XML format for RBA models, version 1

BioSys group, INRA Jouy, France

January 15, 2019

Contents

1 Introduction

1.1 Overview of an RBA model

In Figure 1, we summarize the systemic cell description underlying to the RBA method and the mathematical relationships defining the interactions and allocation of resources between the cellular processes. All these relationships take the form of linear growth-rate dependent equalities and inequalities: for cells growing in exponential phase at a rate μ, (I) the metabolic network has to produce all metabolic precursors necessary for biomass production (equalities C1 in green); (II) the capacity of all molecular machines must be sufficient to ensure their function, i.e. to catalyze chemical conversions at a sufficient rate (inequalities C2 in blue for the enzymes and transporters, in yellow for the molecular machines of macromolecular processes); (III) the intracellular density of compartments and the occupancy of membranes are limited (inequalities C3 in red); (IV) mass conservation is satisfied for all molecule types (equalities C1 in green). Taken together, the equalities and inequalities define, at a given rate μ, a feasibility linear programming (LP) problem that can be solved efficiently. Parsimonious resource allocation between cellular processes is modelled mathematically by optimizing the maximal cell growth, and computed by solving a series of such LP feasibility problems for different growth rate values. For a given medium, an RBA optimization problem predicts the maximal possible growth rate, the corresponding reaction fluxes and the abundances of molecular machines. Consequently, generating an RBA model requires information for formalizing constraints C1, C2 and C3 (Figure 1), and in particular: (i) the localization and the composition of the molecular machines, (ii) the molecules that are consumed and released by the molecular machines for functioning; (iii) the efficiencies of molecular machines, i.e. the rates of the process per amount of the catalyzing molecular machine; (iv) parameters such as the maximal density of each compartment.

1.2 Overview of the document

In this document we present the XML structures used to define a RBA model. A complete RBA model is composed of the following files:

- metabolism.xml (definition of compartments, metabolic species and metabolic reactions).
- proteins.xml (definition of proteins).
- rnas.xml (definition of RNAs).
- dna.xml (definition of DNA).
- enzymes.xml (definition of enzymatic machineries catalyzing metabolic reactions).
- processes.xml (definition of cell processes used to produce macromolecules).
- density.xml (definition of density constraints).

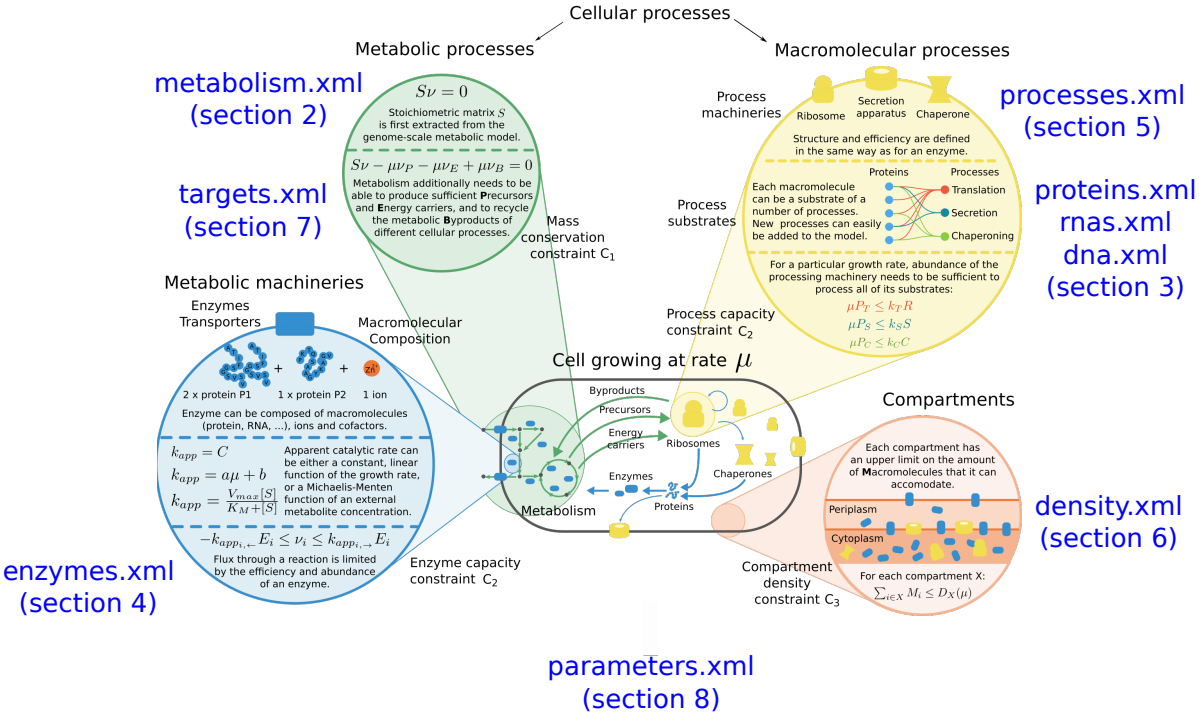

Figure 1: Overview of the files used in the XML model. Files were designed according to the 3 central constraints of RBA.

- targets.xml (definition of production or degradation constraints necessary for growth and maintenance).
- parameters.xml (definition of user-defined functions).

A valid RBA model must contain all these files with these exact names in the same directory to be recognized by the RBApy parser. We dedicate one section of this document for every file. Each section starts with a subsection containing a brief description about the relevance of the node in the RBA model. The following subsections are more technical: we present the nodes that compose the XML structure and show a class diagram with the nodes' attributes and the children nodes that they may/must contain. The last subsection presents examples from real models and from a minimal model (see XML tutorial for RBApy).

1.3 Conventions

Before we start looking into files, we briefly review the conventions we used for the RBA format.

1.3.1 Naming conventions in XML and RBApy

We strongly advise **not** to modify XML files directly. Instead, RBApy can be used to access the XML files and modify them through scripts. Writing scripts allows you to

reproduce all the modifications you have made automatically. For example, if you need to regenerate the basic XML model with preRBA, you will be able to re-apply all the modifications that you had done on your orginal model.

All XML elements can be accessed with the RBApy package. For example, the XML element Reaction can be accessed through python objects of class Reaction. This mirrorring scheme (every XML element has a corresponding python class) is inspired by the scheme used in SBML and libSBML. However, as in libSBML, the naming conventions may differ between python and XML:

- XML elements and python classes both follow `ThisConvention` (e.g. `SpeciesReference`).
- XML attributes follow `thisConvention` (e.g. `boundaryCondition`), while python attributes follow `this_convention` (e.g. `boundary_condition`).
- In XML, list elements follow the convention `listOfThings` when they are seen as an instance/subelement (e.g. `RBAMetabolism` contains one instance of `ListOfCompartments` called `listOfCompartments`). In python, the `listOf` prefix is dropped and lower case is used (e.g. `RBAMetabolism` contains one instance of `ListOfCompartments` called `compartments`).

Throughout this document, XML conventions are used. Please keep in mind that the convention for attributes/instances is different when using the python package.

1.3.2 Boolean attributes

A boolean attribute evaluates to true if it is `"1"` or `"true"` (case does not matter). In all other cases it evaluates to false.

1.3.3 Variables for user-defined functions

The default variable for functions is the growth rate. It can also be explicitly defined as by the string `growth_rate`. Alternatively, a function can take as an input the *external* concentration of a metabolite (e.g. for transport functions). A metabolite identifier is expected to look like `met_c`, where `met` is the name of the metabolite and `c` its compartment. Note that in the current version of RBApy, every time a function based on `met_c` is evaluated, the compartment suffix is ignored and the extracellular concentration of the metabolite is used *no matter where the transport takes place*. Typically, glucose import rates from the periplasm to the cytosol will be based on the concentration of extracellular glucose.

1.3.4 Units

In the current version of RBApy, units are implicitly defined! It is left to the user to make sure that the model is consistent. For example, density constraints rely on a weight unit that is used in the density file (density.xml) and the macromolecule files (proteins.xml, rnas.xml, dna.xml). You need to make sure that you use the same unit in all files! In

the density example, our group usually uses amino acids as a weight unit. The density file sets an upper limit in the number of amino acids that can be contained in the cell. The protein file defines the weight of all amino acid residues to be 1. For RNAs and DNA, we convert the weight of nucleotides to amino acid equivalents. For example, the weight of an Adenosine residue is 2.9036 (a nucleotide is heavier than an amino acid).

