the predictions conceived using the model are associated to the nature and abstraction degree of the used model [10].

2.2. Genome-Scale Models

The reconstruction of metabolic networks has long been an objective of biochemistry. With the recent advances in genome sequencing, it became possible to convert these reconstructions into a mathematical format that represents the genotype-phenotype relations in the metabolism of a specific microorganism [11]. These mathematical formats, or models, contain biochemical, genetic and genomic information, which have been gradually incorporated into GSMMs. These models support the development of distinct methods for the computation of phenotypic behaviors in the form of fluxes for metabolic reactions [11], [12].

In the end of the 1990s, the first metabolic genome-scale reconstruction and GSMM was published [13]. Since then, there has been an exponential growth in the field of genome-scale metabolic network analysis. At the date of writing of this text, well over one hundred genome-scale metabolic reconstructions have already been published [14]. Two lists of models are available in the following web sites: http://systemsbiology.ucsd.edu/InSilicoOrganisms/OtherOrganisms and http://darwin.di.uminho.pt/models.

These models are able to predict which reactions occur in the cell and their fluxes, given specific conditions (e.g. media). The simulation of this behavior is based in external and internal variations. When these variations occur it may result in transformations in the cell in order to adapt and survive. The prediction of these fluxes is one of the main goals of the GSMMs. Therefore, ME aims to use these predictions to optimize an organism to achieve a certain goal, like the maximum production of a desired compound. [15]–[17].

Metabolic models can be grouped into two different types: stoichiometric models and kinetic models [9], [18], [19].

Kinetic models describe the metabolic system by joining kinetics information about particular cellular processes with known stoichiometry [18], [20], [21].

Kinetic models use the dynamic properties of the metabolic network, but an important problem related with defining these models is the lack of available kinetic data and the difference between *in vivo* and *in vitro* kinetic parameters, which leads to greater difficulties in reconstructing these type of models [22].

Given this difficulty this project will be aimed at the study of phenotype simulations using stoichiometric models.

2.2.1. Stoichiometric models

Stoichiometric models represent the metabolic network as a set of stoichiometric equations that desirably correspond to the full set of biochemical reactions in the system. The metabolic network consists of a set of nodes, defined by compounds and a set of edges that represent reactions. The knowledge of the biochemical reactions and their stoichiometries can be mined from several sources, as pathway databases (e.g. KEGG or MetaCyc), published literature or annotated genome information. Then, this information is used to build a stoichiometric matrix assuring the adequate mass balance. The rows of the matrix typically represent metabolites, while the columns represent reactions. This matrix is of great importance because it represents the interpretation of biological knowledge into mathematical expressions.

If a steady-state assumption is taken it is considered that the sum of all the fluxes that produce, transport and degrade an internal metabolite is zero and thus resulting in a stationary concentration. This assumption generates the following flux balance equation:

$$S.v = 0 \tag{2.1}$$

In this equation, *S* is the matrix that encompasses the stoichiometry of the reactions, while *v* is the vector of metabolic fluxes for each reaction.

This system of linear equations is usually underdetermined, since the number of fluxes typically surpasses the number of metabolites. Consequently, a multiplicity of solutions (flux distributions) exists. A particular solution may be found recurring to linear optimization, by declaring an objective function and looking for its maximal value within the stoichiometrically defined domain [12], [13].

Equation 2.1 describes all feasible flux distributions [23] instead of an individual solution. Although being feasible, that does not mean that those solutions have a biologically sense, which means that there is a need to further restrict the solution space. It is possible to restrict the model by limiting the value of the fluxes:

$$\alpha_i \leq v_i \leq \beta_i \tag{2.2}$$

where v_i is the value of the flux, and α_i and β_i are the bounds defined for the flux i.

There are two main forms of restrictions, adjustable and non-adjustable. The former constraints may change through evolution and are specific for different organisms [24] and are usually used to validate the model under specific conditions. The non-adjustable restrictions are, for instance, restrictions imposed by thermodynamics or enzyme capacities.

The restrictions shown in Equation 2.2 allow setting limits to the flux values and also providing ways to restrict the model from the reversibility of the reactions, or by restricting the fluxes to a constant value. A reaction is reversible if $\alpha_i \in \mathbb{R}-, \beta_i \in \mathbb{R}+$, and irreversible if either α_i or β_i are zero. Setting the value of a flux is possible by giving the same value to α_i and β_i. Biologically speaking, these flux restrictions can be considered environmental and/or genetic conditions.

The procedure of stoichiometric modelling of a cell starts by determining the stoichiometric matrix, which will give the mass balances, by assuming a pseudo steady state of the system

(Equation 2.1), following a constraint based approach and limiting the value of the fluxes through constraints (Equation 2.2).

There are different approaches when performing analyzes over this system. When a single solution is desired, under a specific environment, the solution can be acquired by constraining and/or defining some of the fluxes. In metabolic flux analysis (MFA) some exchange fluxes are measured to try to render a determined equation system. This approach can also be combined with extra data provided by the measurement of labeling patterns of specific metabolites to determine the single solution, often referred to as metabolic network analysis (MNA) [14]–[16]. In predictive studies, the methodology of Flux Balance Analysis (FBA) uses an optimization approach, by reporting to linear programming to determine the optimum flux distribution using a specified objective function, for instance the growth maximization (maximizing a defined artificial biomass flux) or ATP production [17], [18].

Another approach is, instead of looking for a particular flux solution for the model, to perform analysis of the topology of the metabolic network through the use of convex analysis, determining sets of elementary flux modes or extreme pathways [19]–[22], which are minimal operating modes of the metabolic systems.

2.3. Application of Metabolic Models in Metabolic Engineering

The number of genome-scale models is growing year after year. These models are mostly stoichiometric and do not explicitly encompass kinetic information, regulatory mechanisms or other known cellular processes. The lack of information and some unawareness of complex cellular regulation are the major reasons for this. Consequently, it seems that the complete *in silico* representation of the cellular behavior is a goal still far from being achieved. Nevertheless, the actual stoichiometric GSMMs present challenges to understand, extract and use all the information that they encompass. As a result, the analysis of these models has revealed some potential in relating genotype with phenotype [25].

As mentioned earlier, ME attempts to optimize the metabolic processes of microorganisms to obtain desired products. This objective has motivated the development of computational approaches to identify targets for genetic intervention, taking advantage of the recent progress in the reconstruction of GSMMs. Mainly, optimization-based approaches have been heavily used for the benefit of ME by identifying the best candidates where changes occurring in GSMMs could lead to improve the production of the intended compound [26]. These methods have been applied in a variety of biotechnology and biomedical applications for the identification of capable biochemical routes for the production of valuable chemicals using microorganisms [27]–[37], for the comprehension of disease metabolism [38]–[40] and the identification of drug targets [41], [42].

2.3.1. Structure of a Mathematical Optimization Problem

A mathematical optimization problem consists in finding the best solution from a set of existing possibilities that obey the constraints defined for the specific problem. These conditions comprise an objective function and a series of constraints. The objective function is the property of the system that will be maximized or minimized. As for the constraints, they are represented in two groups, the steady-state conditions for the reactions are equations, while inequalities are imposed by flux bounds.

As regards the types of problem variables, these can be continuous or discrete, while some problems combine variables of both types. Continuous variables represent continuous properties of the system such as reaction fluxes. Discrete variables are used in discrete decisions like the number of reactions that should be added to the metabolic model or 'yes/no' decisions, such as 'the reaction X will be knocked out or not' which are represented by a specific type of discrete variables, binary variables.

These mathematical problems can be divided in different mathematical programming paradigms. In a linear programming (LP) problem, both the objective function and constraints are linear and all variables are continuous. If one of the terms is non-linear, then it is named a nonlinear programming (NLP) problem. In the cases where there are both continuous and discrete (integer or binary) variables, the problem is named mixed-integer programming (MIP), which can be MILP or MINLP for linear or nonlinear cases respectively. The bi-level programming is another group of mathematical optimization problems, which in a nutshell span the cases where there is a problem within another problem. Here, the inner problem is normally considered a phenotype simulation layer and the outer problem the strain optimization layer which optimizes changes to impose to the organism. This bi-level programming supports both continuous and integer variables and linear or nonlinear objective function and constraints [26].

2.3.2. Algorithms for Phenotype Simulations

There are several methods to generate and explore predictions based on metabolic models (Table 1). These methods have distinct objectives, such as to estimate the maximum theoretical yields, classify the reactions relations, quantify fluxes based on experimental data and analyze results of genetic mutations. Some of these methods will be briefly explained below, while a more comprehensive overview is provided in Table 1 [26].

Flux Balance Analysis (FBA) is one of the dominant and most studied mathematical approaches for phenotype prediction based on GSMMs [44]–[46]. It is an approach that, given the reaction stoichiometry of a microorganism, has the ability to quantitatively predict its growth rate or the production rate of a specific metabolite, being this feature that makes this method so appreciated [47]. The FBA prerequisites are the existence of a metabolic model, a defined biological objective and the environmental conditions.

FBA uses LP to determine the steady-state flux distribution in a metabolic network by maximizing an objective function, for instance, the biomass flux or ATP production. Biomass production is one of the most used objective functions, based on the premise that the metabolic objective of the cell is to maximize growth. On the other hand, the FBA approach can also be used for the phenotype simulation of mutant strains, for instance predicting the effects of disabling one or multiple genes (or reactions). These are considered genetic conditions, where the value of the reactions encoded by these genes is restricted to null.

Although FBA can be used for predicting fluxes in mutant strains, usually these organisms are created in the laboratory and are not exposed to the same evolutionary pressure as the wild-type. Taking this into account, the use of FBA is debatable, and probably is not the best. Alternatively, some studies considered the variation of the objective function resulting in another group of techniques, which include for example, the minimization of metabolic adjustments (MOMA) [5]. The MOMA approach assumes that the mutant cell attempts to minimize the perturbation caused by the mutation in terms of flux variation. This means that, considering a given wild type reference flux distribution (known or obtained using FBA), the MOMA solution will be as close as possible from the reference distribution. In this case, the objective function is the minimization of the quadratic distance of the flux values from both solutions, and therefore is formulated as a quadratic programming (QP) problem.

On the other hand, regulatory on/off minimization (ROOM) [43], is a similar method that minimizes the total number of significant flux changes from the reference flux distribution. This is performed under the assumption that the cell minimizes the effort of adaptation to the new mutation, and it has been shown that there has been evolutionary pressure to minimize the cost of gene expression [43]. In terms of mathematical optimization problem, ROOM is formulated as a mixed integer linear programming (MILP) problem.

In ROOM's predictions, the flux distributions are very different from FBA's predictions, but the value of the growth rate is actually very close, while MOMA's are typically significantly lower. In fact, flux distributions from ROOM are claimed to be more correlated with experimental data than the two other methods [43].

As mentioned above, FBA has some limitations in its predictions, limitations that can be addressed by the addition of constraints to the optimization problem. By adding some features, other methods were developed, incorporating regulatory information, such as regulatory FBA (rFBA) [48], steady-state regulatory FBA (SR-FBA) [49], and probabilistic regulation of metabolism (PROM) [51].

Other studies have focused mostly in the identification or prediction of a physiological relevant objective function based on experimental flux data and developed other kind of algorithms [17], [44], such as ObjFind and BOSS. The former, a bi-level optimization procedure that gives a quantitative score to the reliability of an objective function [17], and the latter, a method that infers new objective functions so that the difference between model prediction flux distribution and measured fluxes is minimized [44].

Another approach is provided by flux variability analysis (FVA), which demonstrates that the linear optimization can be used with other purposes than just identifying the maximum efficiency of a product or biomass. FVA can identify metabolic flux variability, this is, the minimum and maximum flux values for each reaction in the network [45], possibly providing additional constraints (e.g. minimum biomass values).

Flux coupling analysis (FCA) and its variations consist in a series of LPs performed for finding dependencies between fluxes of a metabolic network at steady-state. FCA verifies the feasibility of the network, when a reaction is blocked and the other is carrying flux [46], [47].

Table 1: Optimization algorithms to explore model predictions

Name	**Task**	**Accessibility**	**Reference**
Flux balance analysis	Uses LP to determine fluxes that maximize a given linear objective function, most commonly biomass or ATP	Various, including MATLAB (via COBRA toolbox)	[48]–[50]
rFBA	Extends FBA to account for regulation	N/A	[51]
SR-FBA	Extends FBA to account for regulation	OptFlux	[52]
GIMME	Identify inactive reactions in a metabolic model under a particular condition or in a specific cell type	GAMS	[53]
PROM	Integrate transcriptional regulatory networks and constraint-based modeling	N/A	[54]
MOMA	Calculate fluxes in response to genetic modifications	COBRA and OptFlux	[5]
ROOM	Calculate fluxes in response to genetic modifications	OptFlux	[43]
Geometric FBA	Extends FBA to provide a standard, central, reproducible solution.	MATLAB (via COBRA toolbox)	[55]
ObjFind	Identify hypothesized objective functions consistent with experimental flux measurements	N/A	[17]
BOSS	Identify hypothesized objective functions consistent with experimental flux measurements	N/A	[44]
FVA	Determine minimum and maximum flux values for each reaction in the network	COBRA and OptFlux	[45]
fastFVA	Determine minimum and maximum flux values for each reaction in the network	MATLAB	[56]
FCF	Identify flux couplings in a reaction network	GAMS	[46]

FFCA	Identify flux couplings in a reaction network	MATLAB	[47]
F2C2	Identify flux couplings in a reaction network	MATLAB	[57]
SL Finder	Identify synthetic lethals in a genome-scale model	GAMS	[58]
OptMeas	Determine measurement sets that enable flux elucidation	C	[59]
13C-FLUX	MFA calculations	N/A	[60]

Source: [26]

2.3.3. Algorithms to Redesign Metabolic Networks

The following algorithms (Table 2) represent approaches for the redesign of metabolic networks and can be divided into three groups: pinpointing gene/ reaction deletions, identification of non-native additional pathways and identification of combinations of knockouts, down-regulation and overexpression that lead to the overproduction of a chosen chemical. Some of the most used optimization methods are presented in Table 2 [26].

Gene/Reaction knockout identification

Microorganisms can metabolize and secrete a large quantity of compounds. Most of these are not necessary when seeking to produce a specific target. Thus, a genetic intervention seeks to optimize the microorganism to maximize the production of a target, while minimizing the secretion of secondary compounds that are not profitable. Once a genetic intervention is performed, it is also important that cell growth is not affected significantly.

The OptKnock algorithm was developed to suggest the deletion of specific reactions so that these two requirements are maintained. This method uses a bi-level optimization, searches the optimal set of reactions deletions that maximizes the desired product secretion rate, while subject to the inner layer problem that guarantees steady-state conditions and seeks to maximize the biological aim. A bi-level approach can be altered into a single level MILP [31]. This method can sometimes be overly optimistic, it attempts to determine reaction deletions that optimize product yield, but does not verify for cases where an alternate optimal solution might divert flux away from producing the desired compound. The RobustKnock method, by employing a bi-level max-min optimization, assures a minimal production rate by considering alternate optima that produce less of the desired chemical [61].

To conduct a global search for ME strain designs is difficult because there are a large number of possible genetic manipulations in an organism. Genetic algorithms can be used to iteratively refine strain designs until a strain is found that is able to produce a compound of interest. In OptGene, a set of metabolic genes is provided for each member in an *in silico* population, as well as the number of desired gene deletions. Based on the desired objective, it is determined a fitness score for each individual. The gene vectors of the fittest members of the population are chosen for crossover and subjected to random mutation. Until a genotype providing a desired phenotype is found, this action is carried out [66].

The tilting of the objective function was introduced as an alternative to RobustKnock. It was implemented in OptGene and OptKnock and simulates the worst-case scenario for product formation. Tilting was implemented by adding the product transport to the objective weighting vector [62]. This method is easier to implement and more computationally tractable compared to RobustKnock.

Identification of non-native additional pathways

In OptStrain, in addition to designing growth-coupled production strains, foreign metabolic pathways may be added to expand the range of producible metabolites. Using curated databases containing known enzyme-catalyzed reactions, such as KEGG [63], MetaCyc [64] or BRENDA [65], the minimal number of reactions needed to synthesize a compound of interest is added to the metabolic network of the production strain. Afterwards, bi-level or MILP approaches can be used to identify a set of reaction deletions that optimizes the product of interest, subject to optimal *in silico* growth [66].

SimOptStrain identifies simultaneous gene deletions and non-native reaction additions. BiMOMA, instead of the maximization of the biomass, uses MOMA as the cellular objective in the inner problem to recognize gene knockouts that result in the overproduction of a sought compound [67].

Identification of multiple types of genetic interventions

Desired products can be obtained not only through the deletion of extant metabolic reactions, but also through the up- and down-regulation of other reactions. These modifications can be identified *in silico*. The OptReg extends OptKnock and allows for the prediction of reactions that can be up or down-regulated to aid in strain design [68].

The Genetic Design through Local Search (GDLS) method provides a flux modulation evaluation from the viewpoint of fitness of recombinant strains. It relies on local searches with multiple search paths to find combinations of genetic interventions. A heuristic approach is defined in which it is established the *k* number of genetic manipulations and the number of *M* best solutions desired. MILP is then used to make *k* more genetic changes and keep the *M* best solutions. These are then used as new initial points for *k* more changes. This is iterated until no more improved solutions are found. A faster computation of more complex gene modifications sets is possible through this method, but it is not guaranteed to provide a global optima [69].

Flux scanning based on enforced objective flux (FSEOF) identifies reaction amplification targets to make a product of interest. This method assesses all metabolic fluxes to find the ones that increase when the product flux is forced [70].

OptORF uses a MILP formulation and integrates transcription regulatory rules to identify optimal gene deletions and over expressions to maximize the production of a desired compound [71].

OptForce identifies possible genetic manipulations by classifying whether the reaction flux must change to overproduce a particular compound of interest. The method further identifies a minimal set of fluxes that must be forced through manipulations so that the network is consistent with the overall objective of overproduction of the compound of interest [72].

Enhancing Metabolism with Iterative Linear Optimization (EMILiO) identifies sets of reactions that may augment the production of a growth-coupled product by optimizing their flux. It also predicts the flux range that would maximize yield [73].

Table 2: Optimization algorithm store design metabolic networks

Name	**Type of optimization problem**	**Type of intervention**	**Accessibility**	**Reference**
OptKnock	Bi-level, MILP	Knockouts	GAMS, OptFlux and COBRA	[31]
RobustKnock	Multi-level, MILP	Knockouts	MATLAB	[61]
OptGene	Evolutionary	Knockouts	OptFlux and COBRA toolbox	[74]
Objective tilting	Bi-level, MILP	Knockouts	MATLAB (via COBRA toolbox)	[62]
OptStrain	Bi-level, MILP	Addition of non-native reactions/pathways	N/A	[66]
SimOptStrain	Bi-level, MILP	Knockouts and addition of non-native reactions/pathways	N/A	[67]
BiMOMA	Bi-level, MINLP	Knockouts	N/A	[67]
OptReg	Bi-level, MILP	Knockouts, up regulations and down regulations	N/A	[68]
GDLS	Heuristic	Knockouts, up regulations and down regulations	MATLAB (via COBRA toolbox)	[69]
FSEOF	LP	Up regulations and down regulations	N/A	[70]
OptORF	Bi-level, MILP	Knockouts and over expressions (of both metabolic and regulatory genes)	N/A	[71]
OptForce	Bi-level, MILP	Knockouts, up regulations and down regulations	GAMS	[72]
EMILiO	Bi-level, MILP	Knockouts, up regulations and down regulations	N/A	[73]

Source: [26]

2.3.4. Further algorithms

The previous tables (Table 1 and Table 2) show some of the most important methods used in ME, and some of the most relevant to this project. Despite this fact, this is just a small sample when compared to the dozen of methods and variations that have been studied and developed so far. Table 3 [26] shows another group of methods with a brief explanation of its purpose. These are procedures to curate GSMMs by (i) pinpointing and filling network gaps, (ii) identifying and fixing growth and flux prediction inconsistencies and (iii) restoring thermodynamic feasibility [26]. A more complete list is being updated regularly by the systems biology research group of the University of California, San Diego and can be found online: http://sbrg.ucsd.edu/cobra-predictions-app/. This list shows more than six hundred papers related to all kind of methods and applications, which can be filtered by year, prediction application, computational prediction or the organisms used in the case studies.

Table 3: Optimization algorithms to correct/improve models of metabolism

Name	Task	Accessibility	Reference
GapFind	Identify dead-end metabolites	GAMS and MATLAB(via COBRA toolbox)	[75]
GapFill	Bridge the gaps (i.e., dead-end metabolites)	GAMS and MATLAB(via COBRA toolbox)	[75]
SMILEY	Reconcile growth prediction inconsistencies (single gene mutations)	MATLAB (via COBRA toolbox)	[76]
GrowMatch	Reconcile growth prediction inconsistencies (single gene mutations)	GAMS	[75]
Modified GrowMatch	Reconcile growth prediction in consistencies (single gene mutations)	N/A	[77]
Extended GrowMatch	Reconcile growth prediction inconsistencies (double and higher order gene mutations)	N/A	[78]
GeneForce	Identify and correct transcriptional regulatory rules using growth prediction inconsistencies (single gene mutations)	N/A	[79]

OMNI	Reconcile flux prediction inconsistencies	N/A	[80]
TMFA	Eliminate thermodynamically infeasible pathways and loops	N/A	[81]
ll-COBRA	Eliminate thermodynamically infeasible loops	MATLAB (via COBRA toolbox)	[82]

Source: [26]

2.4. Constraint-based analysis computational tools

With the development and widespread use of methods developed to support ME, some computational tools were implemented to improve and enhance the evolution that has occurred in this field.

2.4.1. Systems Biology Research Tool

The Systems Biology Research Tool (SBRT) is an open-source and multiplatform tool designed to give the user greater ease in performing tasks of systems biology. This tool supports more than 30 methods for analyzing stoichiometric networks, like FBA, FVA or reaction deletions tasks, and more than a dozen methods from fields like graph theory, geometry, algebra, and combinatorics. It is also an application that allows independent software developers to improve it by adding new functionalities [83].

2.4.2. CellNetAnalyzer

CellNetAnalyzer (CNA) is a toolbox for MATLAB (MATrix LABoratory is a commercial computing environment) that offers an extensive and user-friendly environment for structural and functional analysis of metabolic, signaling and regulatory networks. CNA particular strengths are methods for functional network analysis, i.e. methods that describe functional states, pinpointing intervention strategies, identifying functional dependencies, or expose qualitative predictions of the perturbations effects. The capability of the user to call external algorithms from external programs is also one of the strengths of the CNA [7].

2.4.3. COBRA toolbox

COnstraint-Based Reconstruction and Analysis (COBRA) toolbox is a package for MATLAB and one of the most complete computational tools used in ME. It is a tool with methods that simulate, analyze and predict a diversity of metabolic phenotypes resorting to GSMMs. These models can be imported and exported in different formats, a feature that is a great advantage over most tools.

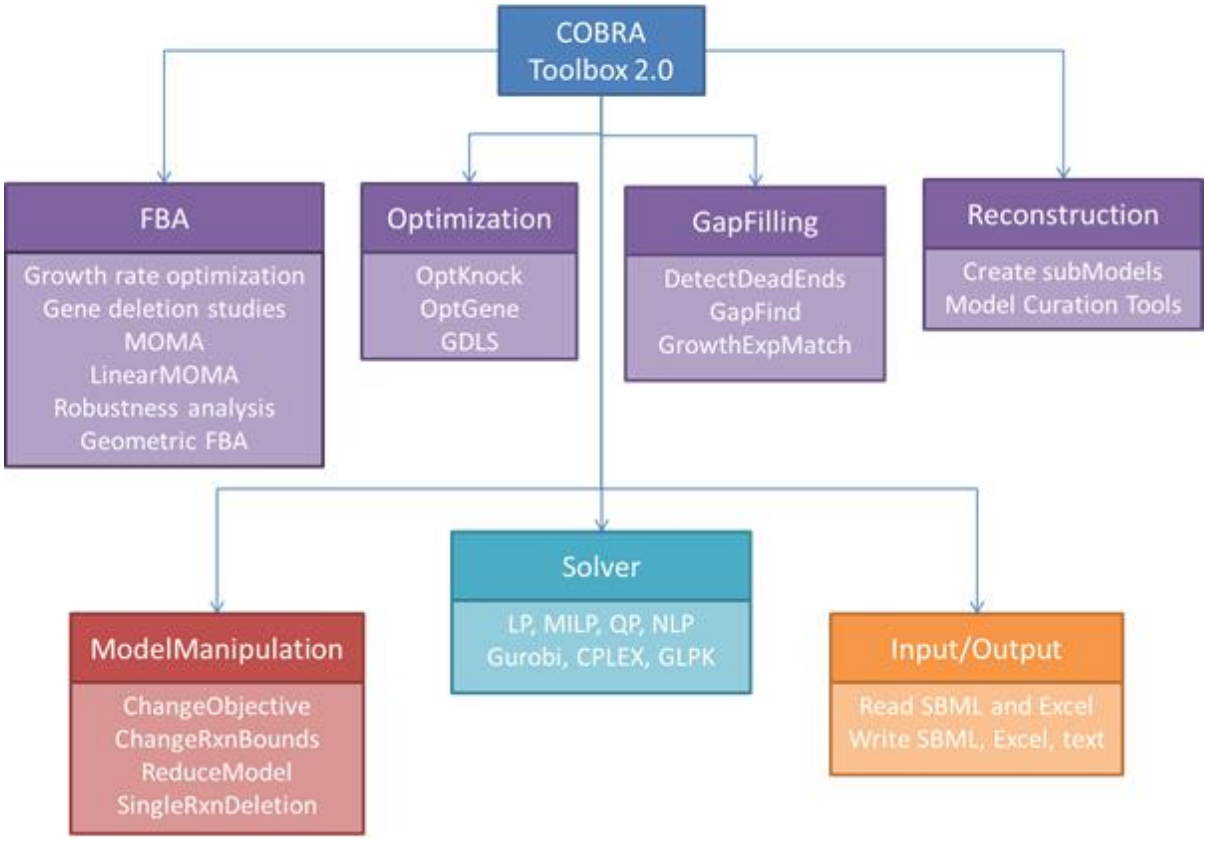


Figure 1 - Overview of the COBRA Toolbox

Figure 1 represents an overview of most of the components present in this tool. The tool consists in several categories of COBRA methods and its wide use by a large number of researchers is due to the quality and quantity of these methods. Operations like FBA, FVA, MOMA, GeometricFBA, Gap Filling, OptGene or OptKnock are available in this toolbox. In terms of solvers, COBRA toolbox contains solver interface functions for LP, MILP, QP, nonlinear programming problems and supports the IBM ILOG CPLEX Optimization Studio (CPLEX), Gurobi and GNU Linear Programming Kit (GLPK) solvers.

One of the disadvantages of this tool is that it is MATLAB-based. MATLAB is a numerical computing environment that requires a commercial license. Nevertheless, the growth of COBRA triggered its conversion to a Python-base framework (free programming language) [84] but this is still quite incomplete.

2.4.4. OptFlux

OptFlux is a free and open source software developed by the Bioinformatics and Systems Biology research group of the University of Minho. The goal of this tool is to provide a user-friendly computational tool for ME applications becoming a reference in this field.

This tool enables importation and exportation of GSMMs, and with it, metabolites, reactions and gene-reaction associations. This tool supports several flat file formats and it is compatible with the standard machine-readable format for representing models, Systems Biology Markup Language (SBML) files [85]. OptFlux also has a model repository that presents already curated models ready to be used.

In OptFlux, GSMMs can be used for phenotype simulation of both wild-type and mutant organisms, strain optimization, elementary modes analysis and flux variability analysis. The phenotype simulations are performed by methods such as FBA, ROOM and MOMA, while strain optimization is executed through the OptKnock algorithm, Evolutionary Algorithms or Simulated Annealing. The latter algorithms provide the identification of sets of reaction deletions that maximize a certain objective function, its major goal is to identify genetic modifications that result in a mandatory change of the microorganism to produce a specific metabolite without minimize its biomass production.

In terms of solvers, this software can use free or commercial solvers provided users have compatible licenses, including the Coin-or Linear Programming (CLP), the GLPK and CPLEX.

OptFlux's architecture is based in plugins, which allows, with some ease, its extension with new functionalities and thus a constant growth and evolution. It is implemented in Java on top of AIBench (http://www.aibench.org), which is a software development framework that was created as a cooperative project between the research group of the University of Minho and researchers from the University of Vigo in Spain. AIBench is a lightweight, non-intrusive, MVC-based Java application framework that simplifies the connection, execution and integration of operations with well-defined input/output, leading to units of work with good consistency that can be simply combined and reused [8].

2.4.5. Other computational tools

The presented computational tools are some of the most relevant in the ME field. Beside then, some others were developed and are often used in this area. Tools such as BioMet Toolbox [86], FASIMU [87], SurrayFBA [88], TIGER [89] and RAVEN [90] are capable of performing phenotype simulations or optimization analyzes and other significant ME tasks.

Chapter 3

Computational frameworks analysis

In this chapter, two of the main frameworks for ME, the OptFlux framework and the COBRA toolbox will be explained in further detail, from the model representation to the constraint-based methods included and how they are performed. These tools have been the ones used in this study, and therefore their knowledge is important to be able to create a connection between them.

3.1. Model structure

The GSMMs representation presented in the previous chapter is used both by OptFlux and COBRA to perform several methods, including phenotype simulation, strain optimization, analysis, and model modification. Although the models are built and computationally represented differently in each of these tools, the underlying mathematical framework is the same. The present section seeks to explain how the tools perform the model construction and also how its structure is represented in both these tools.

3.1.1. Metabolic

The GSMMs in OptFlux is built and managed through two different software projects. These projects are the *BioComponents* and the *Metabolic*.

BioComponents project

The *BioComponents* is a project with a significant Input/Output (I/O) component that is responsible for the creation and management of a structure named *Container*, which is used as the major data structure to keep the whole set of information needed for a GSMM.

The *Container* structure consists of several kinds of information about a certain organism and allows the creation of the organism model.

The *Container* structure holds all the information in smaller data structures and lists:

- The organism name;
- The biomass reaction identifier;
- A list of *CompartmentCI* objects, which holds the information of all organism's compartments, their names and a list of all their associated metabolites;
- A list of *MetaboliteCI* objects, which holds the information of all organism's metabolites, its name and a list of all its associated reactions;
- A list of *ReactionCI* objects, which holds the information of all organism's reactions. It is a structure slightly more complex than *CompartmentCI* or *MetaboliteCI*. Here, the name of the reaction, its reversibility, and two lists with the reactants and the products, are kept. The reactants/ products lists contain the name of the metabolite and the *StoichiometricValueCI* structure which has the name of the metabolite, in which compartment it is presented and its stoichiometric value in the reaction;
- A list of *ReactionConstraintsCI*, which holds the constraint values (the upper and lower bounds) for each reaction of the organism.

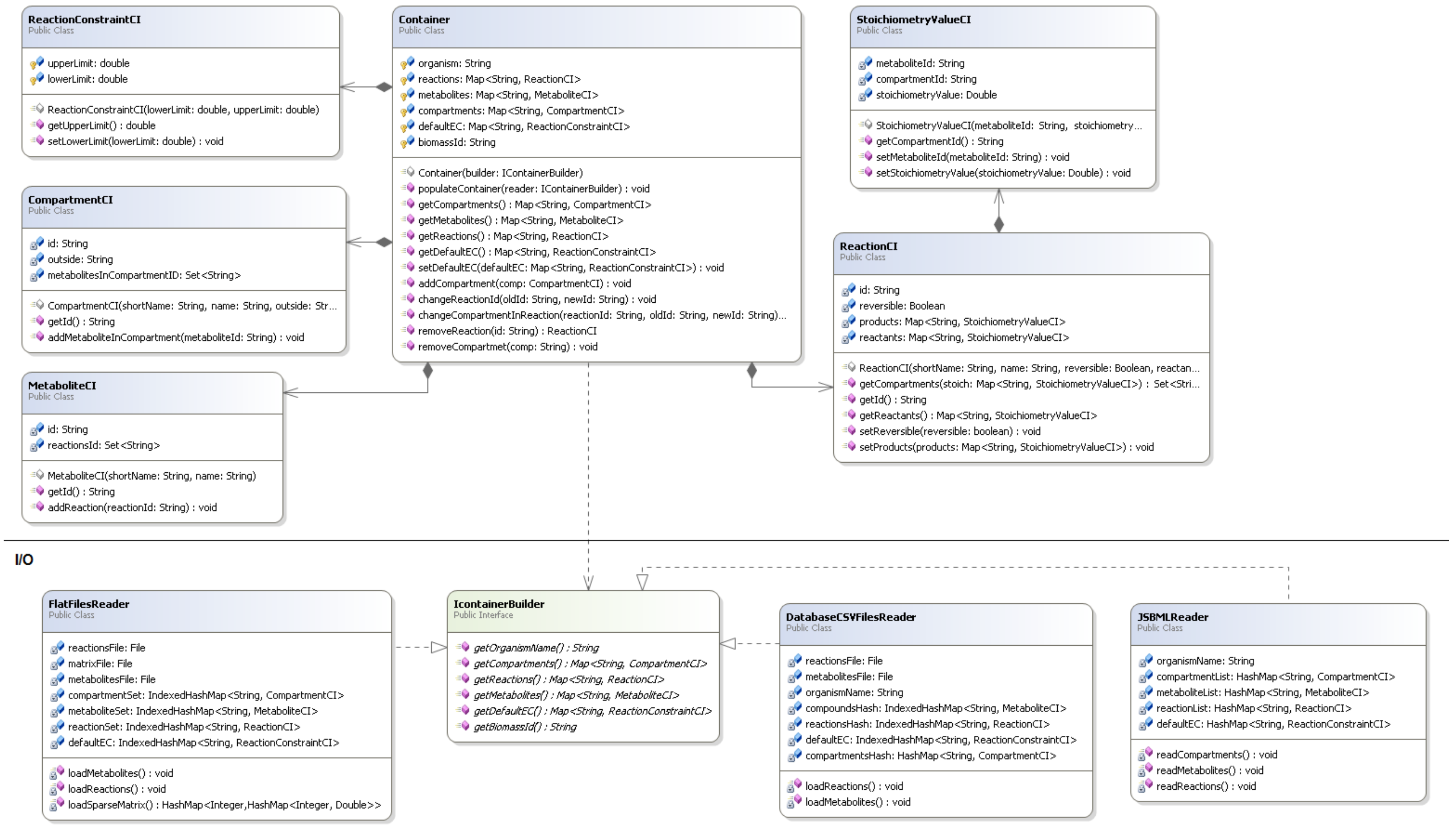

ReactionConstraintCI
Public Class
upperLimit: double
lowerLimit: double
ReactionConstraintCI(lowerLimit: double, upperLimit: double)
getUpperLimit() : double
setLowerLimit(lowerLimit: double) : void
Container
Public Class
organism: String
reactions: Map<String, ReactionCI>
metabolites: Map<String, MetaboliteCI>
compartments: Map<String, CompartmentCI>
defaultEC: Map<String, ReactionConstraintCI>
biomassId: String
Container(builder: IContainerBuilder)
populateContainer(reader: IContainerBuilder) : void
getCompartments() : Map<String, CompartmentCI>
getMetabolites() : Map<String, MetaboliteCI>
getReactions() : Map<String, ReactionCI>
getDefaultEC() : Map<String, ReactionConstraintCI>
setDefaultEC(defaultEC: Map<String, ReactionConstraintCI>) : void
addCompartment(comp: CompartmentCI) : void
changeReactionId(oldId: String, newId: String) : void
changeCompartmentInReaction(reactionId: String, oldId: String, newId: String)...
removeReaction(id: String) : ReactionCI
removeCompartmet(comp: String) : void
StoichiometryValueCI
Public Class
metaboliteId: String
compartmentId: String
stoichiometryValue: Double
StoichiometryValueCI(metaboliteId: String, stoichiometry...
getCompartmentId() : String
setMetaboliteId(metaboliteId: String) : void
setStoichiometryValue(stoichiometryValue: Double) : void
CompartmentCI
Public Class
id: String
outside: String
metabolitesInCompartmentID: Set<String>
CompartmentCI(shortName: String, name: String, outside: Str...
getId() : String
addMetaboliteInCompartment(metaboliteId: String) : void
ReactionCI
Public Class
id: String
reversible: Boolean
products: Map<String, StoichiometryValueCI>
reactants: Map<String, StoichiometryValueCI>
ReactionCI(shortName: String, name: String, reversible: Boolean, reactan...
getCompartments(stoich: Map<String, StoichiometryValueCI>) : Set<Stri...
getId() : String
getReactants() : Map<String, StoichiometryValueCI>
setReversible(reversible: boolean) : void
setProducts(products: Map<String, StoichiometryValueCI>) : void
MetaboliteCI
Public Class
id: String
reactionsId: Set<String>
MetaboliteCI(shortName: String, name: String)
getId() : String
addReaction(reactionId: String) : void
I/O
FlatFilesReader
Public Class
reactionsFile: File
matrixFile: File
metabolitesFile: File
compartmentSet: IndexedHashMap<String, CompartmentCI>
metaboliteSet: IndexedHashMap<String, MetaboliteCI>
reactionSet: IndexedHashMap<String, ReactionCI>
defaultEC: IndexedHashMap<String, ReactionConstraintCI>
loadMetabolites() : void
loadReactions() : void
loadSparseMatrix() : HashMap<Integer,HashMap<Integer, Double>>
IcontainerBuilder
Public Interface
getOrganismName() : String
getCompartments() : Map<String, CompartmentCI>
getReactions() : Map<String, ReactionCI>
getMetabolites() : Map<String, MetaboliteCI>
getDefaultEC() : Map<String, ReactionConstraintCI>
getBiomassId() : String
DatabaseCSVFilesReader
Public Class
reactionsFile: File
metabolitesFile: File
organismName: String
compoundsHash: IndexedHashMap<String, MetaboliteCI>
reactionsHash: IndexedHashMap<String, ReactionCI>
defaultEC: IndexedHashMap<String, ReactionConstraintCI>
compartmentsHash: HashMap<String, CompartmentCI>
loadReactions() : void
loadMetabolites() : void
JSBMLReader
Public Class
organismName: String
compartmentList: HashMap<String, CompartmentCI>
metaboliteList: HashMap<String, MetaboliteCI>
reactionList: HashMap<String, ReactionCI>
defaultEC: HashMap<String, ReactionConstraintCI>
readCompartments() : void
readMetabolites() : void
readReactions() : void

Figure 2 - BioComponents class diagram

This *Container* structure was built to have a better manipulation of all organisms' information. This manipulation has the capability of adding, changing or even removing all the existent data. With this capability, it is possible, for instance, to add new compartments or reactions to the organism, to change in which compartment a specific metabolite is present, to remove reactions, to change its constraints or even to change the stoichiometric value that a specific metabolite has in a certain reaction.

All this *Container* information is obtained from different readers. These readers are structures responsible for getting and handling data from different sources. Usually, the information needed for a constraint-based model can be found in files with different formats, and thus different readers were implemented. In the *BioComponents* project, there are a significant number of readers that cover almost all types of information sources, such as SBML, CSV Files, FlatFiles or Metatool.

Each reader implements the interface *IContainerBuilder*. This interface obliges the implemented reader to reply to all the information essential to create the model by implementing the interface methods. All readers must get information such as the list of compartments, metabolites and reactions from its source to fulfill the agreement with the *IContainerBuilder*. This treaty allows that, although the information sources can be very different, a *Container* is created using the *IContainerBuilder* and the *Metabolic* project is responsible to convert this structure in a constraint based model.

Metabolic project

Metabolic is a project where constraint-based methods for simulation, optimization and analysis are implemented. Also, it is responsible for the creation of GSMMs and their use using constraint-based methods.