2 metabolism.xml

The metabolism file is strongly inspired by SBML. More precisely, it can be seen as a subpart of an SBML file. It is used to define compartments, metabolites and reactions.

2.1 Rationale

metabolism.xml contains the most basic bricks of an RBA model. In our effort to define a minimal structure that contains an RBA model, we decided to start with an SBML structure and strip it down to elements that are essential to RBA.

metabolism.xml defines the structure of the metabolic network: simple chemical species (metabolites) that flow between compartments through transport reactions or transformed into other simple chemical species that will be available as building blocks for more complex molecules.

The description is entirely static: input fluxes are defined through the medium and parameters.xml, output fluxes are defined by targets.xml, the dynamics of internal fluxes is defined in enzymes.xml.

2.2 RBAMetabolism

The outermost part of the metabolism file is an instance of class **RBAMetabolism**, shown in Figure 2.

Currently, **RBAMetabolism** has no simple attributes. It includes exactly one instance of each **ListOf** container class. All **ListOf** classes do not have own attributes, they are merely used to organize a list of instances from another class. This organization was inspired by SBML.

2.3 Compartment

The **Compartment** class is used to list existing cell compartments.

The *id* attribute The **id** attribute is a string defining the identifier of a compartment. Every compartment should have a different id.

2.4 Species

The **Species** class is used to define *metabolic* species.

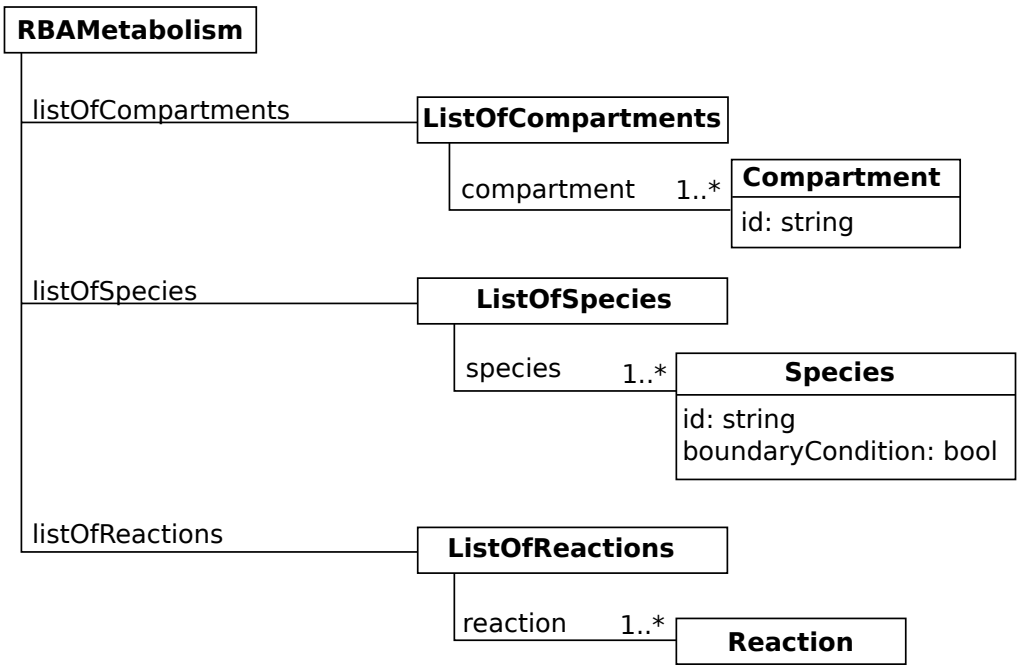

Figure 2: XML structure of metabolism document.

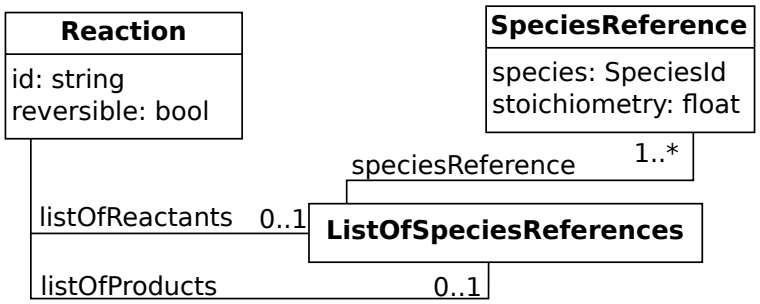

Figure 3: Class storing metabolic reactions.

The *id* attribute The **id** attribute is a string defining the identifier of a metabolite.

The *boundaryCondition* attribute The **boundaryCondition** attribute is a boolean. If the attribute is set to true, the metabolite is considered to be at a constant concentration. In other words, it is not affected by reactions. This is typical for metabolites in the external medium.

2.5 Reaction

The **Reaction** class is used to define metabolic reactions (Fig. 3). Reactants and products are defined using a **ListOfSpeciesReferences**.

The *id* attribute The **id** attribute is a string defining the identifier of a reaction.

The *reversible* attribute The **reversible** attribute is a boolean. If the attribute is set to true, the reaction can occur in both directions. If the attribute is set to false, only the forward reaction can occur.

2.6 SpeciesReference

The **SpeciesReference** class is used to refer to a metabolic **Species** and associate with it a stoichiometry (Fig. 3).

The *species* attribute The **species** attribute must match the identifier of a **Species**.

The *stoichiometry* attribute The **stoichiometry** is a positive real number. It repensents the stoichiometry of a **Species** in a given context (typically a **Reaction**).

2.7 Examples

Figure 4 shows a very simple example with 2 compartments, 4 metabolites and 3 reactions. In this example, we tagged `M_carbon_source_e` with `boundary_condition="true"`, implying that it is an external metabolite whose concentration is known and set through the medium in medium.tsv. Boundary metabolites are essential in the model, as they define input fluxes in the model.

Note that the description of the metabolic network ends with the protein precursor. Proteins *should not* be defined in metabolism.xml. Their composition is described in proteins.xml, while their assembly is described in processes.xml.

3 proteins.xml, rnas.xml and dna.xml

All these files are base on the same class **RBAMacromolecules**.

3.1 Rationale

In RBA, basic molecules are seen either as metabolites or macromolecules. In the current version, macromolecules encompass the polymer molecules proteins, RNAs and DNA. In future versions, we would like to extend the definition to other molecules that have polymer-like properties in their assembly process (e.g. lipids).

The formal distinction between macromolecules and other molecules is that macromolecules can be defined as an ensemble of other, simpler molecules or molecule residues. Elements of a macromolecule family (proteins, RNAs, DNA) are based on the same subset of component molecules (amino acid residues, vitamins, ions for proteins), share common assembly processes, but differ in the stoichiometry of components they are built of.

For example, a protein is often described as a sequence of letters, say "MAGLKYAAALK", where every letter implicitly represents a component (an amino acid residue). It may

```
<RBAMetabolism>
  <listOfCompartments>
    <compartment id="extracellular"/>
    <compartment id="cytosol"/>
  </listOfCompartments>
  <listOfSpecies>
    <species id="M_carbon_source_e" boundaryCondition="true"/>
    <species id="M_carbon_source_c" boundaryCondition="false"/>
    <species id="M_protein_component_precursor_c" boundaryCondition="false"/>
    <species id="M_biomass_c" boundaryCondition="false"/>
  </listOfSpecies>
  <listOfReactions>
    <reaction id="R_transport" reversible="false">
      <listOfReactants>
        <speciesReference species="M_carbon_source_e" stoichiometry="1"/>
      </listOfReactants>
      <listOfProducts>
        <speciesReference species="M_carbon_source_c" stoichiometry="1"/>
      </listOfProducts>
    </reaction>
    <reaction id="R_protein_component_precursor" reversible="false">
      <listOfReactants>
        <speciesReference species="M_carbon_source_c" stoichiometry="1"/>
      </listOfReactants>
      <listOfProducts>
        <speciesReference species="M_protein_component_precursor_c" stoichiometry="1"/>
      </listOfProducts>
    </reaction>
    <reaction id="R_biomass" reversible="false">
      <listOfReactants>
        <speciesReference species="M_carbon_source_c" stoichiometry="1"/>
      </listOfReactants>
      <listOfProducts>
        <speciesReference species="M_biomass_c" stoichiometry="1"/>
      </listOfProducts>
    </reaction>
  </listOfReactions>
</RBAMetabolism>
```

Figure 4: metabolism.xml from the minimal model with 2 compartments, 4 metabolites and 3 reactions.

also contain post-translational modifications, such as a phosphorylation on the tyrosine Y. The RBA format starts by listing the components that are common to all proteins: all amino acid residues (A, C, .., Y), vitamins, ions and other cofactors. The second part of the RBA format lists all proteins broken down as an ensemble of components. For example, the protein above would be described as {(A:4), (G:1), (K:2), (L:2), (M:1), (Y:1), (Phospho:1)}, where Phospho represents the tyrosine phosphorylation (in this example, we pooled all possible phosphorylations into a single component).