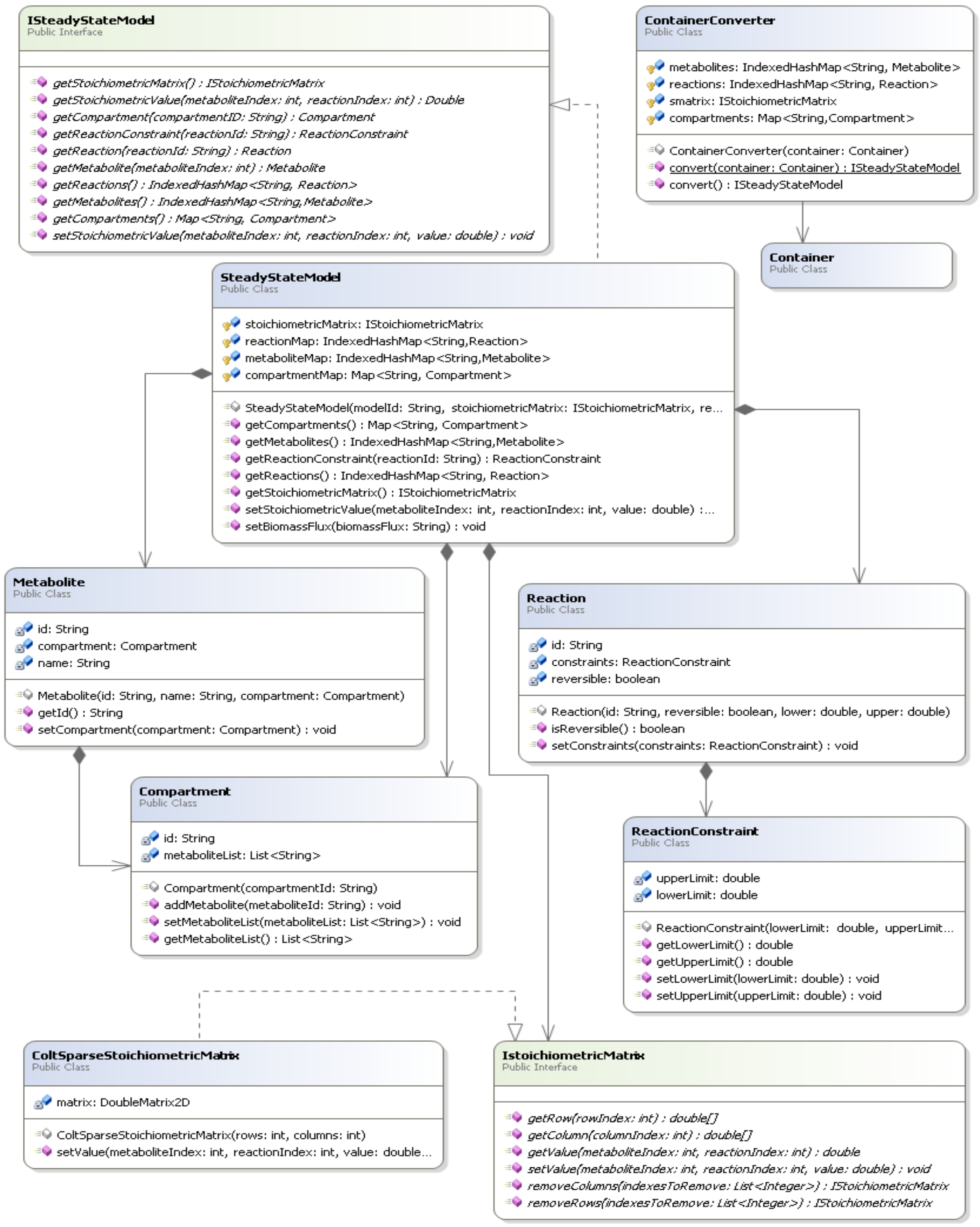

ISteadyStateModel
Public Interface
getStoichiometricMatrix() : IStoichiometricMatrix
getStoichiometricValue(metaboliteIndex: int, reactionIndex: int) : Double
getCompartment(compartmentID: String) : Compartment
getReactionConstraint(reactionId: String) : ReactionConstraint
getReaction(reactionId: String) : Reaction
getMetabolite(metaboliteIndex: int) : Metabolite
getReactions() : IndexedHashMap<String, Reaction>
getMetabolites() : IndexedHashMap<String,Metabolite>
getCompartments() : Map<String, Compartment>
setStoichiometricValue(metaboliteIndex: int, reactionIndex: int, value: double) : void
ContainerConverter
Public Class
metabolites: IndexedHashMap<String, Metabolite>
reactions: IndexedHashMap<String, Reaction>
smatrix: IStoichiometricMatrix
compartments: Map<String,Compartment>
ContainerConverter(container: Container)
convert(container: Container) : ISteadyStateModel
convert() : ISteadyStateModel
Container
Public Class
SteadyStateModel
Public Class
stoichiometricMatrix: IStoichiometricMatrix
reactionMap: IndexedHashMap<String,Reaction>
metaboliteMap: IndexedHashMap<String,Metabolite>
compartmentMap: Map<String, Compartment>
SteadyStateModel(modelId: String, stoichiometricMatrix: IStoichiometricMatrix, re...
getCompartments() : Map<String, Compartment>
getMetabolites() : IndexedHashMap<String,Metabolite>
getReactionConstraint(reactionId: String) : ReactionConstraint
getReactions() : IndexedHashMap<String, Reaction>
getStoichiometricMatrix() : IStoichiometricMatrix
setStoichiometricValue(metaboliteIndex: int, reactionIndex: int, value: double) :...
setBiomassFlux(biomassFlux: String) : void
Metabolite
Public Class
id: String
compartment: Compartment
name: String
Metabolite(id: String, name: String, compartment: Compartment)
getId() : String
setCompartment(compartment: Compartment) : void
Reaction
Public Class
id: String
constraints: ReactionConstraint
reversible: boolean
Reaction(id: String, reversible: boolean, lower: double, upper: double)
isReversible() : boolean
setConstraints(constraints: ReactionConstraint) : void
Compartment
Public Class
id: String
metaboliteList: List<String>
Compartment(compartmentId: String)
addMetabolite(metaboliteId: String) : void
setMetaboliteList(metaboliteList: List<String>) : void
getMetaboliteList() : List<String>
ReactionConstraint
Public Class
upperLimit: double
lowerLimit: double
ReactionConstraint(lowerLimit: double, upperLimit...
getLowerLimit() : double
getUpperLimit() : double
setLowerLimit(lowerLimit: double) : void
setUpperLimit(upperLimit: double) : void
ColtSparseStoichiometricMatrix
Public Class
matrix: DoubleMatrix2D
ColtSparseStoichiometricMatrix(rows: int, columns: int)
setValue(metaboliteIndex: int, reactionIndex: int, value: double...
IstoichiometricMatrix
Public Interface
getRow(rowIndex: int) : double[]
getColumn(columnIndex: int) : double[]
getValue(metaboliteIndex: int, reactionIndex: int) : double
setValue(metaboliteIndex: int, reactionIndex: int, value: double) : void
removeColumns(indexesToRemove: List<Integer>) : IStoichiometricMatrix
removeRows(indexesToRemove: List<Integer>) : IStoichiometricMatrix

Figure 3 - Metabolic class diagram - Model view

Figure 3 represents the class diagram responsible for the creation of the GSMMs in the *Metabolic* project. In this project the model is represented by a structure named *SteadyStateModel* (SSM) that implements the interface *ISteadyStateModel*. Although the SSM holds information also present in the *Container*, its representation and organization are different, since the SSM handles structures that are oriented for numerical computation to support constraint-based methods. One of the major differences is the stoichiometric matrix that exists in the SSM organizing stoichiometric coefficients in a suitable matrix representation. The need of having two similar structures lies in the manipulation of both structures, for instance, while in the container it is easy to remove reactions/metabolites, in the SSM that manipulation is more complex, since the data in this structure is indexed and linked in the matrix.

This structure was created with the purpose of being used for simulation, optimization and other constraint-based methods. This project has a class named *ContainerConverter* that, using all the necessary information present in the *Container*, creates a structure that implements the interface *ISteadyStateModel*, in this case the SSM.

Similarly to the *Container*, the SSM structure is composed by a set of data structures that represent the model. These structures are:

- A list of compartments represented by the *Compartment* structure. This structure consists in the compartment name and a list of associated metabolites;
- A list of metabolites represented by the *Metabolite* structure. This structure consists in the name of the metabolite and the name of the compartment.
- A list of reactions represented by the *Reaction* structure. This structure is slightly different from the Container's *ReactionCI* because it does not have information about the associated metabolites, on the other hand it contains the name of the reaction, its reversibility and the *ReactionConstraint* structure, which consists in the upper and lower bounds that a reaction can have;
- The *IStoichiometricMatrix* structure, it does not exist in the *Container* and it is the base of the constraint-based model. This structure represents the bond between metabolites and reactions. It is a matrix *m* X *n* in where *m* represents the number rows (metabolites), *n* represents the number of columns (reactions). The value of each matrix's cell is obtained from the *Container's ReactionCI* where the connection metabolite-reaction is presented.

All the structures that implement the *ISteadyStateModel*, such as SSM, are obligated to implement some essential functions to most of the constraint-based methods. These functions are the list of reactions, the list of metabolites, the whole stoichiometric matrix and the value of a certain matrix's cell. With these the constraint-based methods have full access to the constraint-based model to perform all its operations. This is the major reason for the development of the *ISteadyStateModel* interface in the *Metabolic* project.

3.1.2. COBRA

Since COBRA is a toolbox developed in the MATLAB environment it has its own ways of data management. The COBRA toolbox uses MATLAB and its numerical computing environment to hold data in a workspace. This workspace contains all the variables used and created in this environment. Here, there is not an order or even a data distribution by categories or classes in the same way other structured programming languages do. In this workspace, all the structures or variables are at the same level and there is complete freedom in the management of their content. The GSMM is a structure array that groups related data using data containers called fields. This structure in the COBRA toolbox consists of several fields, such as:

- ***Rxns***, a string vector where each element represents the ID of a reaction;
- ***Mets***, a string vector where each element represents the ID of a metabolite;
- ***S***, the stoichiometric matrix. It is a matrix *m* X *n* where *m* represent the number of rows (metabolites) and *n* represents the number of columns (reactions);
- ***Rev***, a numeric vector with the same number of elements as the Rxns field and where each element represents the reversibility of the correspondent reaction (binary value);
- ***Lb***, a numeric vector with the same number of elements as the Rxns field and where each element represents the lower bound of the correspondent reaction;
- ***Ub***, a numeric vector with the same number of elements as the Rxns field and where each element represents the upper bound of the correspondent reaction;
- ***C***, a numeric vector with the same number of lines as the Rxns field and where each row represents the objective coefficient of the correspondent reaction. This field is used to define which reaction or reactions will be in the objective function of the model;
- ***RxnsNames***, a string vector with the same number of elements as the *Rxns* field and where each row represents the name of the correspondent reaction;
- ***MetNames***, a string vector with the same number of elements as the *Mets* field and where each element represents the name of the correspondent metabolite.

For loading GSMMs in COBRA, it is necessary to define a list of metabolites, a list of reactions and the stoichiometric matrix to perform its constraint-based operations. There are three standard ways of loading these models in COBRA: through a SBML file, text files and Excel files.

Another approach, less conventional, of loading models in COBRA is through MAT files. These files contain the MATLAB formatted data of any variable or structure supported by this platform and, thus, the GSMMs can be saved and loaded in this format. COBRA does not have a method for loading these sorts of files and so it is not a COBRA functionality but a MATLAB one.

Although there are different approaches when loading the model, all the approaches have the same final result and a unique structure that represents a GSMM. In COBRA, this structure is not static and can be changed, there are some functions in the toolbox that allow to add reactions, remove metabolites or reactions and change the value of each cell of the stoichiometric matrix. On the other hand, being this toolbox built in MATLAB it is also possible to change all the variables within the workspace and thus modify the model. Some of this manipulation functions will be presented in 3.2.2.

This structure of the model is used by almost every operation of the COBRA toolbox. The constraint-based model is the center of all the tools that have constraint-based methods and COBRA is no exception. These methods can be categorized by simulation, optimization and analysis methods and both OptFlux and COBRA have methods that represent these three categories.

3.2. Constraint-based analysis structure

The constraint-based analysis has a great number of methods that are used in metabolic engineering. Within these methods there are some that can be categorized as phenotype simulation methods. A number of simulation methods are developed in the *Metabolic* project and also in the COBRA toolbox. This chapter seeks to explain how the simulation mechanisms in each of the tools work.

3.2.1. Metabolic

As mentioned before, the Metabolic project besides the creation of GSMMs also makes use of them in some implemented constraint-based methods. The implementation of these constraint-based methods, more specifically the phenotype simulation methods, can be divided into two different but connected layers: the abstract and simulation layers.

Figure 4 represents the class diagram of the abstract and simulation layers in the *Metabolic* project. In this project the simulation operations are defined by formulations, being these, for instance, the FBA, MOMA or ROOM. On the center of these formulations is the interface *ISteadyStateSimulationMethod*.

This interface makes all the simulation formulations that exist in this project implement all the interface methods. Through this interface, any constraint-based method that can fully implement these interface methods can be added to the project and have its results shown along with the existing formulations. The most important methods of this interface are:

- A method that returns an *ISteadyStateModel* structure;
- The access and change methods for the structure *EnvironmentalConditions*;
    - The *EnvironmentalConditions* structure is composed by a list of reactions and their *ReactionsContraints*, being this last structure the bounds of the reaction. The *EnvironmentalConditions* are used to modify the reactions bounds of an organism and in an *in vivo* analogy are for instance the environmental conditions that are imposed on the organism.
- The access and change methods for the structure *GeneticConditions*;
    - The *GeneticConditions* structure represents all genetic alterations made to the organism for a specific simulation. This structure is composed by a list of genes/reactions and their expression values, provided that when there are no expressions values the knockouts of the genes/reactions are considered;

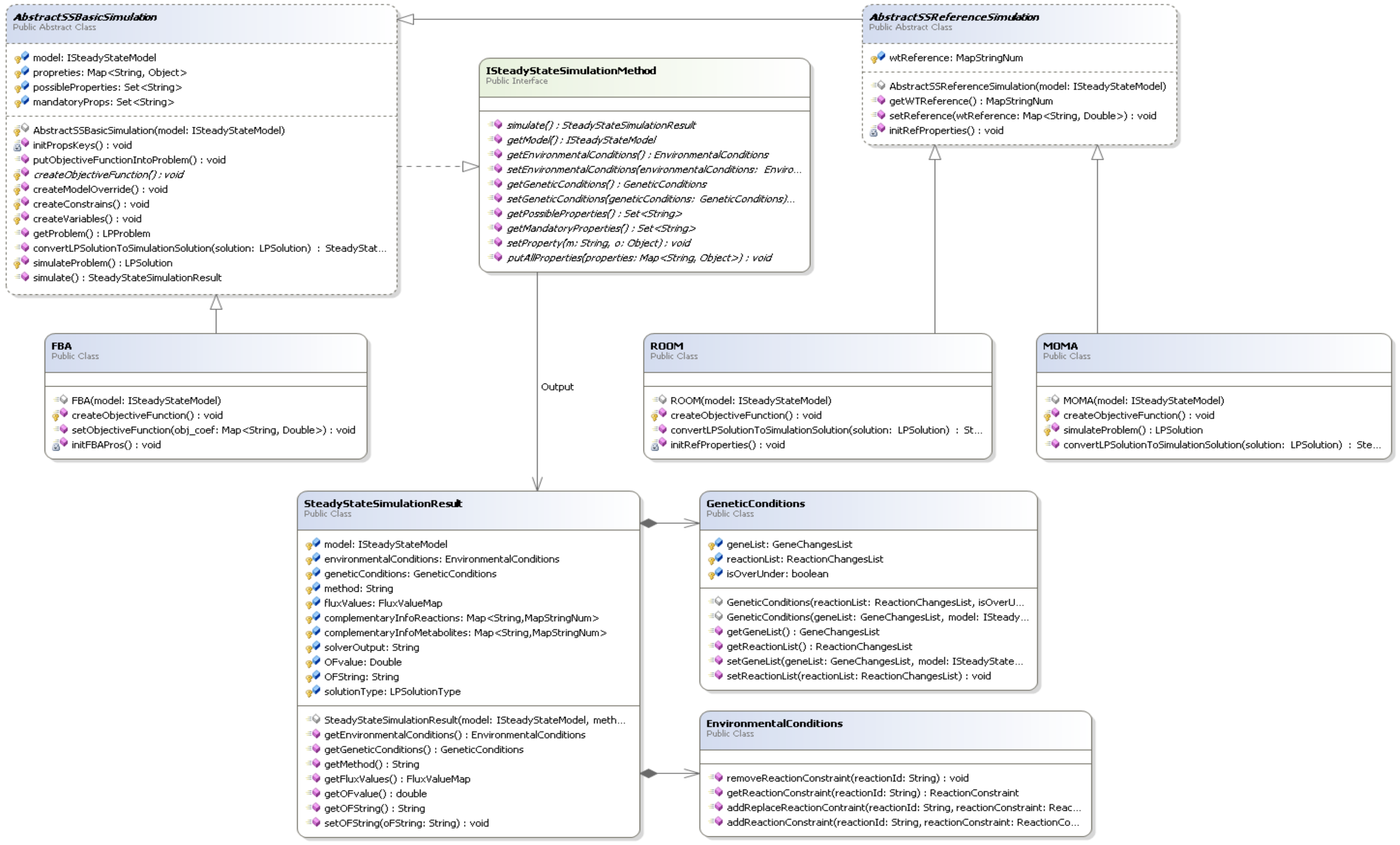

AbstractSSBasicSimulation
Public Abstract Class
model: ISteadyStateModel
propreties: Map<String, Object>
possibleProperties: Set<String>
mandatoryProps: Set<String>
AbstractSSBasicSimulation(model: ISteadyStateModel)
initPropsKeys() : void
putObjectiveFunctionIntoProblem() : void
createObjectiveFunction() : void
createModelOverride() : void
createConstrains() : void
createVariables() : void
getProblem() : LPProblem
convertLPSolutionToSimulationSolution(solution: LPSolution) : SteadyStat...
simulateProblem() : LPSolution
simulate() : SteadyStateSimulationResult
ISteadyStateSimulationMethod
Public Interface
simulate() : SteadyStateSimulationResult
getModel() : ISteadyStateModel
getEnvironmentalConditions() : EnvironmentalConditions
setEnvironmentalConditions(environmentalConditions: Enviro...
getGeneticConditions() : GeneticConditions
setGeneticConditions(geneticConditions: GeneticConditions)...
getPossibleProperties() : Set<String>
getMandatoryProperties() : Set<String>
setProperty(m: String, o: Object) : void
putAllProperties(properties: Map<String, Object>) : void
AbstractSSReferenceSimulation
Public Abstract Class
wtReference: MapStringNum
AbstractSSReferenceSimulation(model: ISteadyStateModel)
getWTReference() : MapStringNum
setReference(wtReference: Map<String, Double>) : void
initRefProperties() : void
FBA
Public Class
FBA(model: ISteadyStateModel)
createObjectiveFunction() : void
setObjectiveFunction(obj_coef: Map<String, Double>) : void
initFBAPros() : void
Output
ROOM
Public Class
ROOM(model: ISteadyStateModel)
createObjectiveFunction() : void
convertLPSolutionToSimulationSolution(solution: LPSolution) : St...
initRefProperties() : void
MOMA
Public Class
MOMA(model: ISteadyStateModel)
createObjectiveFunction() : void
simulateProblem() : LPSolution
convertLPSolutionToSimulationSolution(solution: LPSolution) : Ste...
SteadyStateSimulationResult
Public Class
model: ISteadyStateModel
environmentalConditions: EnvironmentalConditions
geneticConditions: GeneticConditions
method: String
fluxValues: FluxValueMap
complementaryInfoReactions: Map<String,MapStringNum>
complementaryInfoMetabolites: Map<String,MapStringNum>
solverOutput: String
OFvalue: Double
OFString: String
solutionType: LPSolutionType
SteadyStateSimulationResult(model: ISteadyStateModel, meth...
getEnvironmentalConditions() : EnvironmentalConditions
getGeneticConditions() : GeneticConditions
getMethod() : String
getFluxValues() : FluxValueMap
getOFvalue() : double
getOFString() : String
setOFString(oFString: String) : void
GeneticConditions
Public Class
geneList: GeneChangesList
reactionList: ReactionChangesList
isOverUnder: boolean
GeneticConditions(reactionList: ReactionChangesList, isOverU...
GeneticConditions(geneList: GeneChangesList, model: ISteady...
getGeneList() : GeneChangesList
getReactionList() : ReactionChangesList
setGeneList(geneList: GeneChangesList, model: ISteadyState...
setReactionList(reactionList: ReactionChangesList) : void
EnvironmentalConditions
Public Class
removeReactionConstraint(reactionId: String) : void
getReactionConstraint(reactionId: String) : ReactionConstraint
addReplaceReactionContraint(reactionId: String, reactionConstraint: Reac...
addReactionConstraint(reactionId: String, reactionConstraint: ReactionCo...

Figure 4- Phenotype Simulation Abstract Layer Class Diagram

- The access and change methods for the properties of the formulation;
    - These properties are parameters that are used in the formulations, for instance the solver that will be used or used to define a reference flux distribution;
- The simulation method that must return a *SteadyStateSimulationResult* structure.

Despite all formulations generating a result with the same structure and having the interface in common, each is used for different purposes and has different ways of building the result.

The common aspects can be found in the abstract class *AbstractSSBasicSimulation*, a class that is also used as a link between the formulations and the *ISteadyStateSimulationMethod*. Here, the interface methods are implemented and this class can be extended by all the simulation formulations. This abstract layer is also responsible for the construction of the problem based on the variables and constraints of each formulation. However, before this process, the abstract makes all the formulations that extend it specify their objective functions and create an empty problem that can be a LP problem or an extension of that problem, for instance QP or MILP.

Another important task of the abstract layer is the property registration, which can be mandatory or optional, and can vary with the formulation in question leading to different problems. The properties registered at this level are the ones common to every formulation, namely:

- The solver that will be used;
- A flag representing if the simulation is an over and under expression or a knockout one;
- The flux distribution used as reference in the simulation;
- The environmental conditions of the simulation;
- The genetic conditions of the simulation.

Despite this abstract class already registers some properties that are common to every formulation, there are others that must be registered by every formulation implementation based on its requirements. There is a property that is used in more than one formulation, but not shared by all. For this reason a second abstract class named *AbstractSSReferenceSimulation* was created. This extends the *AbstractSSBasicSimulation* and is responsible for the registration of this property of the formulations that use references in their problems:

- The reference that will be used by the formulation, usually is a flux distribution obtained by wild-type simulations;

Along with the registration of the properties, the class also has a function that is used when there is no registration of the reference. This function performs a wild-type simulation and use the resulting flux distribution as reference in the construction of the problem.

As stated, all the simulation formulations must implement the *ISteadyStateSimulationMethod* interface and have the responsibility to create the *SteadyStateSimulationResult* structure as final result. This structure is composed by the following elements:

- An *ISteadyStateModel* structure;
- The formulation used to create this structure, for instance FBA or MOMA;
- The solution type, for instance if the solution was optimal or infeasible;
- The objective function used;
- The value of the objective function for this simulation;
- The flux distribution of the simulation, a string-numeric map that consist in list of reactions and the correspondent numeric flux value;
- The used environmental conditions;
- The used genetic conditions;
- A map that corresponds to the metabolites complementary information. This map is very generic and can be anything that is associated with the metabolites;
- A map that corresponds to the reactions complementary information. This map is very generic and can be anything that is associated with the reactions.

Figure 4 also represents the simulation layer of the *Metabolic* project. This layer corresponds to the implementation of each formulation that extends the previous abstracts layers. Each formulation is responsible for the registration of the properties that are essential to create it. The same happens when the formulations need to create a problem that is not LP. For instance, the FBA formulation, being a LP problem has no need to build a problem and it can use the problem created by the *AbstractSSBasicSimulation*. However, the FBA formulation needs to register two additional properties:

- Which reaction or set of reactions will make the objective function of the problem. This property is not mandatory and by default it uses the biomass reaction as the objective function;

- If the objective function of the problem is maximization or minimization. It is a mandatory property and the problem cannot be created without this information.

On the other hand, other formulations such as MOMA or ROOM beyond the need of registering their own properties also need to construct the problem. Despite the fact that these formulations need to create different problems, MOMA, a Quadratic Programming (QP) problem and ROOM, a Mixed Integer Linear Programming (MILP) problem, share two properties. For this reason they extend the second abstract, the *AbstractSSReferenceSimulation*.

As before, these formulations are responsible for the registration of the necessary properties. The same happens for the construction of the problem, since the *AbstractSSBasicSimulation* only creates a LP problem and both MOMA and ROOM are QP and MILP problems, respectively. As for the result, each of these formulations has different objective functions and also need to define it in the problem and later in the *SteadyStateSimulationResult* structure.

Simulation Control Center

Along with the implementation of each formulation, the *Metabolic* project has a control center where all formulations are registered and executed without having to instantiate or know any of the classes structures.

Figure 5 represents the class diagram of this control center. This is specific for simulation formulations and is a generic structure that performs simulations according to the selected formulation.

This control center also manages the properties for each formulation in a simple approach, properties such as the maximization flag or the genetic and environmental conditions have their own access and change methods. Besides that, this control center also allows the removal or addition of new properties used in other formulations. Each formulation is registered in a factory and it is through this factory that new formulations are registered. Also, this factory knows the type of problem from the formulation and is responsible for knowing all the methods that the control center is able to perform.

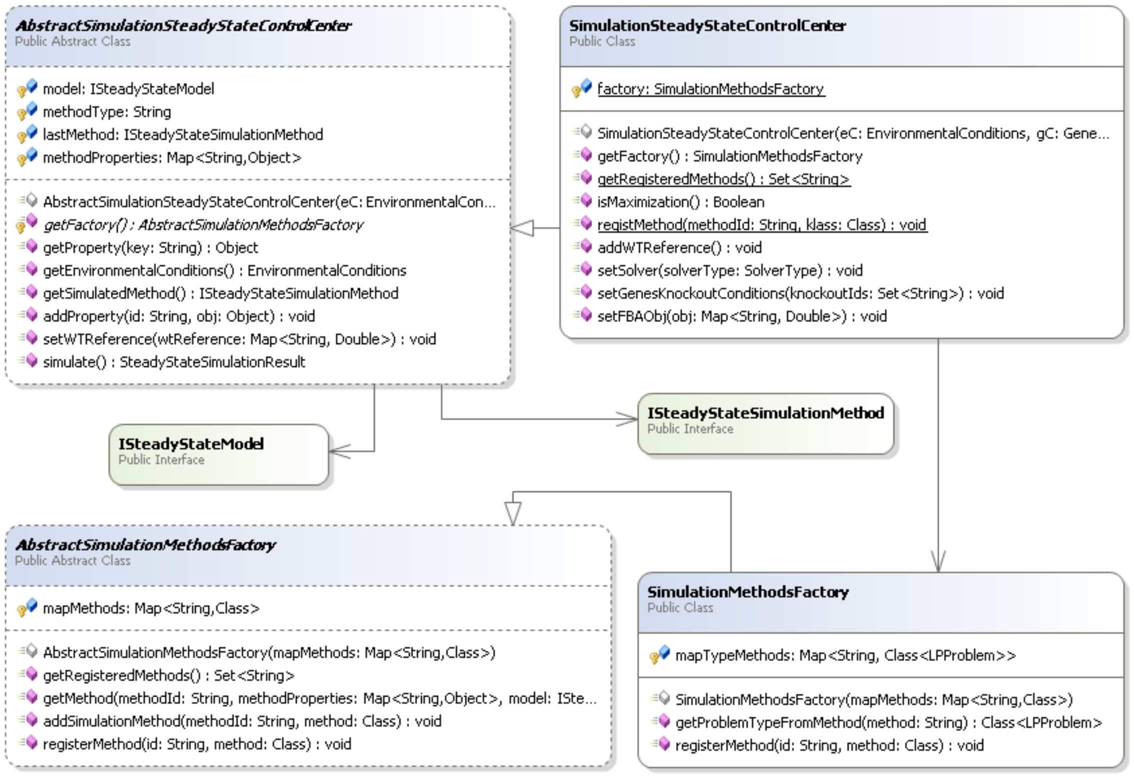

AbstractSimulationSteadyStateControlCenter
Public Abstract Class
model: ISteadyStateModel
methodType: String
lastMethod: ISteadyStateSimulationMethod
methodProperties: Map<String,Object>
AbstractSimulationSteadyStateControlCenter(eC: EnvironmentalCon...
getFactory() : AbstractSimulationMethodsFactory
getProperty(key: String) : Object
getEnvironmentalConditions() : EnvironmentalConditions
getSimulatedMethod() : ISteadyStateSimulationMethod
addProperty(id: String, obj: Object) : void
setWTReference(wtReference: Map<String, Double>) : void
simulate() : SteadyStateSimulationResult
SimulationSteadyStateControlCenter
Public Class
factory: SimulationMethodsFactory
SimulationSteadyStateControlCenter(eC: EnvironmentalConditions, gC: Gene...
getFactory() : SimulationMethodsFactory
getRegisteredMethods() : Set<String>
isMaximization() : Boolean
registMethod(methodId: String, klass: Class) : void
addWTReference() : void
setSolver(solverType: SolverType) : void
setGenesKnockoutConditions(knockoutIds: Set<String>) : void
setFBAObj(obj: Map<String, Double>) : void
ISteadyStateSimulationMethod
Public Interface
ISteadyStateModel
Public Interface
AbstractSimulationMethodsFactory
Public Abstract Class
mapMethods: Map<String,Class>
AbstractSimulationMethodsFactory(mapMethods: Map<String,Class>)
getRegisteredMethods() : Set<String>
getMethod(methodId: String, methodProperties: Map<String,Object>, model: ISte...
addSimulationMethod(methodId: String, method: Class) : void
registerMethod(id: String, method: Class) : void
SimulationMethodsFactory
Public Class
mapTypeMethods: Map<String, Class<LPProblem>>
SimulationMethodsFactory(mapMethods: Map<String,Class>)
getProblemTypeFromMethod(method: String) : Class<LPProblem>
registerMethod(id: String, method: Class) : void

Figure 5- Simulation Control Center Class Diagram

Both this control center and factory are important for this work since the integrated formulations in the platform are going to be registered and used through these structures.

3.2.2. COBRA

The COBRA toolbox has a large number of constraint-based methods and for that reason it is one of the most used tools in the ME. Being a toolbox that runs in a MATLAB environment its functionalities are based in scripts allowing the implementation of new methods and also an easy adjustment of already existing methods to fit the user needs.

Also the capabilities of MATLAB allow users to manipulate and analyze the information sets that are created by most of the constraint-based methods present in COBRA. Additionally, this toolbox can easily be interfaced with other software tools and so be used as a complementary tools, not only for tools running in the MATLAB environment but also in tools built in other languages such as Java, C, or Python.

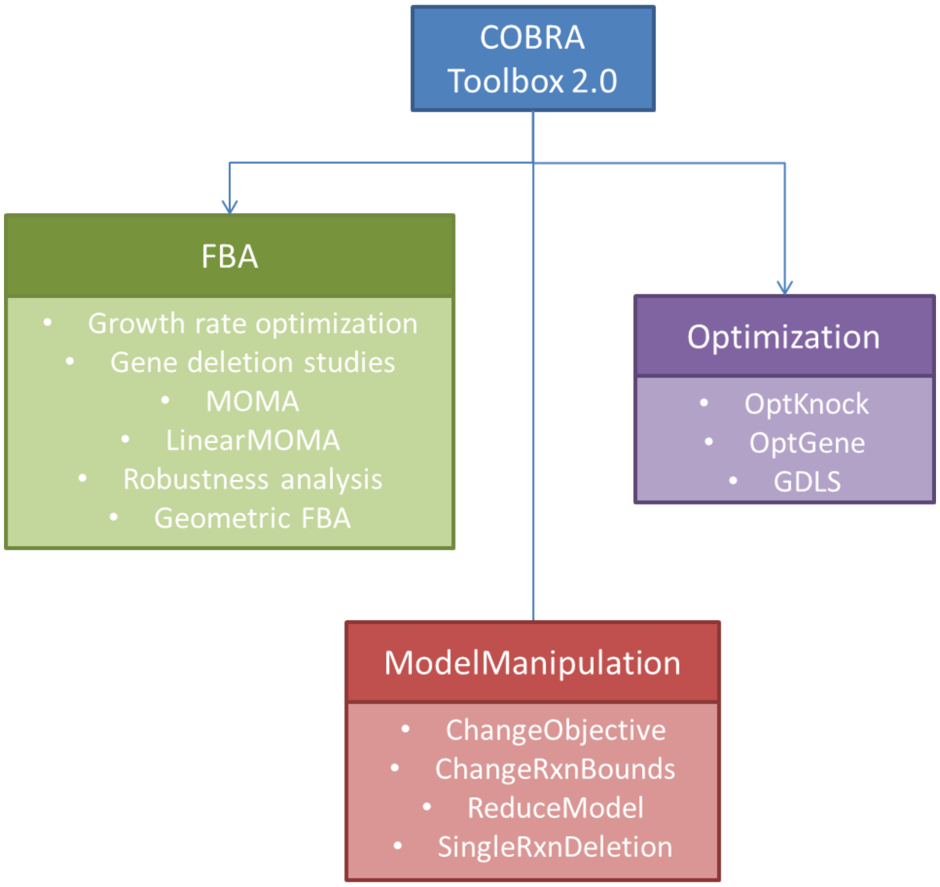


Figure 6 - COBRA Toolbox – Major methods

Figure 6 presents an overview of the COBRA toolbox and the tools that were more studied for this work. The toolbox can be divided in layers:

- The model manipulation layer, where it is possible to change the objective function, the reactions bounds, reduce the model by removing unused reactions or remove a single reaction;
- The FBA layer that contains a commendable number of methods, from growth rate optimization (FBA problem) to the MOMA method. This layer will be explained in detail further in this document;
- The optimization or design layer that contains some optimization methods such as OptKnock and GDLS, which will be also explained in detail in this document.

As mentioned, the FBA layer contains a vast number of methods, although these methods are all in the same group their result and input parameters are slightly different. Each method has its own result structure and is responsible for the creation of the problem, the choice of the solver to use and the creation of the result depending on the solver solution. Nonetheless, the constraint-based model is the center of these methods and of a vast number of the other groups of methods in this layer. This makes the model the only element in common between all them.

Each method is a MATLAB function responsible for creating the structure of the problem and the final result. Also, there are no classes or interfaces that are common to all of the methods and, thus, there is no obligation to comply with any structure. Each function is independent and free to implement the formulation and also free to output any solution structure. This can also be a disadvantage because sometimes there are functions that perform the same tasks with minor modifications and the scripts have duplicated parts: an example can be seen in the LinearMOMA and MOMA, where both functions need the same input and create the same result and thus have a very similar script.

The optimization layer is very similar to the previous one. Here, each of the methods is responsible for creating its result and the input parameters are slightly different in each one.

The modifications in the model, such as changing the reaction constraints or the objective function, will be made before performing the method by other COBRA functions. This could also be different if the model structure was a class and these functions were associated with the model since its purpose it to modify the data of the model.

The methods within this layer use a solver that must be defined before the use of the function. COBRA supports solver such as Gurobi, CPLEX or GLPK, and this definition of each solver will be used is also performed by COBRA functions.

COBRA Functions

COBRA is a MATLAB-based toolbox and thus the functions of this tool are all performed from within this environment. Since this tool does not support GUIs the methods have to be performed through scripts or single functions. This section presents some of the most used COBRA functions, the following are for performing constraint-based methods:

- Simulating optimal growth using FBA: growth is simulated by optimizing the model for flux through the model's biomass function or the other objective function:

  [solution] = optimizeCbModel(model, [osenseStr], [minNorm], [allowLoops])
  - *model* is the constraint-based model;
  - *osenseStr* can be either 'max' or 'min' to maximize or minimize the value of the objective;
  - *minNorm* has 0 as default, if nonzero, attempt to find a solution that minimizes the presence of loops;
  - *allowLoops* is true as default, if false;
  - *solution* is the solution structure and will be explained in 4.4.3.

- Geometric FBA attempts to return the minimal flux distribution central to the bounds of the solution space, while still maintaining optimal growth rate:

  flux = geometricFBA(model)
  - *model* is the constraint-based model;
  - *flux* contains the centered optimal flux distribution.

- OptKnock determine reaction sets to knock-out for the overproduction of a specific product when the model is optimized for internal cellular objectives:

  [OptKnockSol, biLevelMILPproblem] = OptKnock(model, selectedRxnList, [options], [constrOpt], [prevSolutions])
  - *selectedRxnList* a list of possible knockouts;
  - *options* is a complex structure that essentially consists in the desired product and the maximum number of deletions;
  - *constrOpt* is a complex structure that essentially consists in applying constraints on reactions, such as a minimal flux through the biomass function or ATP maintenance;
  - *prevSolutions* list of previous results.

- *OptKnockSol* a structure that will be explained in 4.4.4;
- *biLevelMILPproblem* is the bi-level MILP problem for the solution.

For doing changes in the model before performing the previous methods:

- Change reactions bounds:

  model = changeRxnBounds(model, rxnNameList, value, boundType);

  - *rxnNameList* a vector that corresponds to the reactions IDs;
  - *value* the numeric value of the bound;
  - *boundType* indicates which bound is changed, can be '*l*' for lower, '*u*' for upper and '*b*' for both.

- Change objective function of the model:

  model = changeObjective(model, rxnNameList, [objectiveCoeff]);

  - *rxnNameList* is a vector that corresponds to the reactions IDs;
  - *objectiveCoeff* specifies the weight given to the respective reaction in *rxnNameList.* when empty the coefficient is 1.

These are some of the most used COBRA functions in this work. The study of these two tools was vital for the development of the platform that integrates them and used both model and constraint-based methods.

Chapter 4

Results and Development

The main goal of this work was the design and development of an integrated platform for *in silico* phenotype simulation of microbial strains. This platform integrates the most relevant existing simulation methods, but also enables its easy extension with other methods that may be developed in the future.

This platform was built in Java and has as main feature the ability to connect with the MATLAB environment. This means that this platform can be a link between Java and different MATLAB tools and with that the access of multiple constraint-based methods in a single platform. Although there are several MATLAB tools, this work was mainly focused in the COBRA toolbox and in creating a bridge between this toolbox and OptFlux. The goal was to allow OptFlux to use the COBRA's functionalities and with that enlarge the set of available methods.

This chapter seeks to explain how the implementation of the connection between tools was performed and what the final result as an OptFlux's plugin.

4.1. A language-independent platform

Before the implementation of the platform, a layer of connection or conversion was idealized. This layer should be implemented so that any language and toolbox could be integrated in this Java platform. This implementation should aim for the information exchange between tools independently of the language in which it was built.

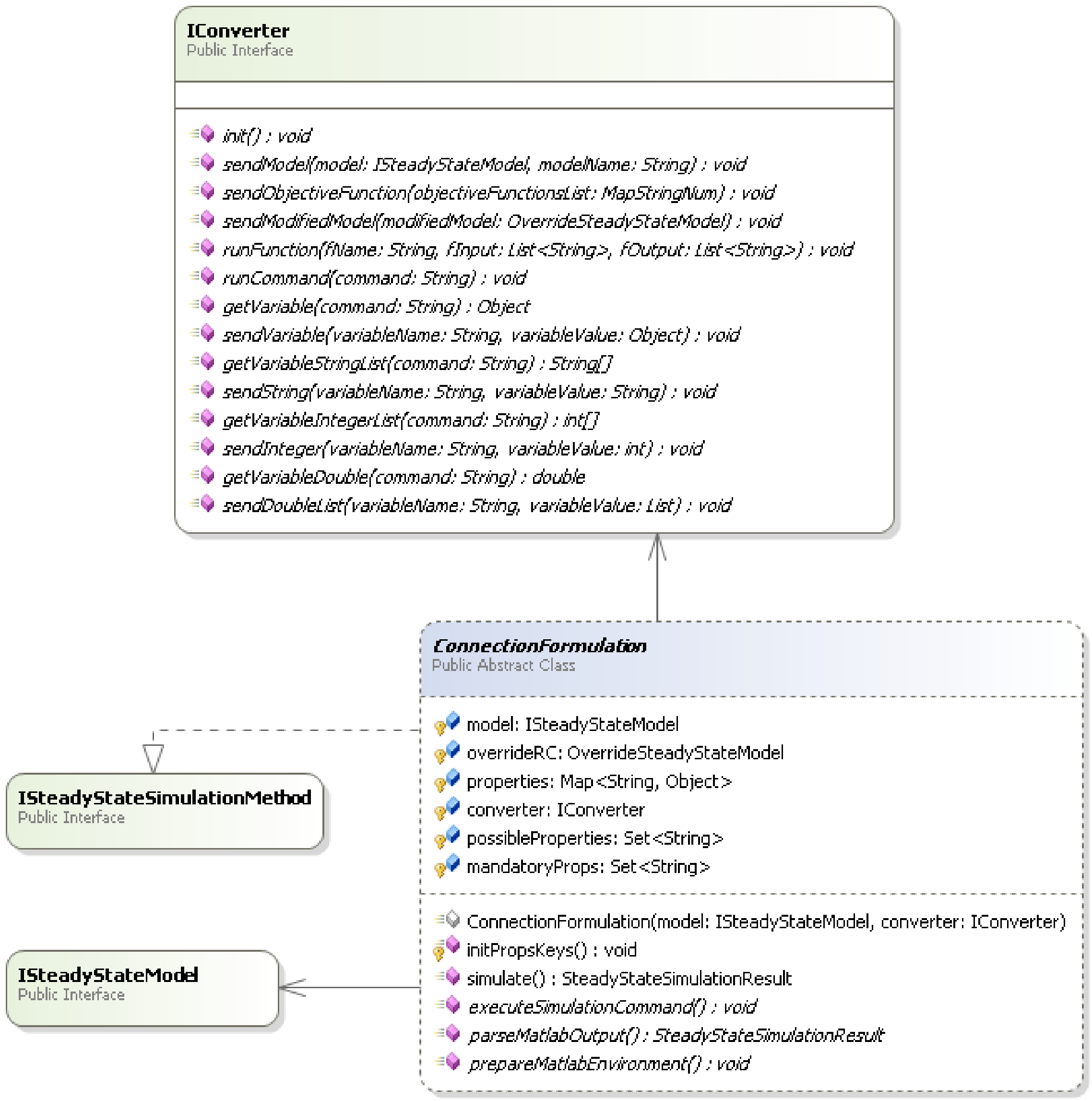

IConverter
Public Interface
init() : void
sendModel(model: ISteadyStateModel, modelName: String) : void
sendObjectiveFunction(objectiveFunctionsList: MapStringNum) : void
sendModifiedModel(modifiedModel: OverrideSteadyStateModel) : void
runFunction(fName: String, fInput: List<String>, fOutput: List<String>) : void
runCommand(command: String) : void
getVariable(command: String) : Object
sendVariable(variableName: String, variableValue: Object) : void
getVariableStringList(command: String) : String[]
sendString(variableName: String, variableValue: String) : void
getVariableIntegerList(command: String) : int[]
sendInteger(variableName: String, variableValue: int) : void
getVariableDouble(command: String) : double
sendDoubleList(variableName: String, variableValue: List) : void
ConnectionFormulation
Public Abstract Class
model: ISteadyStateModel
overrideRC: OverrideSteadyStateModel
properties: Map<String, Object>
converter: IConverter
possibleProperties: Set<String>
mandatoryProps: Set<String>
ConnectionFormulation(model: ISteadyStateModel, converter: IConverter)
initPropsKeys() : void
simulate() : SteadyStateSimulationResult
executeSimulationCommand() : void
parseMatlabOutput() : SteadyStateSimulationResult
prepareMatlabEnvironment() : void
ISteadyStateSimulationMethod
Public Interface
ISteadyStateModel
Public Interface

Figure 7 - Connection Layer Class Diagram

Figure 7 represents the idealized connection layer. On the center of this layer is the *IConverter* interface. This interface has a significant role in this layer and in the entire platform. Its methods are designed with the aim of being a link between this platform and the language in which a specific toolbox was developed. From the programming language side this interface requires the implementation of three categories of methods.

One category is for the methods that are used to initialize the computational environment and the toolbox, for instance, the definition of the toolbox solvers or other global parameters. The second category is a set of methods used for sending commands or functions to any language. It can be a single command or a function with input and output parameters. The last, but not least, category is for the methods used for variables' management. These methods are used to send and obtain generic and typed variables from the computational language. These typed variables are Strings, Integers, Doubles and arrays for each of these three types.

These types' choices was based on the fact that most of the tools that have constraint-based methods use these variables in their operations, also these variables exist in most of the computational languages.

With all these methods the platform can interact with any language, although for the constraint-based analysis toolboxes interaction another set of methods are required. These methods are related with the constraint-based model, this interface obliges the implementation of three methods related with the model: the method responsible for the creation of the model in the toolbox using the converter, this model is based on an *ISteadyStateModel*; the method that defines the objective functions; the one that creates an *OverrideSteadyStateModel*, this model is identical to a *ISteadyStateModel* but some reactions constraints will have different values.

With all this, the class that successfully implements this interface allows the platform to exchange commands between Java and a target language and also creates the constraint-based model in any toolbox. As for the performance of the toolbox operations another class must by implemented.

The *ConnectionFormulation* is an abstract class that is responsible for associating a converter according with the language and the toolbox with the formulations that extend this abstract. In this case the *ConnectionFormulation* is used for simulation methods and it implements the *ISteadyStateSimulationMethod* to be considered a simulation formulation by the Metabolic.

If another category of formulations was desired, for instance optimization formulation, then this abstract should be implemented based on that formulation and by using the converter it could access to all the language and toolbox operations. Since this class is used as example of how this platform can connect with any language and toolbox, in the following chapters it will be given a greater relevance and explanation of this class purpose.

4.2. Java-MATLAB integration

After the idealization of the converter structure that allows the connection with any language, as long as Java can connect, the next step of this platform implementation was the study of how a connection Java-MATLAB could be performed. This connection should allow a Java application to create a MATLAB session and also call MATLAB commands from a Java's application.

MATLAB already distributes a jar file called Java MATLAB Interface (JMI) that allows the access to the MATLAB environment. This interface has two major types of functions, send commands from Java into MATLAB and allow Java to access the data in the MATLAB workspace. Along with JMI is the MATLABControl [91], a Java API that simplifies all the JMI functionalities and allows using MATLAB through Java.