Note that macromolecule components are *not* metabolites. How they are built from metabolites is part of the assembly process defined in processes.xml. They may share identical identifiers as metabolites, but we advise using different identifiers for clarity. For example, an amino acid residue is *not* an amino acid: it is obtained by loading a charged tRNA onto a nascent protein with the appropriate energetic molecules, while releasing several byproducts, including an uncharged tRNA. In a sense, a component can be seen as an *atomic assembly action*, rather than a submolecule.

The macromolecule files contain static descriptions of the macromolecules. How they are assembled from metabolites and the machines needed to assemble them are defined in processes.xml.

3.2 RBAMacromolecules

The outermost part of the protein, RNA and DNA files is an instance of class **RBA-Macromolecules**, shown in Figure 5.

RBAMacromolecules has no simple attributes. It contains exactly one instance of **ListOfComponents** and **ListOfMacromolecules**.

3.3 Component

The **Component** class is used to define the components of a **Macromolecule** (Fig. 5). For example, these are expected to be amino acids, vitamins and ions for proteins. Even if there is a strong connection between metabolic **Species** and **Component**s, they are seen as independent entities with separate identifiers. The connection between **Species** and **Component**s is established in the process file, where **ProcessingMap**s define how components are assembled from metabolites.

The *id* attribute The **id** attribute is a string defining the identifier of a component.

The *weight* attribute The **weight** attribute is a real number defining the weight of a component. This information is essential for the density constraints. The weight of a macromolecule is defined as the sum of the weight of its components. The weight unit is unspecified, however it should be consistent with the parameters used in the density constraints.

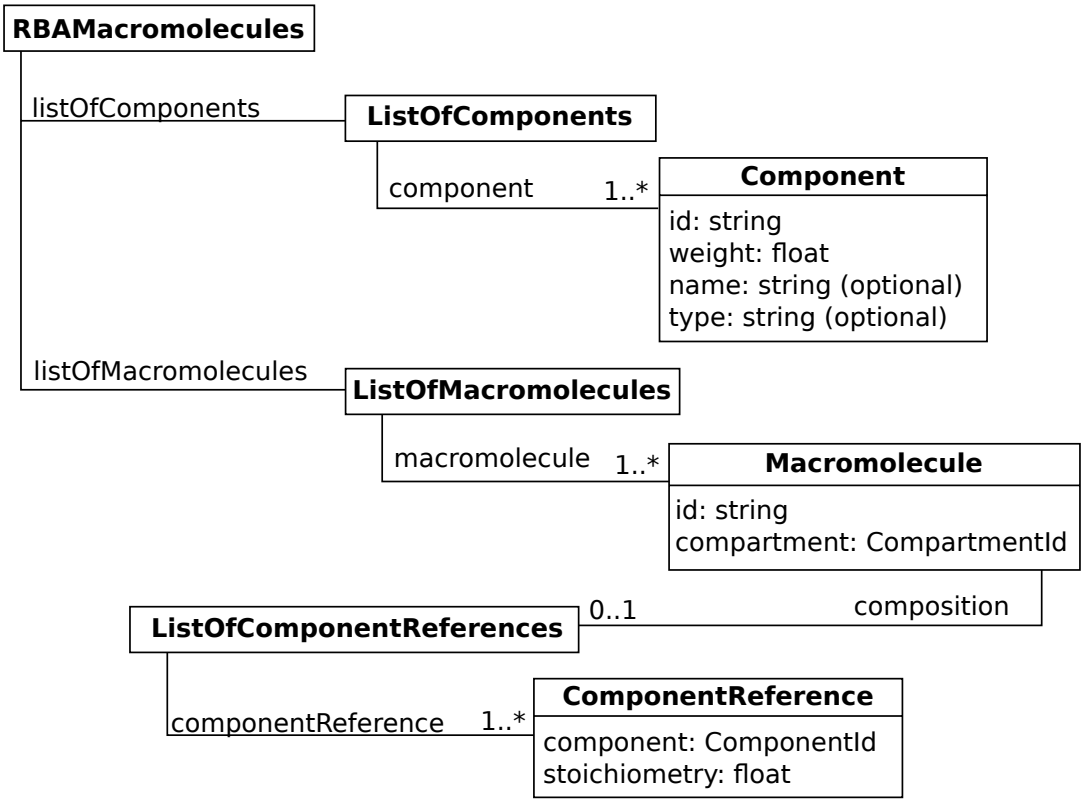

Figure 5: XML structure of macromolecule document.

The *name* and *type* attributes The **name** and **type** attributes are strings that provide additional information about the component. The name is a standard name of the component (*e.g.* full amino acid name). The type can be used to distinguish components if necessary (*e.g.* in amino acids, vitamins, ions).

3.4 Macromolecule

The **Macromolecule** class is used to define macromolecular species (Fig. 5). Its composition is given by a **ListOfComponentReferences**.

The *id* attribute The **id** attribute is a string defining the identifier of the macromolecule.

The *compartment* attribute The **compartment** attribute must match the identifier of a **Compartment**. It represents the compartment where the molecule is thought to be active.

3.5 ComponentReference

The **ComponentReference** class is used to refer to a **Component** and associate with it a stoichiometry (Fig. 5).

The *component* attribute The **component** attribute must match the identifier of a **Component** defined in the same **RBAMacromolecules** instance.

The *stoichiometry* attribute The **stoichiometry** is a positive real number. It represents the stoichiometry of a **Component** (typically how often it appears in a **Macromolecule**, e.g. the number of alanine residues in a protein).

3.6 Examples

The list of components and complexity of macromolecule definitions depends on the desired level of detail. In real models, we include a large list of possible components including all amino acid residues and cofactors for proteins (Fig. 6). In processes.xml, we will define how each component is built from metabolites. In particular, a protein can only be built if all components can be assembled, including cofactors. For example, BSU29470 cannot be produced in the absence of magnesium. Note that in this model, we consider that all amino acid residues have a weight of 1, while cofactors have a weight of 0. This is a simplifying assumption that enables us to estimate the overall weight of proteins in the model. We consider that this level of granularity is sufficient to define density constraints at a satisfactory resolution.

In our minimal model (Fig. 7), we consider that all proteins are made of a single component that more or less represents all amino acid residues, there is no cofactor. We

only define two proteins with different lengths for illustration purposes. In comparison, the real model contains several thousands of proteins.

4 enzymes.xml

4.1 Rationale

The enzyme file is used to define enzyme composition and their catalytic efficiency. It defines efficiency constraints in the RBA model. These constraints ensure that a reaction flux is smaller than the product of efficiency and concentration of the enzyme catalyzing the reaction.

So far, we have defined two types of basic molecules: metabolites (in metabolism.xml) and macromolecules (e.g. in proteins.xml). There is a third type of molecule in RBA models called a *molecular machine*.

Molecular machines are composed of metabolites and macromolecules. Contrary to macromolecules, who share a small pool of components and undergo complex assembly processes, molecular machines can be composed of any metabolite or macromolecule, and their assembly is described by a single reaction. Molecular machines also have a functional role within the cell: catalyzing metabolic reactions or assisting the assembly of macromolecules.

Enzymes are the simplest molecular machines in RBA models, their role is to catalyze metabolic reactions (as defined in metabolism.xml). Process machines are the other molecular machines in RBA models, their composition is defined identically to enzymes, but characterizing their role in the assembly of macromolecules, although similar to the catalysis description, is significantly more complex (as we will see in processes.xml).

The definition of enzymes contains three parts: the composition of the enzyme, the reaction the enzyme catalyzes, and the forward and backward efficiencies of the enzyme. Only one enzyme can be associated with a reaction: if a reaction may be catalyzed by several enzyme, it must be duplicated so that every enzyme can be associated with a separate reaction.

4.2 RBAEnzymes

The outermost part of the metabolism file is an instance of class **RBAEnzymes**, shown in Figure 8.

RBAEnzymes has no simple attributes. It contains excatly one instance of **ListOfEnzymes** that is used to store **Enzyme** information.

4.3 Enzyme

The **Enzyme** class is used to define enzymes (Fig. 8).

It contains a **MachineryComposition** that refers to metabolic **Species** and **Macromolecule**s composing the **Enzyme**. Note that the composition can be left unspecified.

Figure 6: proteins.xml from the model automatically generated by RBApy for the model bacteria *B. subtilis*. Large chunks of the files were removed for brevity. The file contains two lists. The component list contains construction blocks for proteins: amino acid residues and cofactors such as ions and vitamins. The macromolecule list defines all the proteins in the system. Here we show two examples of cytosolic proteins, defined as an ensemble of amino acid residues and cofactors.

```
<RBAProteins>
  <listOfComponents>
    <component id="protein_component_residue" name="Protein residue" type="residue" weight="1"/>
  </listOfComponents>
  <listOfMacromolecules>
    <macromolecule id="small_protein" compartment="cytosol">
      <composition>
        <componentReference component="protein_component_residue" stoichiometry="10"/>
      </composition>
    </macromolecule>
    <macromolecule id="large_protein" compartment="cytosol">
      <composition>
        <componentReference component="protein_component_residue" stoichiometry="20"/>
      </composition>
    </macromolecule>
  </listOfMacromolecules>
</RBAProteins>
```

Figure 7: proteins.xml from the minimal model. In the minimal model, we do not detail every amino acid residue, we just consider that proteins contain different amounts of a "protein_component_residue".

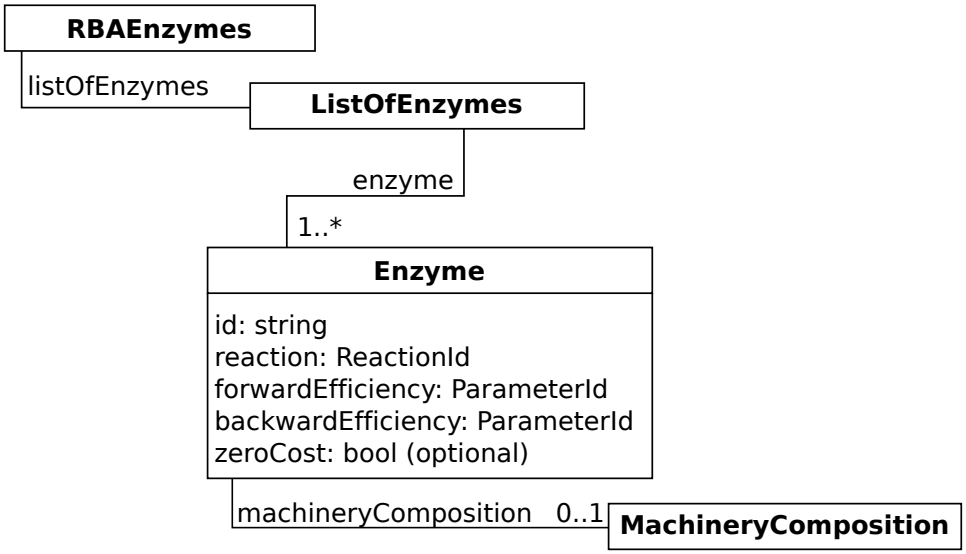

Figure 8: XML structure of enzyme document.

In this case, the reaction associated with the enzyme is considered spontaneous.

The *id* attribute The **id** attribute is a string defining the identifier of the enzyme.

The *reaction* attribute The **reaction** attribute must match the identifier of a metabolic **Reaction**. It represents the reaction catalyzed by the enzyme. This must be a one-to-one mapping. A **Reaction** can only have one associated **Enzyme**. If several **Enzyme**s catalyze the same **Reaction**, the **Reaction** must be duplicated.

The *forwardEfficiency* attribute The **forwardEfficiency** attribute must match the identifier of a parameter (**Function** or **Aggregate**). It represents the forward catalytic constant.

The *backwardEfficiency* attribute The **backwardEfficiency** attribute must match the identifier of a parameter (**Function** or **Aggregate**). It represents the backward catalytic constant (only applicable if reaction catalyzed by enzyme is reversible).

The *zeroCost* attribute The **zeroCost** attribute is a boolean value. If set to true, the reaction associated may occur without having to produce the enzyme. If set to false or unspecified, an efficiency constraint is created where the flux through the reaction has to be smaller than the product of enzyme efficiency and enzyme concentration.

4.4 Examples

A typical enzymes.xml contains a long list of enzymes (Fig. 9), typically one enzyme per metabolic reaction (Fig. 10). If a reaction occurs spontaneously, it is not technically necessary to associate a reaction with it, but RBApy creates an enzyme none-the-less, showing that the reaction was correctly identified as spontaneous.

The **MachineryComposition** is identical to a metabolic reaction, except that it may contain macromolecules, not just metabolites. In the two examples, enzymes are composed of proteins, with identifiers defined in proteins.xml. The reaction could also have included byproducts by defining a `ListOfProducts`. A typical example would have been the inclusion of GTP as a reactant and GDP as a byproduct of the assembly of some enzymes, but we neglected these costs in our models.

Note that the efficiencies are *not* numerical values: they are parameter identifiers, all parameters of the model being defined in parameters.xml. Also note that `zero_cost` is a flag that is almost always set to false. Setting it to true effectively removes the enzyme, making the associated reaction spontaneous. It was included for retrocompatibility with early RBA versions, where it was used to counterbalance numerical instabilities.

```
<RBAEnzymes>
  <listOfEnzymes>
    <enzyme id="R_DM_4crsol_c_enzyme" reaction="R_DM_4crsol_c" forward_efficiency="default_efficiency" backward_efficiency="default_efficiency" zeroCost="false"/>
    <enzyme id="R_DM_5drib_c_enzyme" reaction="R_DM_5drib_c" forward_efficiency="default_efficiency" backward_efficiency="default_efficiency" zeroCost="false"/>
    <enzyme id="R_DM_aacald_c_enzyme" reaction="R_DM_aacald_c" forward_efficiency="default_efficiency" backward_efficiency="default_efficiency" zeroCost="false"/>
    <enzyme id="R_DM_amob_c_enzyme" reaction="R_DM_amob_c" forward_efficiency="default_efficiency" backward_efficiency="default_efficiency" zeroCost="false"/>
                                                                    ⋮
    <enzyme id="R_ACHBS_enzyme" reaction="R_ACHBS" forward_efficiency="default_efficiency" backward_efficiency="default_efficiency" zeroCost="false">
      <machineryComposition>
        <listOfReactants>
          <speciesReference species="b3670" stoichiometry="1.0"/>
          <speciesReference species="b3671" stoichiometry="1.0"/>
        </listOfReactants>
      </machineryComposition>
    </enzyme>
    <enzyme id="R_ACHBS_2_enzyme" reaction="R_ACHBS_2" forward_efficiency="default_efficiency" backward_efficiency="default_efficiency" zeroCost="false">
      <machineryComposition>
        <listOfReactants>
          <speciesReference species="b0077" stoichiometry="1.0"/>
          <speciesReference species="b0078" stoichiometry="1.0"/>
        </listOfReactants>
      </machineryComposition>
    </enzyme>
                                                                    ⋮
```