By implementing the MATLABControl in a Java application, it is possible to create a MATLAB session that is the bridge between both languages. This session allows accessing the MATLAB workspace, and sending and receiving variables within it. Also, this session can call any command or function supported by MATLAB. This connection is unidirectional, although both JMI and the MATLABControl allow a bidirectional communication, for this work the operations will all be sent by Java. This means that Java sends operations to MATLAB and waits for the end of these operations, on the other side MATLAB only executes those and will not send any variables or commands to the Java application, the application will be responsible for knowing what will be created in MATLAB and get all the needed information.

With this implementation in the platform created for this work the next objective became the development of a structure that allowed *Metabolic* to access some of the constraint-based methods present in the COBRA toolbox. As mentioned before, almost all of COBRA's operations need the model to be performed. This, the first objective was to convert the *Metabolic* model and send it to COBRA using the MATLABControl and then be able to perform the COBRA methods.

4.3. Convert Metabolic Model to Cobra Model

The conversion of the *Metabolic* model to the COBRA model was an essential step to the developed platform becoming capable of performing the COBRA methods. For this, it was crucial to analyze both models, the *ISteadyStateModel* structure of the *Metabolic* project and the model structure in COBRA.

Although both structures are slightly different, the information is exactly the same. The major difference between these structures is the way the data is obtained and manipulated.

As aforementioned, both tools can load the models recurring to different approaches. The construction of the model in our work is performed gathering the model information present in the *Metabolic's ISteadyStateModel*. With this approach there is no restriction to any of the existing loading approaches. However, the model can only be built if the *ISteadyStateModel* has all the needed information using one of the loading approaches present in the *Metabolic* project.

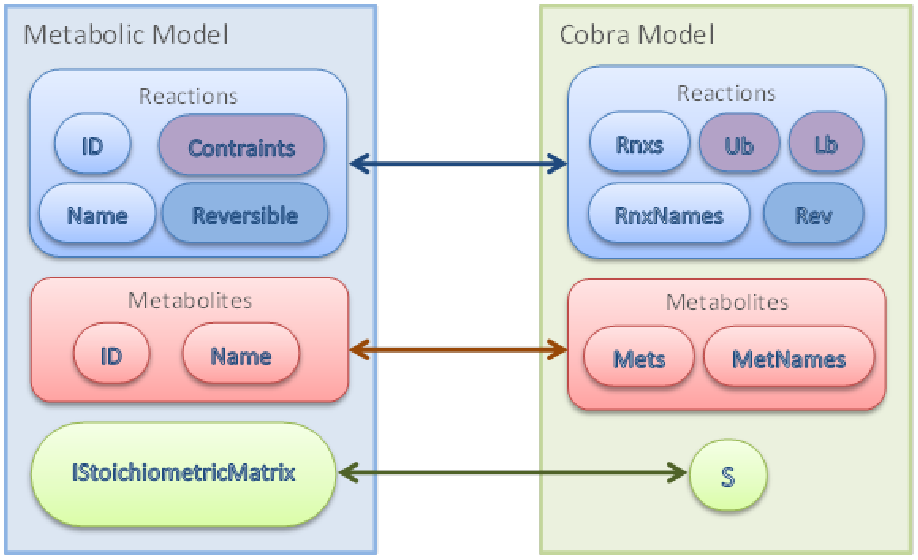


Figure 8 - Comparison between Metabolic and COBRA Models

Figure 8 represents the information that each of the tools structures has in common. Recurring to the MATLABControl class and sending MATLAB basic functionalities used for creating variables it is possible to create any structure in MATLAB. Also, by gathering all the information presented in the *Metabolic's ISteadyStateModel* it became possible to create a constraint-based model in MATLAB that corresponds to the COBRA's model. This model must contain the reactions information, such as the name and the associated constraints, the metabolites information and the stoichiometric matrix. With this, the platform can create a COBRA model which has the same information as the model in the *Metabolic's* project and be used along with the constraint-based methods present in COBRA.

4.4. COBRA methods from Java

After the development of a Java application that could build a COBRA model, a requirement's analysis of all the available COBRA methods was necessary. Then, a selection was made of which methods could be implemented in the platform. This evaluation was accomplished by categorizing the methods by simulation, optimization and analysis group. This categorization allowed defining tangible objectives by giving priority to the implementation of simulation methods.

After the decision on the implementation of simulation methods, this was accomplished by taking into consideration an essential parameter: the method's result having, as minimum requirements, a flux distribution and thus enabling the construction of a *Metabolic's SteadyStateSimulationResult* from the results of the COBRA method.

The importance of this parameter was due to the fact that all the simulation methods implemented in *Metabolic* have the *SteadyStateSimulationResult* structure as a result. For that reason, the new platform was developed by using the same structure as the *Metabolic* for the simulation formulations.

From the analysis of the COBRA methods four were selected to be implemented in the platform: FBA, GeometricFBA, LinearMOMA and MOMA. This choice was made based on the fact that these methods are widely used and have as a result the information capable of creating a *SteadyStateSimulationResult* structure making the implementation in the platform possible.

Although this structure has a significant importance in the platform and the *Metabolic* project integration, another requirement is necessary from *Metabolic* and had some impact in these methods selection. In order for the *Metabolic's* simulation methods to be integrated in OptFlux all must be an *ISteadyStateSimulationMethod* and implement all methods from this interface. For this reason, all COBRA's methods implemented in this platform must also be able to answer to this interface's methods with the aim of being integrated in OptFlux.

With all these considerations the platform was created with two layers: the layer explained in the previous chapter, the converter layer, the one responsible for converting Java information in another language and vice-versa and the layer that ensures that all the formulations have the needed information for the *Metabolic* integration.

4.4.1. Converter Layer

The essential part of this layer was already explained in a previous chapter, but this explanation is now based on a practical example, the use of the *IConverter* as a connection structure Java-MATLAB.

As mentioned before, this layer was developed with the aim of connecting Java with another language and converting the information between both. For this work, this layer and platform only have a Java-MATLAB connection (Figure 9) but as explained a solid structure is already built for the integration with other languages, for instance Python.

Also as stated, all this integration is possible through the *IConverter* interface, in this work the Java-MATLAB interaction is implemented in the class *CobraMATLABConnection*. This class has the obligation of implementing all the interface methods used in the Java-MATLAB connection. To perform this, it uses the MATLABControl mentioned before and with it creates an instance of a MATLABProxy that corresponds to a MATLAB session and uses it in the implementation of all the methods that send commands to MATLAB and manage the variables in the workspace.

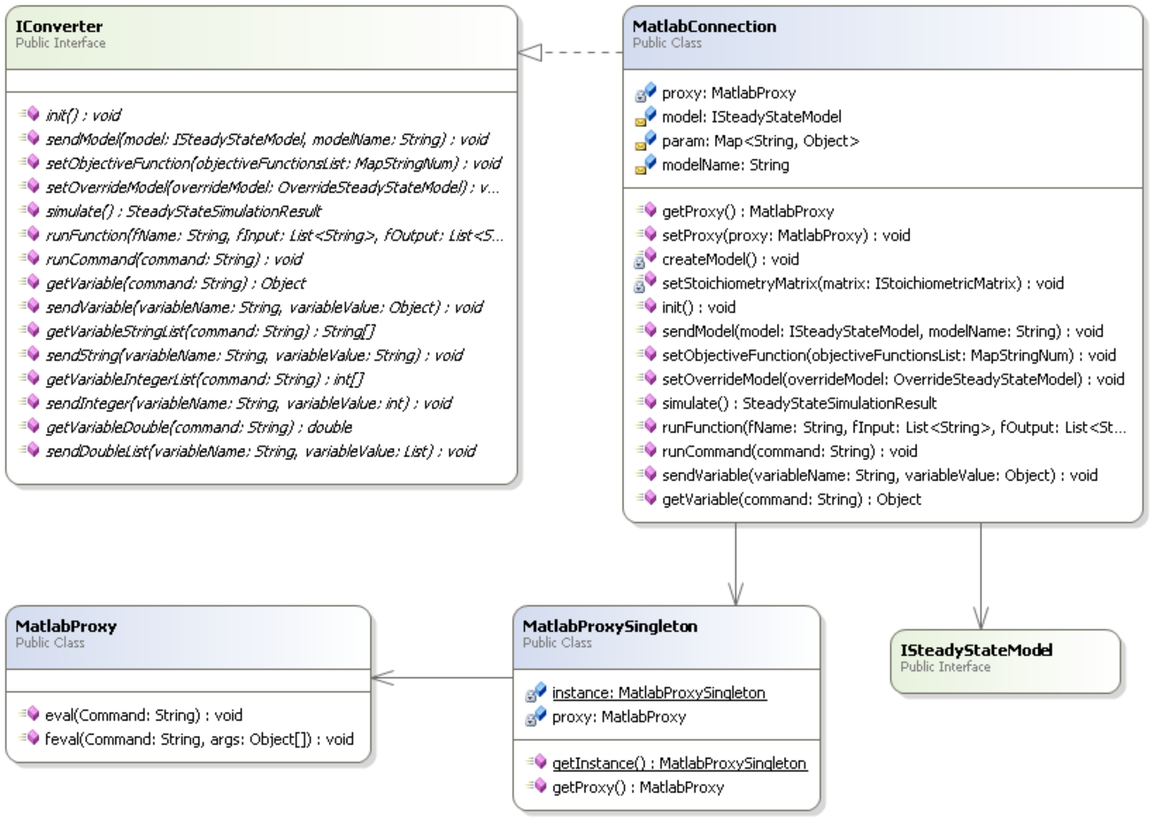

IConverter
Public Interface
init() : void
sendModel(model: ISteadyStateModel, modelName: String) : void
setObjectiveFunction(objectiveFunctionsList: MapStringNum) : void
setOverrideModel(overrideModel: OverrideSteadyStateModel) : v...
simulate() : SteadyStateSimulationResult
runFunction(fName: String, fInput: List<String>, fOutput: List<S...
runCommand(command: String) : void
getVariable(command: String) : Object
sendVariable(variableName: String, variableValue: Object) : void
getVariableStringList(command: String) : String[]
sendString(variableName: String, variableValue: String) : void
getVariableIntegerList(command: String) : int[]
sendInteger(variableName: String, variableValue: int) : void
getVariableDouble(command: String) : double
sendDoubleList(variableName: String, variableValue: List) : void
MatlabConnection
Public Class
proxy: MatlabProxy
model: ISteadyStateModel
param: Map<String, Object>
modelName: String
getProxy() : MatlabProxy
setProxy(proxy: MatlabProxy) : void
createModel() : void
setStoichiometryMatrix(matrix: IStoichiometricMatrix) : void
init() : void
sendModel(model: ISteadyStateModel, modelName: String) : void
setObjectiveFunction(objectiveFunctionsList: MapStringNum) : void
setOverrideModel(overrideModel: OverrideSteadyStateModel) : void
simulate() : SteadyStateSimulationResult
runFunction(fName: String, fInput: List<String>, fOutput: List<St...
runCommand(command: String) : void
sendVariable(variableName: String, variableValue: Object) : void
getVariable(command: String) : Object
MatlabProxy
Public Class
eval(Command: String) : void
feval(Command: String, args: Object[]) : void
MatlabProxySingleton
Public Class
instance: MatlabProxySingleton
proxy: MatlabProxy
getInstance() : MatlabProxySingleton
getProxy() : MatlabProxy
ISteadyStateModel
Public Interface

Figure 9 Java-MATLAB Connection Class Diagram

Along with the Java-MATLAB connection this class is also responsible for the management of the COBRA's model. This class creates the Cobra's model by gathering the needed information present in the *ISteadyStateModel* and it also defines the objective function recurring to the correspondent Cobra's method. The modification of the reactions' constraints of the model is also performed by this class and also recurring to Cobra's methods. As for the implementation of the formulations, in this case the COBRA's formulations, this task is responsibility of the other layer of this platform.

4.4.2. Formulations Layer

In this layer all the formulations will be integrated in the platform and will be dependent of the language and toolbox. It is in this layer that the previous presented *ConnectionFormulation* is implemented.

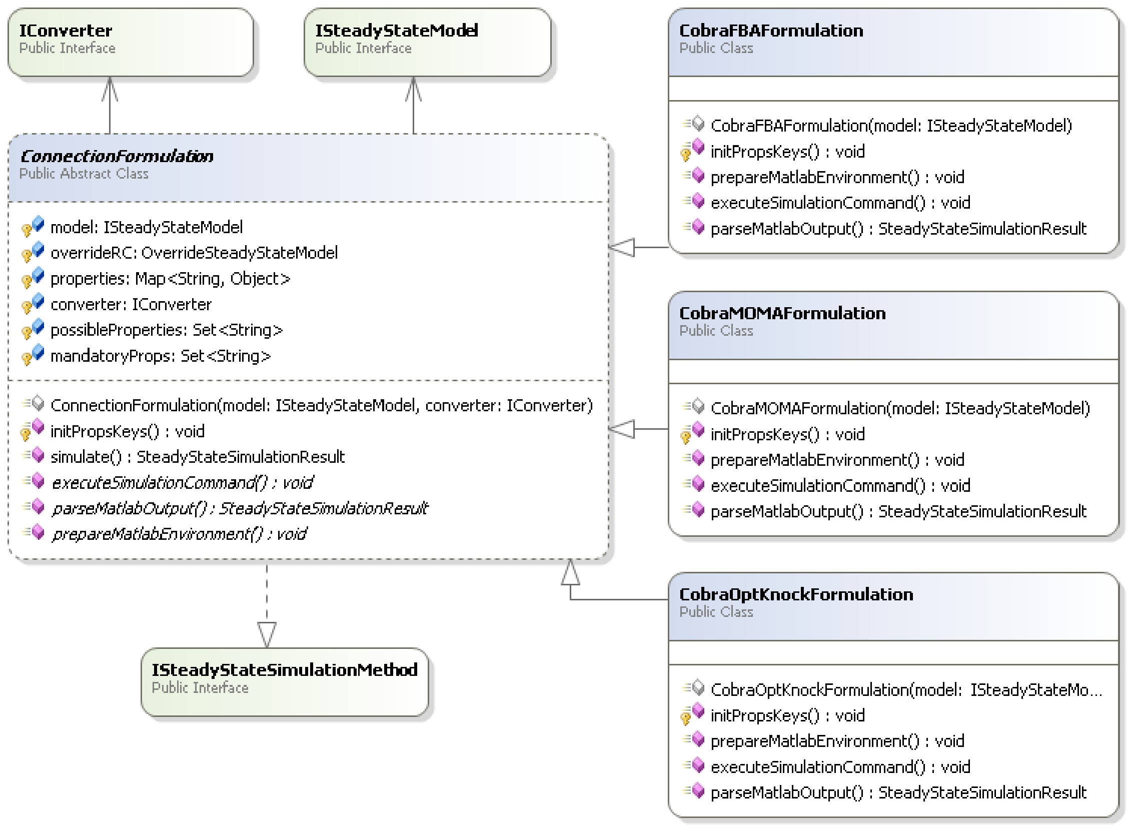

IConverter
Public Interface
ISteadyStateModel
Public Interface
ConnectionFormulation
Public Abstract Class
model: ISteadyStateModel
overrideRC: OverrideSteadyStateModel
properties: Map<String, Object>
converter: IConverter
possibleProperties: Set<String>
mandatoryProps: Set<String>
ConnectionFormulation(model: ISteadyStateModel, converter: IConverter)
initPropsKeys() : void
simulate() : SteadyStateSimulationResult
executeSimulationCommand() : void
parseMatlabOutput() : SteadyStateSimulationResult
prepareMatlabEnvironment() : void
ISteadyStateSimulationMethod
Public Interface
CobraFBAFormulation
Public Class
CobraFBAFormulation(model: ISteadyStateModel)
initPropsKeys() : void
prepareMatlabEnvironment() : void
executeSimulationCommand() : void
parseMatlabOutput() : SteadyStateSimulationResult
CobraMOMAFormulation
Public Class
CobraMOMAFormulation(model: ISteadyStateModel)
initPropsKeys() : void
prepareMatlabEnvironment() : void
executeSimulationCommand() : void
parseMatlabOutput() : SteadyStateSimulationResult
CobraOptKnockFormulation
Public Class
CobraOptKnockFormulation(model: ISteadyStateMo...
initPropsKeys() : void
prepareMatlabEnvironment() : void
executeSimulationCommand() : void
parseMatlabOutput() : SteadyStateSimulationResult

Figure 10 Formulation Layer Class Diagram

After the development of a class that connects Java with MATLAB, more precisely Metabolic with Cobra, the next step was the integration of the Cobra formulations in the platform (Figure 10). As stated before, all the formulations that integrate the *Metabolic* need to implement the *ISteadyStateSimulationMethod* interface. The Cobra's formulations are no exception and this mandatory implementation is present in an abstract class named *ConnectionFormulation*. . This class is responsible for associating a converter according with the language and toolbox with the formulations that extend this abstract. This abstract class does not know the language in which the formulations are built, this class only uses the converter to perform operations in a toolbox. On the other hand, this abstract and the classes that extend it must know the structures of the formulations.

In this case, the used converter is the *CobraMATLABConnection* converter implemented in the previous layer. Also, this class is responsible for the registry of properties that will be used and are common for all the formulations. The properties in this class are the global parameters that the toolbox needs to use the formulations. Also, each formulation must register their own properties based on each method parameters. For instance, the COBRA's FBA needs to register if the objective function will maximize or minimize. The global properties can be the genetic conditions and/or the environmental conditions, with these the abstract will change the model according with the registered conditions.

As for the simulation, the *ConnectionFormulation* defines three abstract methods that each extension class must implement in to perform the formulations and create the *SteadyStateSimulationResult*:

- ***prepareMATLABEnvironment***, a method that prepares the toolbox environment, is used to create all the variables that the implemented formulation will need to be performed;
- ***executeSimulationCommand***, a method that will perform the toolbox formulation;
- ***parseMATLABOutput***, a method that based on the formulation solution and the formulation itself creates the *SteadyStateSimulationResult* structure.

With this, all the formulations that extend this abstract class will be considered an *ISteadyStateSimulationMethod* recognized by the *Metabolic* project and consequently by OptFlux. The next step was the implementation of the selected COBRA's formulations.

4.4.3. COBRA's formulations implementations

To demonstrate this platform some of the COBRA's formulations were integrated in it. To accomplish this it was needed to analyze the methods based on the results and also in the input information that each method demands to be performed. Next, the implemented methods will be presented and their input parameters will be shown and how the result was engaged with the *SteadyStateSimulationResult* structure.

FBA

The first formulation to be integrated in the platform was the COBRA's FBA and this implementation originated the *CobraFBASimulation* class. This class uses the COBRA's method *optimizeCbModel* and only needs the constraint-based model to be performed. As optional parameter, the method takes a flag representing if the objective function is to be maximized or minimized (if this value is not defined then the objective is maximized). This parameter is registered in the properties of the simulation.

This COBRA's method constructs a LP problem and returns a solution structure that will be engaged in a *SteadyStateSimulationResult* structure. The following elements represent the solution structure of this formulation:

- ***f,*** the objective value;
- ***x,*** an unique column that has the same number of rows as the model's reactions and represents the flux distribution of the simulation;
- ***stat***, the solution status, being, for instance, 1 for optimal and 0 for infeasible;
- ***w***, a unique column that has the same number of rows as the model's reaction and represents the reduced costs (values that indicate how much each reaction affects the objective [92]).
- ***y***, a unique column that has the same number of rows as the model's metabolites and represents the shadow prices (values for each metabolite that indicate how much the addition of the corresponding metabolite will increase or decrease the objective value [92]).
- ***solver***, the name of the solver used;
- ***time***, the time taken for the task.

Figure 11 represents the result of this formulation and what fields were used to build the *SteadyStateSimulationResult*.

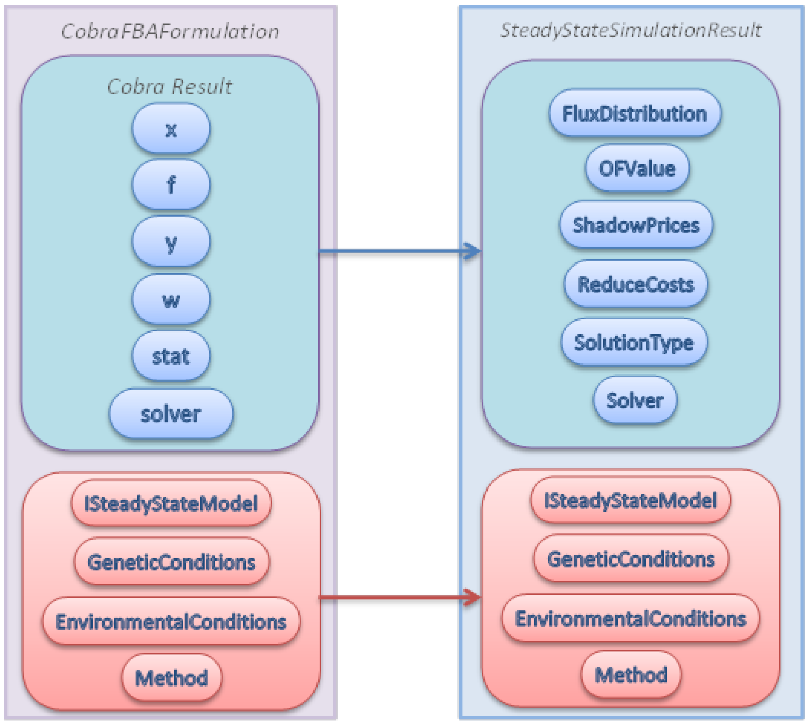


Figure 11 - Construction of COBRA's FBA result

There are some fields in the *SteadyStateSimulationResult* structure that cannot be obtained from the formulation result. Nevertheless, some of these fields are already known by the class. For this specific formulation these fields are: the *ISteadyStateModel*, the *GeneticConditions* and *EnvironmentalConditions* structures, obtained from the formulation properties and the method used, in this formulation is the Cobra FBA.