Figure 9: enzymes.xml from a model automatically generated by RBApy for the model bacteria *E. coli*. Large chunks of the files were removed for brevity. The list starts with 4 enzymes with no composition, indicating that the associated reactions occur spontaneously. The last two enzymes have different compositions but are associated with identical metabolic reactions: either of these enzyme may be produced by the cell to catalyze the reaction.

```
<RBAEnzymes>
  <listOfEnzymes>
    <enzyme id="R_transport_enzyme" reaction="R_transport" forward_efficiency="kcat_transport" backward_efficiency="zero" zeroCost="false">
      <machineryComposition>
        <listOfReactants>
          <speciesReference species="large_protein" stoichiometry="2"/>
        </listOfReactants>
      </machineryComposition>
    </enzyme>
    <enzyme id="R_protein_precursor_enzyme" reaction="R_protein_component_precursor" forward_efficiency="kcat_precursor" backward_efficiency="zero"
    zeroCost="false">
      <machineryComposition>
        <listOfReactants>
          <speciesReference species="small_protein" stoichiometry="2"/>
        </listOfReactants>
      </machineryComposition>
    </enzyme>
    <enzyme id="R_biomass_enzyme" reaction="R_biomass" forward_efficiency="kcat_biomass" backward_efficiency="zero" zeroCost="false">
      <machineryComposition>
        <listOfReactants>
          <speciesReference species="small_protein" stoichiometry="1"/>
          <speciesReference species="large_protein" stoichiometry="1"/>
        </listOfReactants>
      </machineryComposition>
    </enzyme>
  </listOfEnzymes>
</RBAEnzymes>
```

Figure 10: enzymes.xml from the minimal model. We associate one enzyme with all 3 reactions defined in metabolism.xml. Enzymes are defined by using the proteins that we defined in proteins.xml.

5 processes.xml

5.1 Rationale

The process file is used to define cellular processes involved in the production or degradation of macromolecules. In proteins.xml, rnas.xml and dna.xml, we have described proteins in terms of components (amino acid residues, nucteotides, protein cofactors). processes.xml defines how components are assembled from metabolites and what machines are needed to catalyze the assembly.

A typical example of a cellular process is protein translation. Translation relies on ribosomes as a molecular machine, and participates in protein production. The metabolites used during proteins assembly include charged and uncharged-tRNAs and GTP/GDP, e.g. an alanine residue is obtained by using a charged tRNA-alanine and GTP as reactants, generating an uncharged tRNA, GDP and water as a byproduct.

Theoretically, the translation of a protein could be written down as a single reaction, listing all charged-/uncharged-tRNAs, GTP/GDP, according to the SBML format. In a model containing 1000 proteins, we would have to write 1000Âăreactions that follow the same template. Now, if we add other processes undergone by proteins (translation, chaperoning, translocation), we would have to add more reactions per protein to account for the fact that each process is catalyzed by the appropriate molecular machine such as an enzyme or a ribosome. For example, if all proteins need chaperones for folding, we need to add 1000 reactions for chaperoning to the pre-existing 1000 reactions for reaction synthesis. If we add translocation/transport of proteins to target locations, that's 1000 more reactions. If we update the template of one of the process, e.g. we revise the number of GTPs that are consumed per amino acid for protein synthesis, we need to rewrite all 1000 reactions describing protein synthesis, which is an extremely error-prone process.

RBA **Process**es can be seen as template reactions that specify how **Macromolecule**s are produced and degraded based on their **Component**s. Instead of writing one reaction per protein, we write one generic reaction per process, and proteins are seen as input of these processes. Every cellular process is described by (1) the molecular machine that catalyzes the process (e.g. the ribosome), (2) the list of macromolecules to be processed or produced (e.g. proteins), (3) the processing costs (e.g. tRNAs and GTP consumed to assemble an amino acid), and (4) the efficiency of the molecular machine in catalyzing the process, which corresponds to the rate of the process per amount of process machinery (e.g. translation rate for ribosomes in amino acids per hour).

The processing costs are defined by a **ProcessingMap**, which explicits how every **Component** is processed, i.e. what **Species** are consumed and what **Species** are produced as byproducts of the assembly process. For example, in order to assemble an Alanine residue, a charged alanine-tRNA is consumed along with GTP, and an uncharged alanine-tRNA and GDP are generated as byproducts. **ProcessingMap**s define one reaction per **Component**, frow which the overall production/degradation reactions of **Macromolecule**s can be deduced.

Processes are flexible: **Macromolecule**s can be listed as input of several processes.

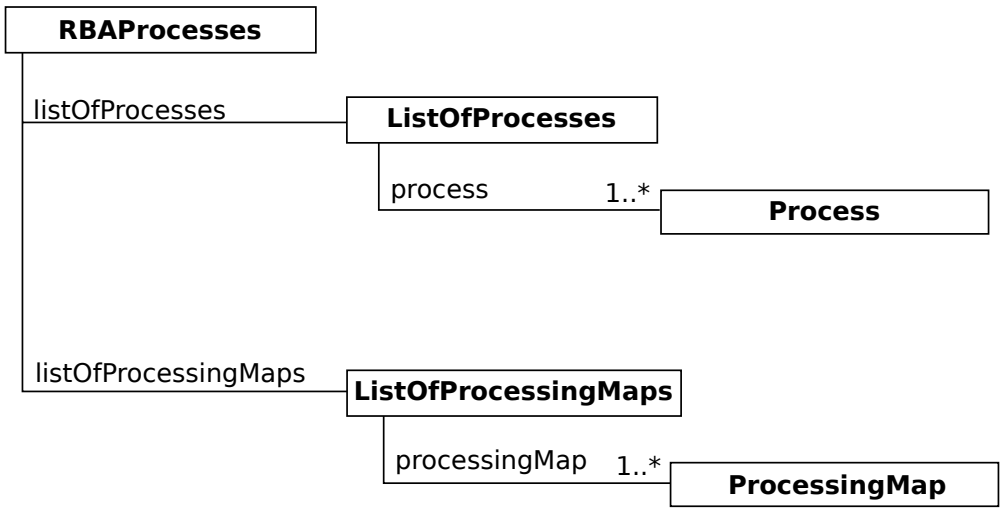

Figure 11: XML structure of process document.

For example, proteins may undergo translation, folding and translocation. The overall production/degradation reaction of a **Macromolecule** reflects all the **Process**es it traverses. When we modify a process, e.g. we update the number of GTPs consumed per amino acid, it automatically affects the production of all input macromolecules.

5.2 RBAProcesses

The outermost part of the process file is an instance of class **RBAProcesses**, shown in Figure 11.

RBAProcesses has no simple attributes. It contains exactly one instance of **ListOfProcesses** and **ListOfProcessingMaps**.

5.3 Process

The **Process** class is used to define cellular processes (Fig. 12).

A **Process** revolves around 2 optional substructures. The **Machinery** is the molecular entity enabling the process (*e.g.* ribosome for translation.) Each machinery unit has a limited production/degradation capacity. Every macromolecule produced by a process has a metabolite cost (metabolites needed to produce/degrade it and byproducts). However, if a machinery is defined, there is an additional cost to produce the machinery that will enable the production/degradation of the target. This is similar to the production of **Enzyme**s in order to catalyze metabolic **Reaction**s.

Processings define the sets of macromolecules that a process produces or degrades. The production reaction of a **Macromolecule** is determined by the **Process**es it goes through. For example, a protein's production reaction is defined by listing the protein as an input in the **Processings** of the translation process. If a protein is not listed as an input of any process, its production reaction is empty, meaning that it does not

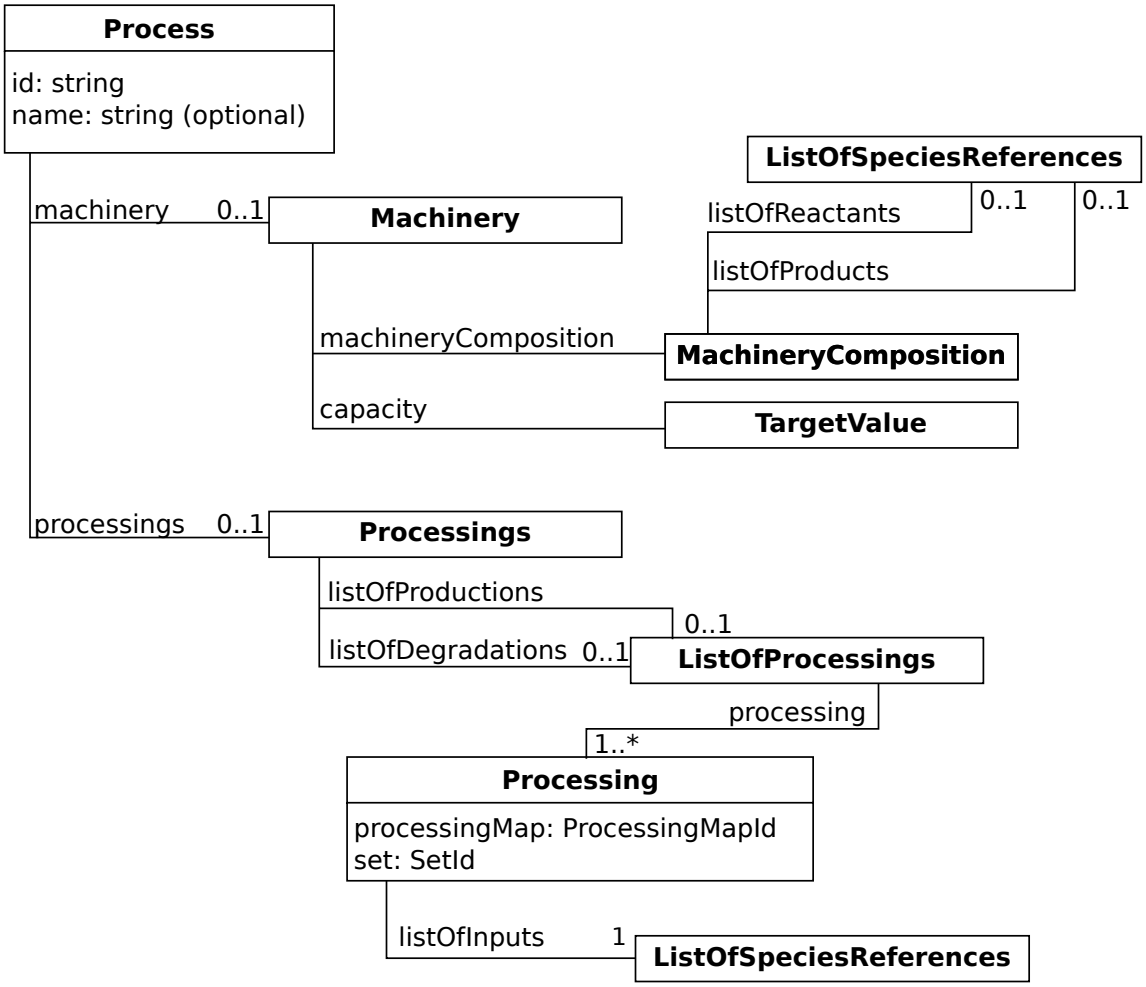

Figure 12: Class used to store processes.

cost anything to produce the protein. **Processings** break down **Macromolecule**s in metabolic **Species** and **Machinery** costs.

The *id* attribute The **id** attribute is a string defining the identifier of a process.

The *name* attribute The **name** attribute is a string that can be used to give the process a more human understandable name.

5.4 Machinery

The **Machinery** class defines the machinery used by a process (Fig. 12). **Machinery** has no simple attributes. If a **Machinery** is defined, it defines a *capacity constraint*. Every **Machinery** unit is produced accordint to the reaction defined by a **MachineryComposition**. Every unit also has a capacity defined by a **TargetValue**. The capacity defines how many targets a **Machinery** can process in 1 unit of time. Total capacity (base capacity multiplied by number of **Machinery** units) must always exceed the number of targets produced.

5.5 MachineryComposition

The **MachineryComposition** class defines the assembly of a complex molecular machinery (Fig. 12). **MachineryComposition** has no simple attributes. It contains two **ListOfSpeciesReferences**. One is for reactants, the other for potential byproducts of the assembly reaction of the complex itself (e.g. GDP when connecting ribosomal subunits). Note that in this case, **SpeciesReference**s can refer to *both* metabolic **Species** and **Macromolecule**s. The assembly reaction should contain obvious components of the machinery, but also metabolic costs related to assembly (such as ATP/GTP costs) *unless* these costs are already covered by a process.

5.6 Processings

The **Processings** relates **Macromolecule**s production/degradation to some given **Process**es (Fig. 12). **Processings** has no simple attributes. It may contain two **ListOfProcessings**, one for production and one for degradation.

5.7 Processing

The **Processing** class defines how **Macromolecule**s are produced/degraded (Fig. 12). **Processing** is used to break down **Macromolecule**s into metabolites by linking them to a **ProcessingMap**. It contains one **ListOfSpeciesReferences** that lists **Macromolecule**s that are inputs of this process. In this context, the species of a **SpeciesReference** must be a macromolecule and the stoichiometry attribute is ignored.

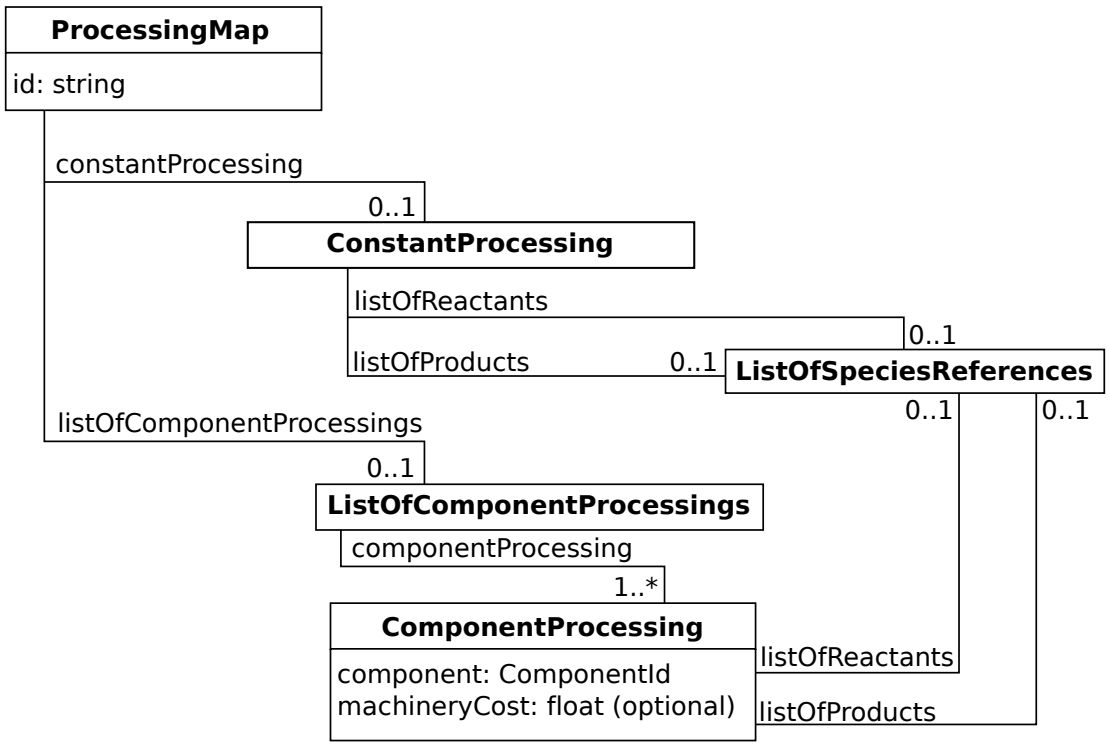

Figure 13: Class used to compute production/degradation of macromolecules.

The *processingMap* attribute The **processingMap** attribute must match the identifier of a **ProcessingMap**. This **ProcessingMap** will be used to compute the production/degradation reaction of **Macromolecule**s, as well as **Machinery** costs.

The *set* attribute The **set** attribute must refer to a **Macromolecule** set. Currently, the only acceptable values are **protein**, **rna** and **dna**. **Macromolecule**s that are listed as input must belong to this set.