Geometric FBA

Another formulation that was integrated in the platform was the GeometricFBA and it originated the *CobraGeometricFBA* class. This formulation is an alternative to standard FBA and has the constraint-based model as essential field. As for its solution result structure is only a unique column with the same number of rows as the model's reactions and represents the centered optimal flux distribution. Although this solution has only one field, it is enough to construct the *SteadyStateSimulationResult* structure. The following Figure 12 represents this construction.

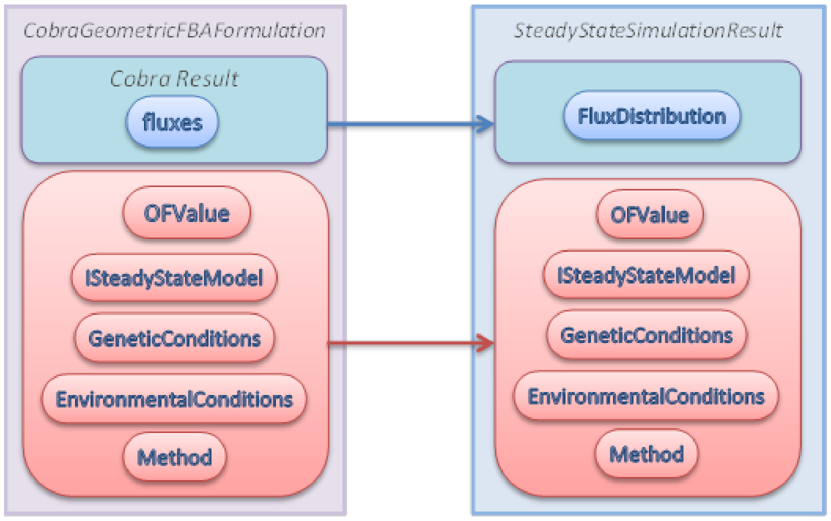


Figure 12 - Construction of COBRA's GeometricFBA result

This centered optimal flux distribution field will be the flux distribution of this structure, as for the other fields, they will be specified by the default properties of the formulation. These fields are:

- The *ISteadyStateModel*, already known by the class;
- The objective function value, will be the value of the objective reaction present in the flux distribution;
- The *GeneticConditions* structure, obtained from the formulation properties;
- The *EnvironmentalConditions* structure, obtained from the formulation properties;
- The method used, will be Cobra GeometricFBA;
- The result structure will not have the solver definition, status of the solution nor reaction/metabolites complementary information.

MOMA and Linear MOMA

The final simulation formulations to be integrated in this platform were the MOMA formulation and its linear version, the LinearMOMA. These formulations have exactly the same input parameters and the same solution result and for this reason they will be presented together. These formulations have as mandatory input parameters two constraint-based models and as optional parameter the maximization/minimization flag as used in FBA.

The two mandatory models requested in these formulations are, normally, a wild-type and a mutant model. While the mutant model is previously created in MATLAB, by the *ConnectionFormulation*, based on the genetic and environmental conditions registered in the properties the wild-type model is created in the class that implements this formulation, the *CobraMOMAFormulation* class. Here the model is sent to MATLAB without the environmental/genetic changes and thus is considered a wild-type model.

The solution resulting from these formulations has the same structure and consists in four elements:

- ***solutionWT***, a structure that is the result of the FBA simulation using the wild-type model, this structure is identical to the FBA result;
- ***solutionDel***, a structure that is the result of the FBA simulation using the mutant model, this structure is identical to the FBA result but without the ***y*** and ***w*** fields;
- ***solStatus***, a numeric field that represents the status of the solution;
- ***totalFluxDiff***, a numeric field that represents the MOMA objective, the minimum distance between the flux distributions.

In these formulations, the construction of the *SteadyStateSimulationResult* structure did not need all the information present in these elements. Figure 13 shows the information used to populate this structure.

As in the previously presented formulations, there is information that the MATLAB solution does not return and is needed by the *SteadyStateSimulationResult*. As both LinearMOMA and MOMA formulations have the same solution the addition of this extra information is the same. Each formulation populates this final structure with: the *ISteadyStateModel*, the *GeneticConditions* and *EnvironmentalConditions* structures, obtained from the formulation properties and finally the method used, LinearMOMA or MOMA depending on the formulation.

With these formulations implemented in the platform, the *Metabolic* project earned four new simulation methods and more importantly it also gained a connection with MATLAB and with this opened a door to future method implementations and toolbox connections.

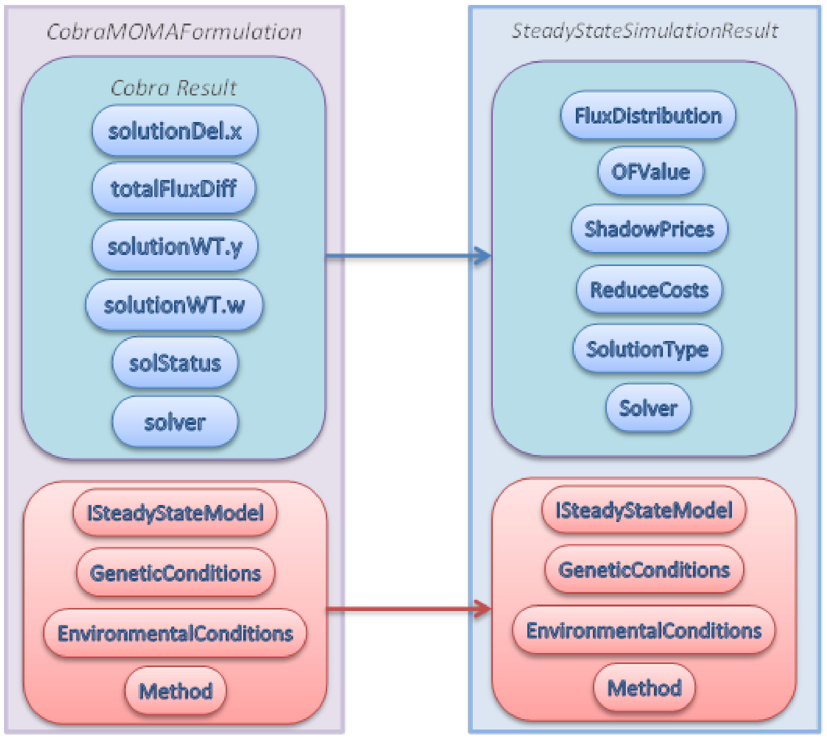


Figure 13- Construction of COBRA's LinerMOMA/MOMA result

Just like the other simulation formulations, these COBRA's methods are registered in the simulation control center presented previously. This registration enables the *Metabolic* to use these formulations the same way it uses the ones already implemented.

4.4.4. Other COBRA's formulation implementations

As mentioned, the main objective of this work was the implementation of simulation methods in an integrated platform. However, by the time the COBRA's methods were studied, other methods that by definition are not considered simulation methods were also implemented. These methods are the OptKnock and GDLS methods, both considered optimization methods.

The *Metabolic* project also supports optimization methods, for this reason the implementation of these two methods became an objective. Nevertheless, the optimization in *Metabolic* is not at the same development level neither is so easy to integrate new methods as for the simulation. While for the simulation, there is a solid structure and new simulation methods can be easily implemented as long as the final result can be created, on the other hand the optimization structure is more restrictive to the implementation of new methods given the specific nature of the strain optimization methods currently implemented.

The first problem is focused in the creation of the problem. There is not a generic structure for all the optimization methods, such as the *ISteadyStateSimulationMethod* interface for simulations. Without a similar interface for the optimization that oblige the centralization of the problem creation, the implementation of new methods has to be from scratch and each implementation can be very distinct although all being optimization formulations. The other problem is focused in the result of the optimization. In the simulation case all the formulations were able to return solutions that could be used to create the *SteadyStateSimulationResult*, in the optimization case the result of each formulation can be very different and for that reason the *Metabolic* project does not have a singular structure that can be used for all formulation results.

For this work the implementation of this centralized structure was not performed mostly due to the fact that the *Metabolic* already has a different structure for every optimization formulation and solution and the implementation of a structure similar to the simulation was practically impossible in the time available for this project.

Given the nature of the OptKnock and GDLS method, on the other hand, their implementation was possible by performing an adaptation of the result in order to be fitted for the *SteadyStateSimulationResult* structure.

OptKnock

The OptKnock implementation in the platform was possible by adapting the formulation and the result into simulation formulations. This adaptation had two phases, one for the creation of the problem and the other for the creation of the *SteadyStateSimulationResult* structure using the solution of the optimization.

Similar to the simulation formulation implemented in the platform, this formulation class is an extension of the *ConnectionFormulation* and an implementation of *ISteadyStateSimulationMethod*. With this, the implemented class, the *CobraOptKnockFormulation*, despite being an optimization is considered by the *Metabolic* project as a simulation. For this, the formulation parameters and result were converted in a simulation structure, this was performed by the abstract methods that every formulation must implement. The first step was the conversion of the parameters, comparing with the previous formulations, OptKnock has very distinct input parameters. The mandatory parameters to perform this formulation are:

- ***model***, the constraint-based model, used by all formulations;
- ***selectedRxnList***, a list of the reactions that can be knocked-out;
- ***targetRxn***, the target reaction that will be maximized;
- ***numDel***, the number of maximum deletions allowed in the optimization;
- ***rxnList, values, sense***, a list of reactions with constraint values and sense. This list represents the constraint reactions, for instance, for the optimization result to be valid the biomass or other reaction must have a specific minimum value.

All these parameters must be previously registered in the parameters properties that all formulations have and then sent to the MATLAB environment to be used with the MATLAB's OptKnock method. As for the result, this formulation returns a structure that has as essential simulation fields: the objective value, the flux distribution and the reactions that must be knocked-out in the model to result in this flux distribution.

Figure 14 presents the fields from the OptKnock solution and how they were converted in the simulation structure *SteadyStateSimulationResult*. Similar to the simulation formulations, there are fields in the final structure that are filled by the information obtained from the MATLAB result and others that are filled by the class itself. Also, there are fields that can be empty, in this formulation these fields are the reaction complementary information and the metabolite complementary information. The fields that are filled by the formulation class are the model used, the environmental conditions, and the method used, in this case the Cobra's OptKnock. As for the fields filled by the MATLAB result they are the objective function value, the flux distribution, the solution type, the solver used and the list of reactions to be knocked-out are placed in the genetic conditions field. By doing this it was possible to integrate an optimization method in the simulation structure without mislaying information and biological sense.

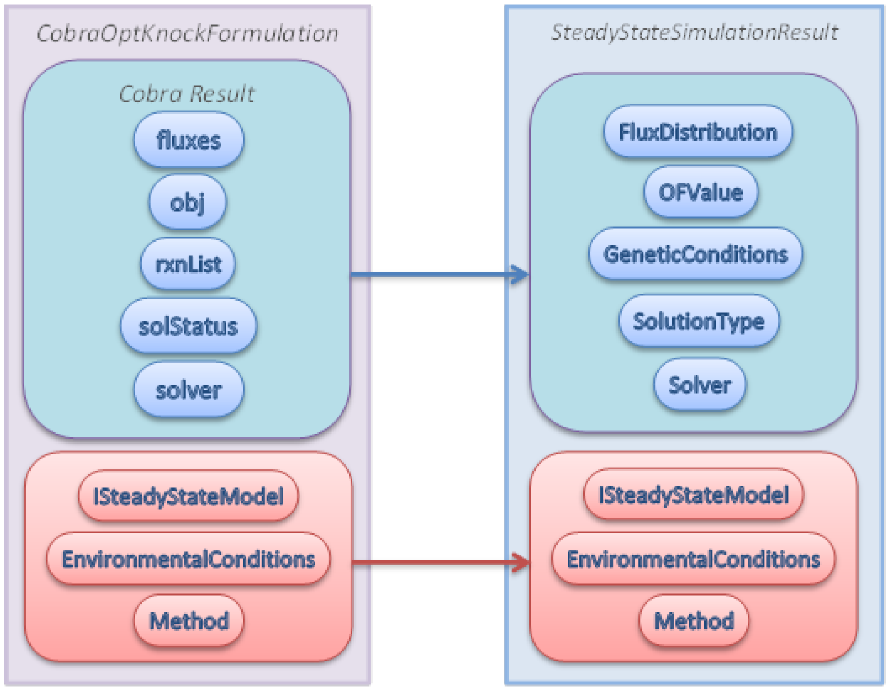


Figure 14- Construction of COBRA's OptKnock result

GDLS

The implementation of the GDLS formulation was based in the same principles as the OptKnock formulation. The implemented *CobraGDLSFormulation* class is also an extension of the *ConnectionFormultation* and thus an implementation of the *ISteadyStateSimulationMethod*. Similar to the OptKnock this formulation has mandatory parameters that are registered in the formulation properties and will be sent to the MATLAB environmental. These parameters are:

- ***model***, the constraint-based model;
- ***nbhdsz***, a value that corresponds to the neighborhood size;
- ***M***, a value corresponds to the number of search paths;
- ***maxKO***, the number of maximum deletions allowed in the optimization;
- ***selectedRxn***, a list of the reactions that can be knocked-out;
- ***targetRxn***, the reaction/product that will be maximized;
- ***minGrowth***, a value that corresponds to the minimum value that the biomass must have in the optimization result.

As for the result, the GDLS formulation returns a slightly different solution compared with the OptKnock. The result consists in two important structures that will be used to fill the *SteadyStateSimulationResult* structure. One of the structures consists in the biomass value, the maximum and minimum value of the target product/reaction and the reactions that must be knocked-out in the model as to result this values. The other structure is a set of smaller structures that consists in three simulation results: one with the maximization of the biomass as objective function and the other two with the maximization and minimization of the target reaction as objective function.

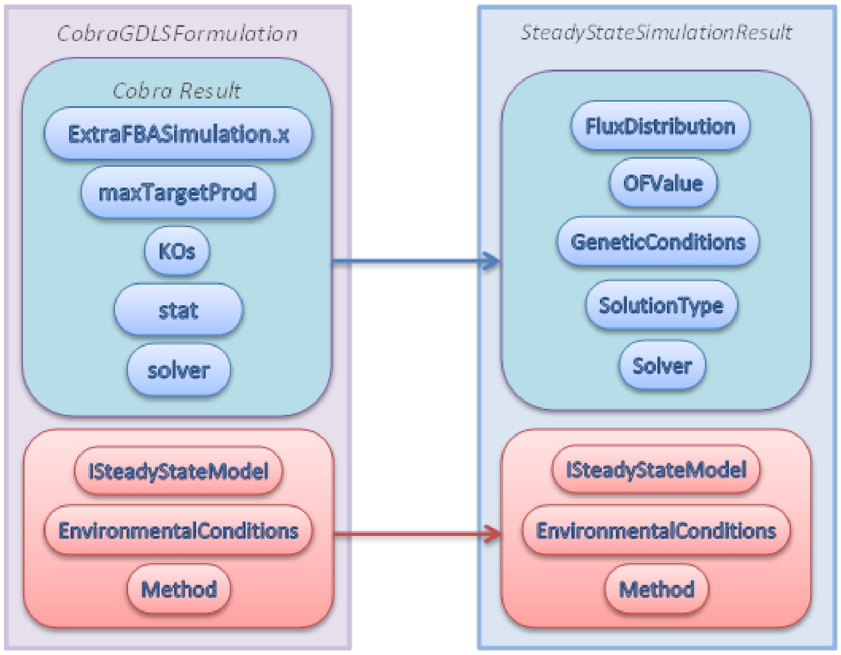


Figure 15 - Construction of COBRA's GDLS result

Figure 15 represents the fields from the GDLS solution and how they were converted in the simulation structure *SteadyStateSimulationResult*. This conversion was similar to the one accomplished for the OptKnock formulation, however the GDLS solution has some important information missing, the flux distribution. Despite the existence of a structure with three simulation results, none has the flux distributions needed for the *SteadyStateSimulationResult*. Although, this fact does not preclude achieving a flux distribution by performing a FBA simulation with the conditions present in the GDLS solution for the maximum value of the target product/reaction. By doing so the flux distribution was obtained from that simulation along with the reduced costs and shadow prices (complementary information). The objective value is the maximum value of the target product, the genetic conditions are handled similarly to the OptKnock case, by using the reactions to be knocked-out, the solver and solution type are obtained by one of the three simulation results existent in the GDLS solution. As for the model, method and environmental conditions fields the class has the responsibility to fill them.

Optimization Control Center implementation

With these implementations, the platform gained two new COBRA's methods that although are considered optimization methods could be, with some special arrangements, implemented in the simulation group. Maybe in the future, with the enhancement of this platform, these methods will be relocated in an optimization group along with other similar formulations.

Despite these formulations were, in some way, converted into simulations a new control center was created to use these formulations in the *Metabolic* project. An optimization control center was created so as not to blend the simulation formulations in the *Metabolic* and these two formulations that, as mentioned, despite being converted in simulation are in a different category.

This new control center is responsible for performing the optimizations and managing their properties. Along with the control center also a factory for the optimization formulation was created and this factory is responsible for registering and removing formulations. This control center will be essential by the time the new plugin will use these two formulations.

Although these two components were developed for these two new formulations, in the future the optimization formulations that exist in *Metabolic* will also be managed by this optimization control center.

4.4.5. Platform enrichment

This platform is well documented and any developer that has some knowledge in Java can integrate new formulations. At this point, the platform only has a Java-MATLAB connection and as for formulations and toolboxes only COBRA's formulations are integrated, although through this API documentation it is easy to integrate new formulations and also new toolboxes in the platform. As for toolboxes in MATLAB similar to COBRA, the integration will use most of the present structure and the implementation will be more focused in the development of new classes that deal directly with the formulations. On the other hand, if a new language connection is the objective it will be necessary to implement more structures, but using this platform and documentation as base the integration will be easier than if it was made from scratch.

4.5. Integration with OptFlux

Since this platform was developed using the *Metabolic* project as a basis, an integration with the OptFlux tool is an expected goal. In this section, the integration of this platform with OptFlux will be explained and also a new plugin will be presented.

4.5.1. The OptFlux tool

As previously stated OptFlux is a free and open source ME software, allowing any user to use it and be able to contribute with new features and improvements. Being modular, allows OptFlux to increment features through plugin integration. This means that OptFlux is nothing more than a GUI that gives to the user a way to use these plugins functionalities. The *BioComponents* and the *Metabolic* projects previously presented are a fine example of this integration. With them, OptFlux offers operations to visualize, import and export GSMMs, including reactions, metabolites, equations and gene-reaction associations if available. It also allows the use of GSMMs for phenotype simulation of both wild-type and mutant organisms. Operations related with analysis and optimization or even metabolic network visualization framework are present in OptFlux also as plugins.

OptFlux allows the increment of plugins through the use of AIBench framework. AIBench is a lightweight, non-intrusive, Model-View-Controller–based (MVC) Java application framework that eases the connection, execution and integration of operations with well-defined input/output. By being MVC-based this framework as three main concepts:

- **Datatypes**: represent the data of the application. These datatypes can be simple or combinations of multiple datatypes. Users can manage objects, instances of the different datatypes, through the use of a hierarchical clipboard.
- **Views**: represent the visualization of the datatypes. Each datatype can have one or more views, represented in different tabs.
- **Operations**: available actions that allow the creation of instances of the datatypes. Operations are transformations that have a set of arguments as input, and a set of output objects. Operation's user interface can be automatically generated by AIBench or defined by the developer.

This plugin based facilitates reusing and integrating new functionalities in the tool. For this reason the platform explained in this work was integrated in OptFlux and with this it allows the tool to perform some of the COBRA's formulations previously explained in this chapter. With this integration a new plugin was created, the Cobra4Optflux3 plugin.

4.5.2. A COBRA-MATLAB plugin for OptFlux

This plugin consists in a platform that connects Java with MATLAB, more precisely with COBRA. With this plugin, OptFlux expanded the present formulations with some present in COBRA. This was possible because the formulation layer of the developed platform was built based on a generic formulation as explained in 4.4.2 and thus it is capable of performing simulations and optimizations using a constraint-based model. This also allowed an easy integration of this plugin in OptFlux. Same as the platform, this plugin has the simulation and optimization has the two major operations.