5.8 ProcessingMap

The **ProcessingMap** class is used to convert **Macromolecule**s in metabolic and machinery costs (Fig.13).

There are two types of processings. The **ConstantProcessing** lists metabolites that are always consumed or produced when processing a macromolecule, no matter its composition (*e.g.* translation initiation). The **ListOfComponentProcessing** container details **ComponentProcessing**s depending on the individual **Component**s of the **Macromolecule**. They cover metabolites used to assemble the **Component** onto the nascent **Macromolecule**. They also cover machinery costs, *i.e.* how many **Machinery** units are needed to assemble the **Component**.

The *id* attribute The **id** attribute is a string defining the identifier of a processing map.

5.9 ConstantProcessing

The **ConstantProcessing** class defines metabolites consumed and byproducts generated by an assembly process (Fig.13). It contains two **ListOfSpeciesReferences**, one for metabolites consumed and one for metabolites produced. Note that in this context, a **SpeciesReference** must refer to a metabolic **Species**.

5.10 ComponentProcessing

The **ComponentProcessing** class defines metabolites consumed and byproducts generated when assembling a specific **Component** (Fig.13). It contains two **ListOfSpeciesReferences**, one for metabolites consumed and one for metabolites produced. Note that in this context, a **SpeciesReference** must refer to a metabolic **Species**. Additionally, it defines a machinery cost used in a **Machinery**'s capacity constraint.

The *component* attribute The **component** attribute is a string that must match the identifier of a **Component**.

The *machineryCost* attribute The **machineryCost** attribute is a real value that is used to compute how many **Machinery** units are needed to assemble the **Component**. For example, let the machinery cost for the processing of an amino acid be 1. The capacity of the **Machinery** (the ribosome) is the number of amino acids it can assemble per unit of time. The machinery cost allows to compute how many ribosomes are needed to produce the **Component** and, in the end, the **Macromolecule** (in this example the number of amino acids divided by the ribosome's capacity).

5.11 Examples

The best example to illustrate processes is protein translation (Fig. 14). The XML structures are quite long, but processes depend on 3 relatively simple substructures. First, we define the process machine. The definition is reminiscent of enzymes: a machine composition, here all ribosome components (proteins and RNAs), and a catalytic rate, here expressed in number of amino acids per hour. Second, we list all macromolecules that undergo the process, here all proteins. Finally, the processing map, containing one reaction per macromolecule component. The question this structure answers is the following: my macromolecule contains a component named "alanine residue", how do I produce it from metabolites? Is the process machine involved? The answers are: for every alanine residue, you consume a charged t-RNA, water and GTP, by-producing an uncharged t-RNA, GDP, phosphate and protons. The machine cost is an additive cost that is related to the capacity defined earlier. For translation, we define the machine cost for every amino acid to be 1. In the end the total production cost of a protein is

its number of amino acids. We defined the ribosome capacity in terms of amino acid per hour, say 1000. If a protein has 500 amino acids, a single ribosome will be able to produce two copies of this protein per hour.

The machine costs define a capacity constraint for the cell: in order to be viable, the cell needs not only to produce essential macrocomponents, but also the machines that assemble these macrocomponents. Production of machines cannot be neglected, as they increase very rapidly. When the production rate of macromolecule increases, the amount of machines needed for production increases in a seemingly linear fashion. However, machines are themselves composed of macromolecules (Fig. 15), and you need more machines to assemble those. Roughly speaking, if you want more proteins, you need to produce more ribosomes. But if you want more ribosoms, you need ribosomes to produce the new ribosomes. In the end the amount of machines increases more than linearly, imposing strong constraints on the cell.

6 density.xml

The density file contains density constraints for the RBA model.

6.1 Rationale

A cell can only contain a limited number of macromolecules: there is a threshold that the total density (weight per unit volume) cannot exceed. Every macromolecule has a weight that equals the sum of its components' weights:

$$W_i = \sum_{c \in \text{components}} w_c$$

If we assign every macromolecule a concentration C_i, the total density is:

$$\sum_{i \in [1..M]} C_i W_i$$

where M is the total number of macromolecules.

There are two types of density contraints. In the first case, the density *must* be equal to some density d:

$$\sum_{i \in [1..M]} C_i W_i = d$$

Alternatively, the density must not exceed some maximal density d_{max}

$$\sum_{i \in [1..M]} C_i W_i \leq d_{max}$$

The XML format allows for equality and inequality constraints by specifying whether the bound is a lower bound (not applicable for density), an upper bound (like d_{max}), or a set value (like d).

Figure 14: processes.xml from the hand curated model for model bacteria *B. subtilis*. Large chunks of the files were removed for brevity. This example focuses on some aspects of the definition of the translation process. The first element that is defined is the process machine: the ribosome. The machine is defined as a single assemble reaction from macromolecules (ribosomal proteins and rRNAs) and metabolites (GTP needed to assemble ribosomal proteins and rRNAs). Process machines have a catalytic rate termed capacity: for the ribosome, this is the number of amino acids processed per hour, later defined by the parameter P_TA_capacity. Finally the process contains the list of macromolecules to produce or degrade, here all the proteins in the model. A processing map explicits how macromolecules are produced from their components. Here, the processing map starts by defining metabolites that are consumed and produced independent of protein composition (translation initiation). We also show how an alanine residue is built from metabolites.

```
<RBAProcesses>
  <listOfProcesses>
    <process id="translation" name="Translation process">
      <machinery>
        <machineryComposition>
          <listOfReactants>
            <speciesReference species="small_protein" stoichiometry="1"/>
            <speciesReference species="large_protein" stoichiometry="1"/>
          </listOfReactants>
        </machineryComposition>
        <capacity value="ribosome_capacity"/>
      </machinery>
      <processings>
        <listOfProductions>
          <processing processingMap="translation_map" set="protein">
            <listOfInputs>
              <speciesReference species="small_protein" stoichiometry="1"/>
              <speciesReference species="large_protein" stoichiometry="1"/>
            </listOfInputs>
          </processing>
        </listOfProductions>
      </processings>
    </process>
  </listOfProcesses>
  <listOfProcessingMaps>
    <processingMap id="translation_map">
      <constantProcessing>
        <listOfReactants>
          <speciesReference species="M_carbon_source_c" stoichiometry="1"/>
        </listOfReactants>
      </constantProcessing>
      <listOfComponentProcessings>
        <componentProcessing component="protein_component_residue" machineryCost="0">
          <listOfReactants>
            <speciesReference species="M_protein_component_precursor_c" stoichiometry="1"/>
            <speciesReference species="M_carbon_source_c" stoichiometry="1"/>
          </listOfReactants>
        </componentProcessing>
      </listOfComponentProcessings>
    </processingMap>
  </listOfProcessingMaps>
</RBAProcesses>
```

Figure 15: processes.xml from the minimal model. We only define one process corresponding to pseudo-translation. It contains a two-protein machine with a capacity that we will define later. We list all the proteins from the model as input of translation. Proteins only had one component so the processing map is short. We define a constant cost corresponding to translation initiation and metabolites consumed for every residue that has to be assembled.

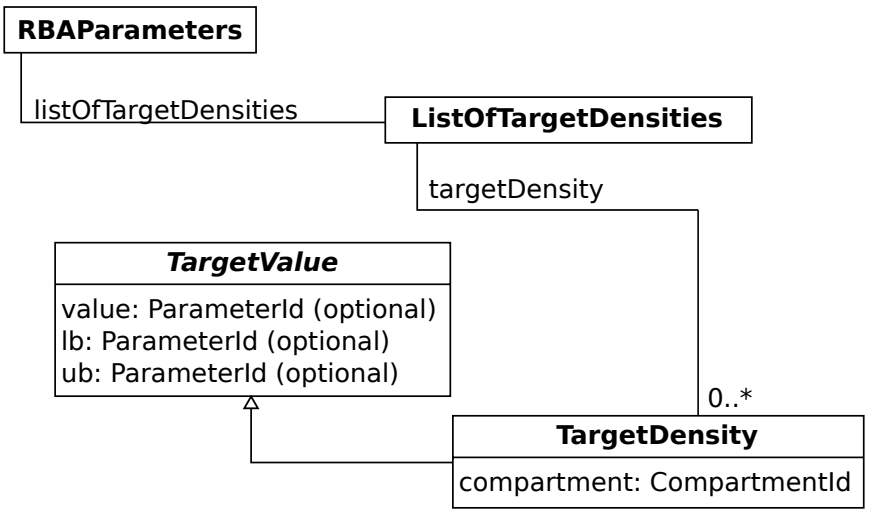

Figure 16: XML structure of density document.