COBRA simulations in OptFlux

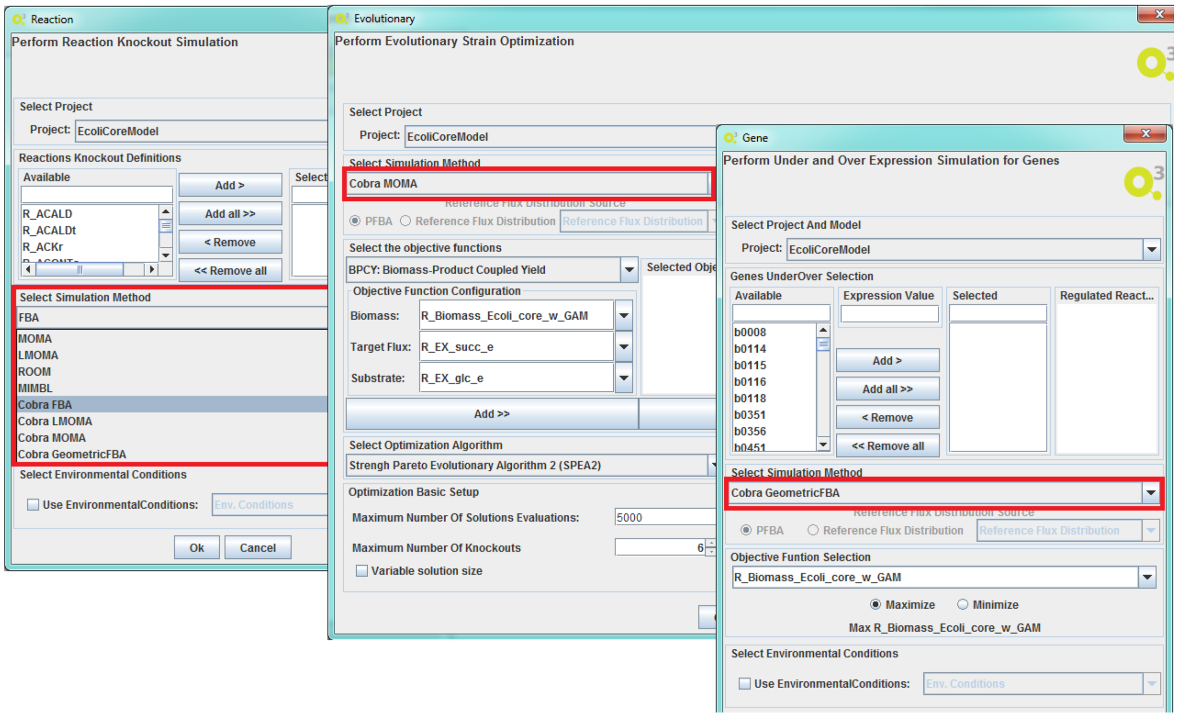
OptFlux already performs simulations using a model, for instance, FBA, MOMA, ROOM. These formulations are performed by the integration of Metabolic in OptFlux and by using a simulation control center that registers all formulations and makes them available for the tool to perform them.

With this new plugin the simulations that are implemented from COBRA will be registered in the same simulation control center by the time OptFlux loads the plugin. This is possible since the COBRA's formulations have the same structure as the *Metabolic* simulations formulations and for that reason the integration with the control center is the same for both. In other words, the structure of both Metabolic and COBRA's simulations formulations are identical and for that the control center handles them as equals.

This registry in the control center it is the only task the plugin have to fulfil to allow OptFlux to perform the COBRA's simulations. This registry will increase the simulations methods that OptFlux can perform and the COBRA's simulation methods will always be present along with the *Metabolic's* simulation methods.

The simulation operations in OptFlux are always performed through the simulation control center and the GUIs that perform simulation recur to the control center to present to the user the simulations that OptFlux is capable to perform.

Figure 16 represents three distinct GUIs in OptFlux that use the simulation control center. These GUI are used to perform reaction knockout simulations, strain optimization with evolutionary algorithms and under/over expression gene simulation. These GUIs are examples of the many operations that use the simulations formulations in OptFlux. In these operations, a simulation formulation is selected and performed given certain parameters. This list of formulations is filled from the simulations registered in the simulation control center and the GUI parameters are registered in the control center properties.


Perform Reaction Knockout Simulation

Project: EcoliCoreModel

Select Simulation Method: FBA, MOMA, LMOMA, ROOM, MIMBL, Cobra FBA, Cobra LMOMA, Cobra MOMA, Cobra GeometricFBA

Use EnvironmentalConditions: Env. Conditions

Perform Evolutionary Strain Optimization

Project: EcoliCoreModel

Select Simulation Method: Cobra MOMA

Select the objective functions: BPCY: Biomass-Product Coupled Yield

Biomass: R_Biomass_Ecoli_core_w_GAM

Target Flux: R_EX_succ_e

Substrate: R_EX_glc_e

Select Optimization Algorithm: Strength Pareto Evolutionary Algorithm 2 (SPEA2)

Maximum Number Of Solutions Evaluations: 5000

Maximum Number Of Knockouts: 6

Perform Under and Over Expression Simulation for Genes

Project: EcoliCoreModel

Select Simulation Method: Cobra GeometricFBA

Objective Funtion Selection: R_Biomass_Ecoli_core_w_GAM

[✓ Maximize] Minimize

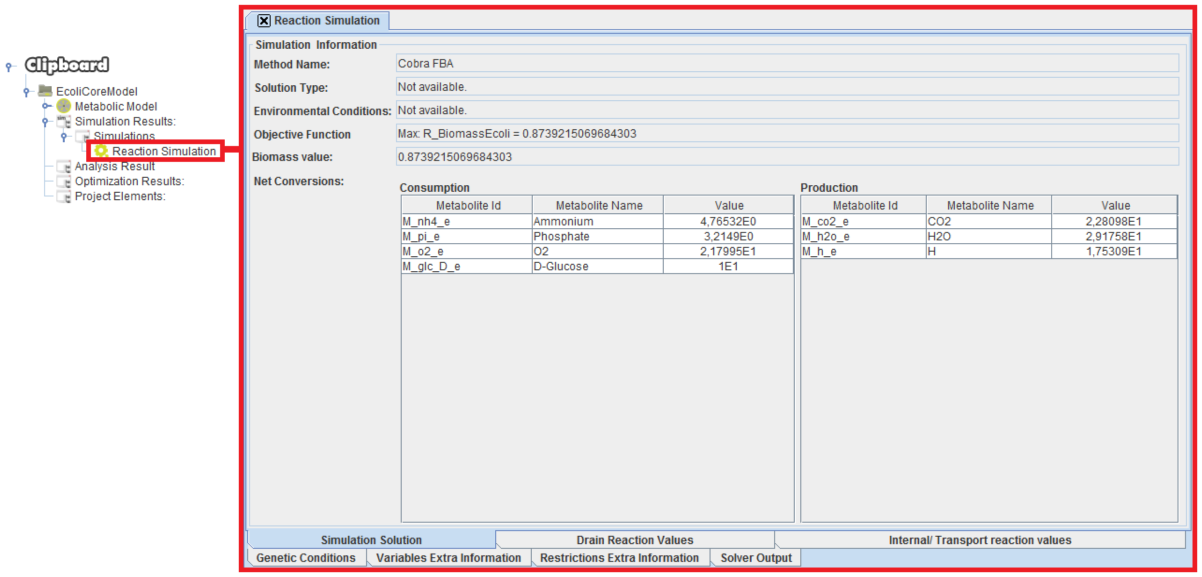
Max R_Biomass_Ecoli_core_w_GAM

Use EnvironmentalConditions: Env. Conditions

Figure 16 - OptFlux GUIs with simulations

Being the COBRA formulations registered in the same control center by this plugin, these formulations will also be available in Optflux for all these operations. As Figure 16 shows, every GUI where these simulation methods can be selected, the COBRA formulations will also be present and can be performed.

The reaction knockout simulations' result is represented in the Figure 17. Since all the COBRA's simulations formulations create the same structure as the *Metabolic* original methods there was no need for the new plugin to create a new datatype neither a new view to present the results of these formulations.



Simulation Information

Method Name:	Cobra FBA
Solution Type:	Not available.
Environmental Conditions:	Not available.
Objective Function	Max: R_BiomassEcoli = 0.8739215069684303
Biomass value:	0.8739215069684303

Net Conversions:

Consumption

Metabolite Id	Metabolite Name	Value
M_nh4_e	Ammonium	4,76532E0
M_pi_e	Phosphate	3,2149E0
M_o2_e	O2	2,17995E1
M_glc_D_e	D-Glucose	1E1

Production

Metabolite Id	Metabolite Name	Value
M_co2_e	CO2	2,28098E1
M_h2o_e	H2O	2,91758E1
M_h_e	H	1,75309E1

Figure 17 - Simulation datatype and result view

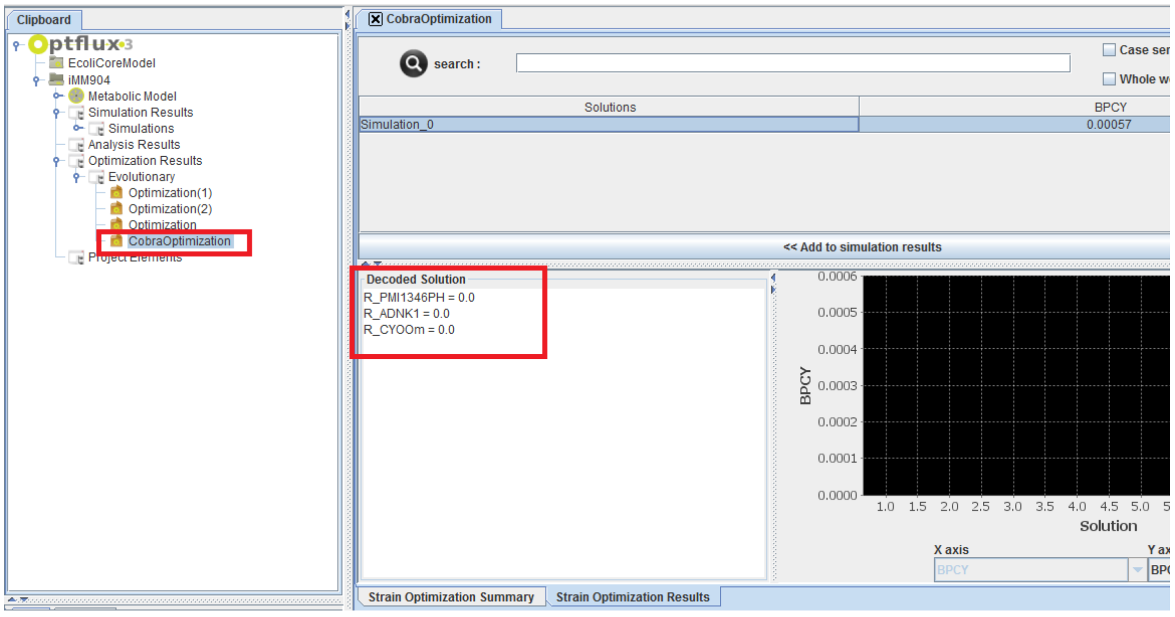
This view is the same for all simulations results and it contains all the information from that formulation. This information is the simulation method that was used, the type of solution, if it was used any environmental conditions, the value of the objective function and biomass value, the genetic conditions used, the extra metabolite and reaction information and also the flux distribution resulting from the simulation.

COBRA optimization in OptFlux

The integration of COBRA optimization formulation in OptFlux was different from the simulations. This occurred due the fact that the optimization definition and purpose is different from the simulation, the same occurs for the input parameters and results. Another difference is the control center used, it was created an optimization control center where these formulations and properties are registered and also performed.

Being the input parameters slightly different from the simulations and both categories cannot be considered as equals it was needed to implement a new operation and GUI in OptFlux that allowed performing these formulations. In this GUI, the input parameters are registered in the optimization control center and passed to the formulation through properties. In this operation, the control center will return a simulation structure and convert it in a structure that is the equivalent to the result structure of the OptFlux's optimization formulations.

Through this conversion the result of this COBRA's optimizations will be considered an OptFlux's optimization and with that the datatypes and views that present this category of formulations will also present this COBRA's result.



Solutions	BPCY
Simulation_0	0.00057

```
R_PMI1346PH = 0.0
R_ADNK1 = 0.0
R_CYOOm = 0.0
```

Figure 18 – Optimization datatype and view

Figure 18 represents the datatype and the view of a COBRA optimization formulation in OptFlux. The datatype is present along with the other optimizations results. The view is also the same used for this type of results and consists in the list of reactions to be knocked-out, the value of the desired product, the environmental conditions used, a structure that represents a simulation and can be added to the clipboard along with the other simulations datatypes. This integration shows how easy it was to convert a COBRA's optimization result in a result supported by OptFlux thanks to the structure developed.

Other plugin functionalities

Since this plugin needs a MATLAB connection to perform the COBRA operations it is also needed to create properties that control this connection. For that reason, a new node was added to the preferences of OptFlux. Here the path of the MATLAB application and the path to the JMI jar [4.2] will be added to guarantee that this plugin can connect with MATLAB and perform the implemented COBRA's formulations.

This preference node can also be used in the future to register some properties that can be common in the formulations, for instance the solver to be defined in COBRA.

With this plugin the OptFlux is able to integrate MATLAB-COBRA in its operations. This plugin, as all OptFlux plugins is open-source and new COBRA's formulations can be implemented in the future. This plugin can be installed from the RepositoryManager. To obtain the software and further information, visit the website www.optflux.org. A tutorial of this plugin operations' is also available on-line at the OptFlux Wiki (http://darwin.di.uminho.pt/optfluxwiki/).

4.6. Case studies

To prove the value of the developed plugin, three different strain optimization methods were used: the OptFlux's OptGene, the Cobra's GDLS and the Cobra's OptKnock. These methods were all performed in two case studies with two different desired products. These case studies objective is to demonstrate that the developed plugin can perform these methods through the OptFlux tool.

These three methods all provide the same output: a set of changes in an organism, in the form of proposed gene knockouts, so that it will be able to produce a product of interest. However, the methods have different input parameters.

The OptFlux's OptGene parameters were the glucose, as subtract flux, the FBA as simulation method and the Strength Pareto Evolutionary Algorithm 2 [93] (SPEA2) as optimization algorithm and 10000 as maximum number of solutions evaluations. The Biomass-Product Coupled Yield was defined as objective, which means that the method searches for mutants that are likely to exhibit higher productivities, since biomass production is also included in the objective. As for the maximum number of knockouts there were performed optimizations for both three and five knockouts. The solver used for this formulation was CPLEX.

The COBRA’s GDLS parameters were the number of maximum knockouts, also three and five, and a minimum growth rate of 0,05 mmol gDW-1 h-1. This parameter assures that the knockouts returned from the method do not result in an organism without growth. The solver used for this formulation was Gurobi.

The COBRA’s OptKnock parameters were the maximum number of knockouts, also three and five, and a restriction of 0.05 on the biomass value to assure the growth of the organism. The solver used for this formulation was also Gurobi.

For these case studies the *Saccharomyces cerevisiae* iMM904 model [94] was used in an aerobic condition. The model is available for download directly from OptFlux's internal repository. It has 1577 reactions (164 external and 1413 internal), 1228 metabolites (164 external, 1064 internal). These two case studies present the results of the optimization methods when trying to maximize two different products, the succinate and malate.

4.6.1. Case study I: succinate

For the first case study the target product of the optimizations was succinate, a precursor to a wide variety of products, from pharmaceuticals to the food industry. The performed optimizations resulted in the data presented in Table 4. This table is divided in the input parameters and in the results of the optimization.

The input parameters are the method used, the target, in this case the succinate and the maximum number of knockout reactions. The result consists in the value of the biomass, the maximum product value and a list of possible knockout reactions, from the best solution obtained from these three methods in these conditions.

The results of the Table 4 show that, although these three methods have the same objective, they present different possible knockouts and different biomass and product values. The results show that both OptGene and OptKnock formulations resulted, in this case, knockouts that guarantee a good value of desired product and also assure the growth of the organism. On the other hand the results of the GDLS show that the value of the desired value has a bigger weight in the objective of the method.

The major goal of this case study is not the comparison of results from these methods but to demonstrate that with this plugin it is possible to perform different methods from different tools through OptFlux. This also shows that centralizing several methods from different tools in a single tool is an advantage for the researcher analysis.

Table 4: Optimization values for Succinate

Input Parameters		**Results**		
Method	**Max KOs**	**Biomass mmol gDW^{-1} h^{-1}**	**Product Value mmol gDW^{-1} h^{-1}**	**Result KOs IDs**
OptFlux OptGene	3	0.20043	8.00868	R_Plt2m R_PAtm_SC R_CO2t
	5	0.19373	7.73036	R_AKGt2r R_EX_co2_e_ R_EX_btd_RR_e_ R_D_LACt2 R_ETOHt
COBRA's GDLS	3	0.05818	12.09006	R_CYOR_u6m R_ME2m R_PYRDC
	5	0.05818	12.09006	R_CYOR_u6m R_ME2m R_PYRDC
COBRA's OptKnock	3	0.14998	10.45342	R_G3PT R_CO2t R_MDH
	5	0.20623	7.71296	R_ALAt2r R_BTDD_RR R_ETOHt R_PFK_3 R_VALt2m

4.6.2. Case study II: malate

In this case study multiple approaches to optimize malate production were performed. The problem settings were similar to the first except for the target product defined in the optimizations, the malate reaction. Malic acid is an organic compound that is produced by all living organisms. This compound is used as food additive and contributes to the sour taste of fruits, for instance green apples.

Table 5 structure is similar to Table 4. The results consist in the value of the biomass, the maximum product value and a list of possible knockout reactions and are presented next.

Table 5: Optimization values for Malate

Input Parameters		**Results**		
Method	**Max KOs**	**Biomass mmol gDW^{-1} h^{-1}**	**Product Value mmol gDW^{-1} h^{-1}**	**Result KOs IDs**
OptFlux OptGene	3	0.28787	0.0	-
	5	0.28787	0.0	-
COBRA's GDLS	3	0.28786	0.0	-
	5	0.28786	0.0	-
COBRA's OptKnock	3	0.12630	9.37051	R_CO2t R_FUM R_H2Otm
	5	0.05265	10.36379	R_ATPtm_H R_FRDcm R_MALtm R_PDHm R_PYRDC

The results of this table show that the production of malate from this organism is not easy to accomplish if a minimum growth is a restriction. The results show that both OptGene and GDLS approaches were unable to find a list of knockouts that could guarantee a minimum growth and the production of malate. On the other hand the results of the OptKnock show it is possible for the organism to produce malate and also have a minimum growth.

Similar to the first case study the major goal was not comparing results or performance of these methods but to demonstrate that using this plugin the researcher can analyze in parallel results obtained from methods that exist in different tools and with that achieve better conclusions.

Chapter 5

Conclusions and Future Work

5.1. Discussion

In the last decade, the number of metabolic tools and methods used for phenotype simulation of microorganisms has grown exponentially. Almost all of these tools and methods have a constraint-based model as center of their operation. Also, with this growth these constraint-based tools were developed for slightly different purposes, with different methods implemented and also built in different computational languages. All these differences sometimes become a problem when it is time to choose which tool to use.

For this work, a platform was developed in Java and its major feature is the ability to integrate and use these constraint-based analysis tools despite the distinct computational languages. With this, all the formulations from any tool will be centralized in a single platform which allows the user to access any available formulation and perform it without the need of a vast knowledge of any of the integrated tools and formulations.

The structure of this platform enables the continuous expansion with methods and language connections by integrating other tools. At the moment this platform has a Java-MATLAB connection and in terms of tools has an OptFlux-COBRA assembly.

This platform was integrated with OptFlux by the development of a plugin. This allows ME researchers to use some of the main COBRA's formulations through OptFlux. These formulations are the FBA, LinearMOMA, MOMA and GeometricFBA for the simulation category, as for the optimization category OptFlux can use the OptKnock and GDLS formulations that are presented in COBRA.

By being integrated with OptFlux, this plugin allows the user to perform the mentioned COBRA's formulation through a user friendly interface, visualize and also compare the results from different methods with the same objective. This proves to be an advantage because COBRA is a script-based tool and thus obligates the users to have certain knowledge in informatics and

MATLAB, while through this plugin any user can perform the stated formulations without the need of knowing them in detail.

The developed platform is well documented and, thus, it is possible for any developer to integrate new languages and toolboxes in the platform and with this a constant expansion is possible. As for the OptFlux plugin, being this an open-source tool the integration of new COBRA methods can be easily performed by any developer with some Java knowledge.

5.2. Future work

The goals proposed in this work were mostly accomplished, but still there are some points that can be improved in future work. These points can be divided between the platform and the OptFlux's plugin and are the following:

- Integrate the connection to other computational languages within this platform, for instance creating a connection with Python or R.
- Integrate new toolboxes and extend the available constraint-based methods.
- In the plugin, new constraint-based methods present in COBRA could be integrated, also new structures that allow other categories of methods could be also implemented and with that integrating for instance analysis methods;
- Integrate other tools and frameworks built in MATLAB, for instance RAVEN and TIGER toolboxes and their most relevant methods.
- Enable the plugin to import and export models to and from MATLAB, this functionality would allow OptFlux to use models built in COBRA.

Bibliography

[1] G. Stephanopoulos, “Metabolic fluxes and metabolic engineering.,” *Metab. Eng.*, vol. 1, no. 1, pp. 1–11, Jan. 1999.

[2] M. Tomita, “Whole-cell simulation: a grand challenge of the 21st century.,” *Trends Biotechnol.*, vol. 19, no. 6, pp. 205–10, Jun. 2001.

[3] A. M. Feist, M. J. Herrgård, I. Thiele, J. L. Reed, and B. Ø. Palsson, “Reconstruction of biochemical networks in microorganisms.,” *Nat. Rev. Microbiol.*, vol. 7, no. 2, pp. 129–43, Feb. 2009.

[4] K. J. Kauffman, P. Prakash, and J. S. Edwards, “Advances in flux balance analysis,” *Curr. Opin. Biotechnol.*, vol. 14, no. 5, pp. 491–496, Oct. 2003.

[5] D. Segrè, D. Vitkup, and G. M. Church, “Analysis of optimality in natural and perturbed metabolic networks.,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 99, no. 23, pp. 15112–7, Nov. 2002.

[6] J. Schellenberger, R. Que, R. M. T. Fleming, I. Thiele, J. D. Orth, A. M. Feist, D. C. Zielinski, A. Bordbar, N. E. Lewis, S. Rahmanian, J. Kang, D. R. Hyduke, and B. Ø. Palsson, “Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0.,” *Nat. Protoc.*, vol. 6, no. 9, pp. 1290–307, Sep. 2011.

[7] S. Klamt, J. Saez-Rodriguez, and E. Gilles, “Structural and functional analysis of cellular networks with CellNetAnalyzer,” *BMC Syst. Biol.*, vol. 13, pp. 1–13, 2007.

[8] I. Rocha, P. Maia, P. Evangelista, P. Vilaça, S. Soares, J. P. Pinto, J. Nielsen, K. R. Patil, E. C. Ferreira, and M. Rocha, “OptFlux: an open-source software platform for in silico metabolic engineering.,” *BMC Syst. Biol.*, vol. 4, p. 45, Jan. 2010.

[9] J. Nielsen, *Encyclopedia of Physical Science and Technology*, 3^{o} ed. 2002, pp. 391–406.

[10] K. R. Patil, M. Akesson, and J. Nielsen, “Use of genome-scale microbial models for metabolic engineering.,” *Curr. Opin. Biotechnol.*, vol. 15, no. 1, pp. 64–9, Feb. 2004.

[11] B. Ø. Palsson, *Systems Biology Properties of Reconstructed Networks*. Cambridge University Press, 2006.

[12] N. D. Price, J. L. Reed, and B. Ø. Palsson, “Genome-scale models of microbial cells: evaluating the consequences of constraints.,” *Nat. Rev. Microbiol.*, vol. 2, no. 11, pp. 886–97, Nov. 2004.

[13] J. S. Edwards, “Systems Properties of the Haemophilus influenzae Rd Metabolic Genotype,” *J. Biol. Chem.*, vol. 274, no. 25, pp. 17410–17416, Jun. 1999.

[14] A. M. Feist, M. J. Herrgård, I. Thiele, J. L. Reed, and Ø. Bernhard, “Reconstruction of Biochemical Networks in Microbial Organisms,” vol. 7, no. 2, pp. 129–143, 2011.

[15] J. S. Edwards, R. U. Ibarra, and B. O. Palsson, “In silico predictions of Escherichia coli metabolic capabilities are consistent with experimental data.,” *Nat. Biotechnol.*, vol. 19, no. 2, pp. 125–30, Feb. 2001.

[16] R. U. Ibarra, J. S. Edwards, and B. O. Palsson, "Escherichia coli K-12 undergoes adaptive evolution to achieve in silico predicted optimal growth," vol. 420, no. November, pp. 20–23, 2002.

[17] A. P. Burgard and C. D. Maranas, "Optimization-based framework for inferring and testing hypothesized metabolic objective functions.," *Biotechnol. Bioeng.*, vol. 82, no. 6, pp. 670–7, Jun. 2003.

[18] A. K. Gombert and J. Nielsen, "Mathematical modelling of metabolism.," *Curr. Opin. Biotechnol.*, vol. 11, no. 2, pp. 180–6, Apr. 2000.

[19] W. Wiechert, "Modeling and simulation: tools for metabolic engineering.," *J. Biotechnol.*, vol. 94, no. 1, pp. 37–63, Mar. 2002.

[20] G. R. Cronwright, J. M. Rohwer, and B. A. Prior, "Metabolic control analysis of glycerol synthesis in Saccharomyces cerevisiae.," *Appl. Environ. Microbiol.*, vol. 68, no. 9, pp. 4448–56, Sep. 2002.

[21] W. Prathumpai, J. B. Gabelgaard, P. Wanchanthuek, P. J. I. van de Vondervoort, M. J. L. de Groot, M. McIntyre, and J. Nielsen, "Metabolic control analysis of xylose catabolism in Aspergillus.," *Biotechnol. Prog.*, vol. 19, no. 4, pp. 1136–41.

[22] C. Chassagnole, N. Noisommit-Rizzi, J. W. Schmid, K. Mauch, and M. Reuss, "Dynamic modeling of the central carbon metabolism of Escherichia coli.," *Biotechnol. Bioeng.*, vol. 79, no. 1, pp. 53–73, Jul. 2002.

[23] S. J. Wiback, I. Famili, H. J. Greenberg, and B. Ø. Palsson, "Monte Carlo sampling can be used to determine the size and shape of the steady-state flux space.," *J. Theor. Biol.*, vol. 228, no. 4, pp. 437–47, Jun. 2004.

[24] F. Llaneras and J. Picó, "An interval approach for dealing with flux distributions and elementary modes activity patterns.," *J. Theor. Biol.*, vol. 246, no. 2, pp. 290–308, May 2007.

[25] N. D. Price, J. A. Papin, C. H. Schilling, and B. O. Palsson, "Genome-scale microbial in silico models: the constraints-based approach.," *Trends Biotechnol.*, vol. 21, no. 4, pp. 162–9, Apr. 2003.

[26] A. R. Zomorrodi, P. F. Suthers, S. Ranganathan, and C. D. Maranas, "Mathematical optimization applications in metabolic networks.," *Metab. Eng.*, vol. 14, no. 6, pp. 672–86, Nov. 2012.

[27] H. Alper, Y.-S. Jin, J. F. Moxley, and G. Stephanopoulos, "Identifying gene targets for the metabolic engineering of lycopene biosynthesis in Escherichia coli.," *Metab. Eng.*, vol. 7, no. 3, pp. 155–64, May 2005.

[28] S. Atsumi, A. F. Cann, M. R. Connor, C. R. Shen, K. M. Smith, M. P. Brynildsen, K. J. Y. Chou, T. Hanai, and J. C. Liao, "Metabolic engineering of Escherichia coli for 1-butanol production.," *Metab. Eng.*, vol. 10, no. 6, pp. 305–11, Nov. 2008.

[29] S. Atsumi and J. C. Liao, "Directed evolution of Methanococcus jannaschii citramalate synthase for biosynthesis of 1-propanol and 1-butanol by Escherichia coli.," *Appl. Environ. Microbiol.*, vol. 74, no. 24, pp. 7802–8, Dec. 2008.

[30] D. R. Bond and D. R. Lovley, "Electricity production by Geobacter sulfurreducens attached to electrodes.," *Appl. Environ. Microbiol.*, vol. 69, no. 3, pp. 1548–55, Mar. 2003.

[31] A. P. Burgard, P. Pharkya, and C. D. Maranas, “Optknock: a bilevel programming framework for identifying gene knockout strategies for microbial strain optimization.,” *Biotechnol. Bioeng.*, vol. 84, no. 6, pp. 647–57, Dec. 2003.

[32] N. Misawa, S. Yamano, and H. Ikenaga, “Production of beta-carotene in Zymomonas mobilis and Agrobacterium tumefaciens by introduction of the biosynthesis genes from Erwinia uredovora.,” *Appl. Environ. Microbiol.*, vol. 57, no. 6, pp. 1847–9, Jun. 1991.

[33] C. E. Nakamura and G. M. Whited, “Metabolic engineering for the microbial production of 1,3-propanediol.,” *Curr. Opin. Biotechnol.*, vol. 14, no. 5, pp. 454–9, Oct. 2003.

[34] A. P. Oliveira, J. Nielsen, and J. Förster, “Modeling Lactococcus lactis using a genome-scale flux model.,” *BMC Microbiol.*, vol. 5, p. 39, Jan. 2005.

[35] P. Pharkya, A. P. Burgard, and C. D. Maranas, “Exploring the overproduction of amino acids using the bilevel optimization framework OptKnock.,” *Biotechnol. Bioeng.*, vol. 84, no. 7, pp. 887–99, Dec. 2003.

[36] M. Sauer, D. Porro, D. Mattanovich, and P. Branduardi, “Microbial production of organic acids: expanding the markets.,” *Trends Biotechnol.*, vol. 26, no. 2, pp. 100–8, Feb. 2008.

[37] E. Scott, F. Peter, and J. Sanders, “Biomass in the manufacture of industrial products—the use of proteins and amino acids,” *Appl. Microbiol. Biotechnol.*, vol. 75, no. 4, pp. 751–762, Mar. 2007.

[38] P. J. Bosma, “Inherited disorders of bilirubin metabolism.,” *J. Hepatol.*, vol. 38, no. 1, pp. 107–17, Jan. 2003.

[39] C. J. Danpure, “Primary hyperoxaluria type 1: AGT mistargeting highlights the fundamental differences between the peroxisomal and mitochondrial protein import pathways.,” *Biochim. Biophys. Acta*, vol. 1763, no. 12, pp. 1776–84, Dec. 2006.

[40] A. Zelezniak, T. H. Pers, S. Soares, M. E. Patti, and K. R. Patil, “Metabolic network topology reveals transcriptional regulatory signatures of type 2 diabetes.,” *PLoS Comput. Biol.*, vol. 6, no. 4, p. e1000729, Apr. 2010.

[41] N. Jamshidi and B. Ø. Palsson, “Investigating the metabolic capabilities of Mycobacterium tuberculosis H37Rv using the in silico strain iNJ661 and proposing alternative drug targets.,” *BMC Syst. Biol.*, vol. 1, p. 26, Jan. 2007.

[42] D.-Y. Lee, L. T. Fan, S. Park, S. Y. Lee, S. Shafie, B. Bertók, and F. Friedler, “Complementary identification of multiple flux distributions and multiple metabolic pathways.,” *Metab. Eng.*, vol. 7, no. 3, pp. 182–200, May 2005.

[43] T. Shlomi, O. Berkman, and E. Ruppin, “Regulatory on/off minimization of metabolic flux changes after genetic perturbations.,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 102, no. 21, pp. 7695–700, May 2005.

[44] E. P. Gianchandani, M. A. Oberhardt, A. P. Burgard, C. D. Maranas, and J. A. Papin, “Predicting biological system objectives de novo from internal state measurements.,” *BMC Bioinformatics*, vol. 9, p. 43, Jan. 2008.

[45] R. Mahadevan and C. H. Schilling, “The effects of alternate optimal solutions in constraint-based genome-scale metabolic models.,” *Metab. Eng.*, vol. 5, no. 4, pp. 264–76, Oct. 2003.

[46] A. P. Burgard, E. V Nikolaev, C. H. Schilling, and C. D. Maranas, “Flux coupling analysis of genome-scale metabolic network reconstructions.,” *Genome Res.*, vol. 14, no. 2, pp. 301–12, Feb. 2004.

[47] L. David, S.-A. Marashi, A. Larhlimi, B. Mieth, and A. Bockmayr, “FFCA: a feasibility-based method for flux coupling analysis of metabolic networks.,” *BMC Bioinformatics*, vol. 12, no. 1, p. 236, Jan. 2011.

[48] D. A. Fell and J. R. Small, “Fat synthesis in adipose tissue. An examination of stoichiometric constraints.,” *Biochem. J.*, vol. 238, no. 3, pp. 781–6, Sep. 1986.

[49] J. M. Savinell and B. O. Palsson, “Network analysis of intermediary metabolism using linear optimization. II. Interpretation of hybridoma cell metabolism.,” *J. Theor. Biol.*, vol. 154, no. 4, pp. 455–73, Feb. 1992.

[50] A. Varma and B. O. Palsson, “Metabolic Capabilities of Escherichia coli: I. Synthesis of Biosynthetic Precursors and Cofactors,” *J. Theor. Biol.*, vol. 165, no. 4, pp. 477–502, Dec. 1993.

[51] M. W. Covert, C. H. Schilling, and B. Palsson, “Regulation of gene expression in flux balance models of metabolism.,” *J. Theor. Biol.*, vol. 213, no. 1, pp. 73–88, Nov. 2001.

[52] T. Shlomi, Y. Eisenberg, R. Sharan, and E. Ruppin, “A genome-scale computational study of the interplay between transcriptional regulation and metabolism.,” *Mol. Syst. Biol.*, vol. 3, p. 101, Jan. 2007.

[53] S. A. Becker and B. O. Palsson, “Context-specific metabolic networks are consistent with experiments.,” *PLoS Comput. Biol.*, vol. 4, no. 5, p. e1000082, May 2008.

[54] S. Chandrasekaran and N. D. Price, “Probabilistic integrative modeling of genome-scale metabolic and regulatory networks in Escherichia coli and Mycobacterium tuberculosis.,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 107, no. 41, pp. 17845–50, Oct. 2010.

[55] K. Smallbone and E. Simeonidis, “Flux balance analysis: a geometric perspective.,” *J. Theor. Biol.*, vol. 258, no. 2, pp. 311–5, May 2009.

[56] S. Gudmundsson and I. Thiele, “Computationally efficient flux variability analysis.,” *BMC Bioinformatics*, vol. 11, no. 1, p. 489, Jan. 2010.

[57] A. Larhlimi, L. David, J. Selbig, and A. Bockmayr, “F2C2: a fast tool for the computation of flux coupling in genome-scale metabolic networks.,” *BMC Bioinformatics*, vol. 13, no. 1, p. 57, Jan. 2012.

[58] P. F. Suthers, Y. J. Chang, and C. D. Maranas, “Improved computational performance of MFA using elementary metabolite units and flux coupling.,” *Metab. Eng.*, vol. 12, no. 2, pp. 123–8, Mar. 2010.

[59] Y. Chang, P. F. Suthers, and C. D. Maranas, “Identification of optimal measurement sets for complete flux elucidation in metabolic flux analysis experiments.,” *Biotechnol. Bioeng.*, vol. 100, no. 6, pp. 1039–49, Aug. 2008.

[60] W. Wiechert, M. Möllney, S. Petersen, and A. A. de Graaf, “A universal framework for 13C metabolic flux analysis.,” *Metab. Eng.*, vol. 3, no. 3, pp. 265–83, Jul. 2001.

[61] N. Tepper and T. Shlomi, “Predicting metabolic engineering knockout strategies for chemical production: accounting for competing pathways.,” *Bioinformatics*, vol. 26, no. 4, pp. 536–43, Feb. 2010.

[62] A. M. Feist and B. O. Palsson, “The biomass objective function.,” *Curr. Opin. Microbiol.*, vol. 13, no. 3, pp. 344–9, Jun. 2010.

[63] M. Kanehisa, M. Araki, S. Goto, M. Hattori, M. Hirakawa, M. Itoh, T. Katayama, S. Kawashima, S. Okuda, T. Tokimatsu, and Y. Yamanishi, “KEGG for linking genomes to life and the environment.,” *Nucleic Acids Res.*, vol. 36, no. Database issue, pp. D480–4, Jan. 2008.

[64] R. Caspi, H. Foerster, C. A. Fulcher, R. Hopkinson, J. Ingraham, P. Kaipa, M. Krummenacker, S. Paley, J. Pick, S. Y. Rhee, C. Tissier, P. Zhang, and P. D. Karp, “MetaCyc: a multiorganism database of metabolic pathways and enzymes.,” *Nucleic Acids Res.*, vol. 34, no. Database issue, pp. D511–6, Jan. 2006.

[65] M. Scheer, A. Grote, A. Chang, I. Schomburg, C. Munaretto, M. Rother, C. Söhngen, M. Stelzer, J. Thiele, and D. Schomburg, “BRENDA, the enzyme information system in 2011.,” *Nucleic Acids Res.*, vol. 39, no. Database issue, pp. D670–6, Jan. 2011.

[66] P. Pharkya, A. P. Burgard, and C. D. Maranas, “OptStrain: a computational framework for redesign of microbial production systems.,” *Genome Res.*, vol. 14, no. 11, pp. 2367–76, Nov. 2004.

[67] J. Kim, J. L. Reed, and C. T. Maravelias, “Large-scale bi-level strain design approaches and mixed-integer programming solution techniques.,” *PLoS One*, vol. 6, no. 9, p. e24162, Jan. 2011.

[68] P. Pharkya and C. D. Maranas, “An optimization framework for identifying reaction activation/inhibition or elimination candidates for overproduction in microbial systems.,” *Metab. Eng.*, vol. 8, no. 1, pp. 1–13, Jan. 2006.

[69] D. S. Lun, G. Rockwell, N. J. Guido, M. Baym, J. A. Kelner, B. Berger, J. E. Galagan, and G. M. Church, “Large-scale identification of genetic design strategies using local search.,” *Mol. Syst. Biol.*, vol. 5, p. 296, Jan. 2009.

[70] H. S. Choi, S. Y. Lee, T. Y. Kim, and H. M. Woo, “In silico identification of gene amplification targets for improvement of lycopene production.,” *Appl. Environ. Microbiol.*, vol. 76, no. 10, pp. 3097–105, May 2010.

[71] J. Kim and J. L. Reed, “OptORF: Optimal metabolic and regulatory perturbations for metabolic engineering of microbial strains.,” *BMC Syst. Biol.*, vol. 4, no. 1, p. 53, Jan. 2010.

[72] S. Ranganathan, P. F. Suthers, and C. D. Maranas, “OptForce: an optimization procedure for identifying all genetic manipulations leading to targeted overproductions.,” *PLoS Comput. Biol.*, vol. 6, no. 4, p. e1000744, Apr. 2010.

[73] L. Yang, W. R. Cluett, and R. Mahadevan, “EMILiO: a fast algorithm for genome-scale strain design.,” *Metab. Eng.*, vol. 13, no. 3, pp. 272–81, May 2011.

[74] K. R. Patil, I. Rocha, J. Förster, and J. Nielsen, “Evolutionary programming as a platform for in silico metabolic engineering.,” *BMC Bioinformatics*, vol. 6, p. 308, Jan. 2005.

[75] V. Satish Kumar, M. S. Dasika, and C. D. Maranas, “Optimization based automated curation of metabolic reconstructions.,” *BMC Bioinformatics*, vol. 8, p. 212, Jan. 2007.

[76] J. L. Reed, T. R. Patel, K. H. Chen, A. R. Joyce, M. K. Applebee, C. D. Herring, O. T. Bui, E. M. Knight, S. S. Fong, and B. O. Palsson, “Systems approach to refining genome annotation.,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 103, no. 46, pp. 17480–4, Nov. 2006.

[77] C. S. Henry, J. F. Zinner, M. P. Cohoon, and R. L. Stevens, “iBsu1103: a new genome-scale metabolic model of Bacillus subtilis based on SEED annotations.,” *Genome Biol.*, vol. 10, no. 6, p. R69, Jan. 2009.

[78] A. R. Zomorrodi and C. D. Maranas, “Improving the iMM904 S. cerevisiae metabolic model using essentiality and synthetic lethality data.,” *BMC Syst. Biol.*, vol. 4, p. 178, Jan. 2010.

[79] D. Barua, J. Kim, and J. L. Reed, “An automated phenotype-driven approach (GeneForce) for refining metabolic and regulatory models.,” *PLoS Comput. Biol.*, vol. 6, no. 10, p. e1000970, Jan. 2010.

[80] M. J. Herrgård, S. S. Fong, and B. Ø. Palsson, “Identification of genome-scale metabolic network models using experimentally measured flux profiles.,” *PLoS Comput. Biol.*, vol. 2, no. 7, p. e72, Jul. 2006.

[81] C. S. Henry, L. J. Broadbelt, and V. Hatzimanikatis, “Thermodynamics-based metabolic flux analysis.,” *Biophys. J.*, vol. 92, no. 5, pp. 1792–805, Mar. 2007.

[82] J. Schellenberger, N. E. Lewis, and B. Ø. Palsson, “Elimination of thermodynamically infeasible loops in steady-state metabolic models.,” *Biophys. J.*, vol. 100, no. 3, pp. 544–53, Feb. 2011.

[83] J. Wright and A. Wagner, “The Systems Biology Research Tool: evolvable open-source software.,” *BMC Syst. Biol.*, vol. 2, p. 55, Jan. 2008.

[84] J. Schellenberger, R. Que, and R. Fleming, “Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2. 0,” *Nat. Protoc.*, vol. 6, no. 9, pp. 1290–1307, 2011.

[85] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J.-H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novere, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang, “The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models,” *Bioinformatics*, vol. 19, no. 4, pp. 524–531, Mar. 2003.

[86] M. Cvijovic, R. Olivares-Hernández, R. Agren, N. Dahr, W. Vongsangnak, I. Nookaew, K. R. Patil, and J. Nielsen, “BioMet Toolbox: genome-wide analysis of metabolism.,” *Nucleic Acids Res.*, vol. 38, no. Web Server issue, pp. W144–9, Jul. 2010.

[87] A. Hoppe, S. Hoffmann, A. Gerasch, C. Gille, and H.-G. Holzhütter, “FASIMU: flexible software for flux-balance computation series in large metabolic networks.,” *BMC Bioinformatics*, vol. 12, p. 28, Jan. 2011.

[88] A. Gevorgyan, M. E. Bushell, C. Avignone-Rossa, and A. M. Kierzek, “SurreyFBA: a command line tool and graphics user interface for constraint-based modeling of genome-scale metabolic reaction networks.,” *Bioinformatics*, vol. 27, no. 3, pp. 433–4, Feb. 2011.

[89] P. A. Jensen, K. A. Lutz, and J. A. Papin, “TIGER: Toolbox for integrating genome-scale metabolic models, expression data, and transcriptional regulatory networks.,” *BMC Syst. Biol.*, vol. 5, p. 147, Jan. 2011.

[90] R. Agren, L. Liu, S. Shoaie, W. Vongsangnak, I. Nookaew, and J. Nielsen, “The RAVEN toolbox and its use for generating a genome-scale metabolic model for Penicillium chrysogenum.,” *PLoS Comput. Biol.*, vol. 9, no. 3, p. e1002980, Jan. 2013.

[91] “A Java API to interact with MATLAB.” [Online]. Available: https://code.google.com/p/matlabcontrol/.

[92] A. Varma and B. O. Palsson, “Metabolic capabilities of Escherichia coli: I. synthesis of biosynthetic precursors and cofactors.,” *J. Theor. Biol.*, vol. 165, no. 4, pp. 477–502, Dec. 1993.

[93] E. Zitzler, M. Laumanns, and L. Thiele, “SPEA2: Improving the strength pareto evolutionary algorithm,” 2001.

[94] M. L. Mo, B. O. Palsson, and M. J. Herrgård, “Connecting extracellular metabolomic measurements to intracellular flux states in yeast.,” *BMC Syst. Biol.*, vol. 3, no. 1, p. 37, Jan. 2009.