6.2 RBADensity

The outermost part of the density file is an instance of class **RBADensity**, shown in Figure 16.

RBADensity has no simple attributes. It includes exactly one instance of **ListOf** container classes. All **ListOf** classes do not have own attributes, they are merely used to organize a list of instances from another class.

6.3 TargetDensity

The **TargetDensity** class is used to define density constraints (Fig. 16). In a RBA model, a density constraint defines how many molecules a given compartment can contain. It inherits **TargetValue** for the constraint definition part.

The *compartment* attribute The **compartment** attribute must match the identifier of a **Compartment**.

6.4 TargetValue

The **TargetValue** class is used to define the sign of an additional RBA constraint and the value of its second member (Fig. 16). It is designed to be inherited. The child class usually holds information about the first member of the constraint (*e.g.* compartment for a density constraint, metabolite for a production constraint).

The *value*, *lowerBound* and *upperBound* attributes Every attribute can be left undefined, or contain the identifier of a **Function** or an **Aggregate**.

```
<RBADensity>
  <listOfTargetDensities>
    <targetDensity upperBound="Cytoplasm_density" compartment="Cytoplasm"/>
    <targetDensity upperBound="Cell_membrane_density" compartment="Cell_membrane"/>
  </listOfTargetDensities>
</RBADensity>
```

Figure 17: density.xml from a hand-curated model for the model bacteria *B. subtils*. The model defines two density constraints, one per compartment. By using upperBound, we define two inquality constraints: the density of macromolecules may not exceed parameter Cytoplasm_density in the cytoplasm or Cell_membrane_density in the membrane, but any lower value is acceptable.

```
<RBADensity>
  <listOfTargetDensities>
    <targetDensity upperBound="maximal_cytosol_density" compartment="cytosol"/>
  </listOfTargetDensities>
</RBADensity>
```

Figure 18: density.xml from the minimal model. There is only one compartment in the minimal model, for which we define an upperBound type density constraint.

If **value** is defined, the constraint is an equally constraint. **lowerBound** and **upperBound** are ignored. If **value** is undefined, **lowerBound** (resp. **upperBound**) defines a lower bound (resp. upper bound) inequality constraint. Note that **lowerBound** and **upperBound** may both be defined, yielding two separate inequality constraints.

6.5 Examples

density.xml is by far the shortest file in an RBA model (Fig. 17 and 18). Usually it contains one constraint per compartment, even though there may be zero or more than one constraint per compartment. In general, we advise using inequality constraints for greater flexibility in the model.

7 targets.xml

The targets file is used to define production and degradation constraints, i.e. fluxes of metabolic **Species** or **Macromolecule**s that must be maintained for the cell to be functional.

7.1 Rationale

In an RBA model, the production of machines (enzymes and process machines) is automatically computed to sustain the metabolic fluxes and the production of macromolecules necessary to achieve optimal growth rate. However, in order to grow, the cell must produce more than just enzymes and ribosomes. These production constraints are

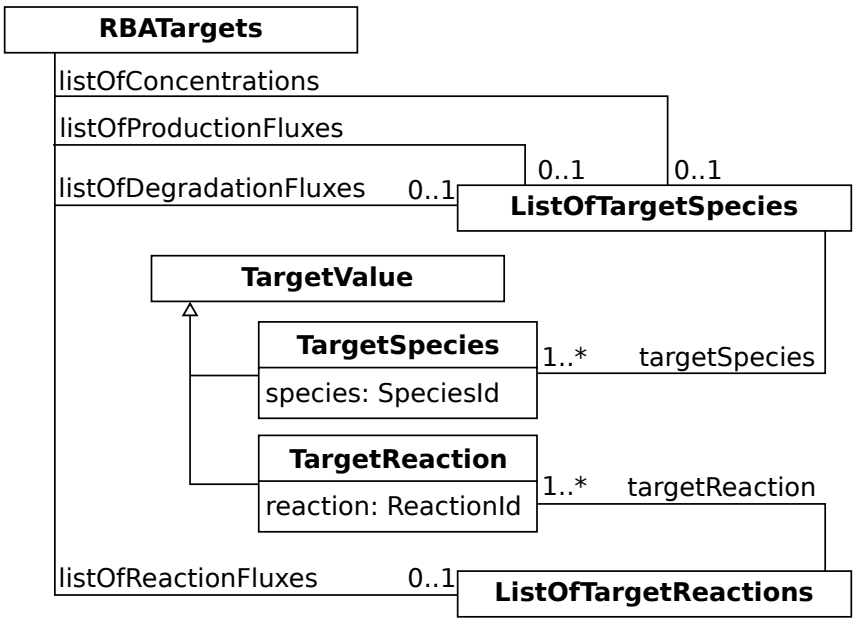

Figure 19: XML structure of target document.

defined in targets.xml as fluxes or concentrations that a cell must maintain in order to be functional.

Typical targets include molecules such as DNA, mRNAs, or housekeeping proteins whose concentration must be maintained at all times. Structural molecules contained in cell wall for example, must also be maintained at a constant concentration. A less obvious example are essential metabolites whose concentration is high and well regulated, such as NADP/NADPH. Defining a target concentration for NADP forces de novo production of NADP, otherwise the model assumes that recycling existing NADP is sufficient to achieve any growth rate. We suggest defining target concentrations for metabolites whenever metabolomic data are available.

In general, targets are defined as concentrations to maintain, but it's also possible to specify absolute fluxes of production for metabolites or reactions. A typical reaction flux that must be maintained is the production of maintenance ATP (ATP used by secondary processes of the cell).

From the XML point of view, the definition of target constraints uses the same structure as the density constraints. In particular, the constraints may be equalities (the concentration must be maintained to that exact value) or inequalities (the concentration must be maintained between these bounds).

7.2 RBATargets

The outermost part of the process file is an instance of class **RBATargets**, shown in Figure 19.

RBATargets has no simple attributes. It contains 3 **ListOfTargetSpecies**. These

targets allow to define metabolic **Species** or **Macromolecule** fluxes. One is for production fluxes, another for degradation fluxes. The last list is for maintaining a target at a given concentration. The difference with a simple production flux is that keeping a target at a concentration depends on growth rate. More precisely, the flux needed to keep the concentration is the growth rate multiplied by the target concentration. Note that all fluxes must be positive. If the target is a **Macromolecule**, production/degradation can only occur if the **Processings** section of some **Process** defines how the **Macromolecule** is actually produced/degraded.

It contains a **ListOfTargetReactions**. It is also possible to define target fluxes as reaction fluxes. These targets add constraints on the flux of a specific metabolic **Reaction**. In this case, fluxes may be positive or negative.

7.3 TargetSpecies

The **TargetSpecies** class defines constraints for a species flux (Fig. 19). It inherits **TargetValue** for the constraint definition part, allowing for equality or inequality constraints.

The *species* attribute The **species** attribute is a string that must match the identifier of a metabolic **Species** or a **Macromolecule**. Note that the **Macromolecule** must be broken down into metabolite costs through the **Processings** section of some **Process**. Otherwise no cost will be applied.

7.4 TargetReaction

The **TargetReaction** class defines constraints for a reaction flux (Fig. 19). It inherits **TargetValue** for the constraint definition part, allowing for equality or inequality constraints.

The *reaction* attribute The **reaction** attribute is a string that must match the identifier of a metabolic **Reaction**.

7.5 Examples

Most targets in an RBA model are concentration targets for DNA production, mRNA production, housekeeping protein production, de novo metabolite production, by specifying a concentration that must be maintained (Fig. 20 and 21).

The second class of targets are fluxes related to degradation. For example, our *B. subtilis* model explicitly accounts for mRNA degradation. As a result, there are 3 target fluxes associated with mRNAs: a flux associated with the target concentration to maintain, a flux indicating how many molecules are degraded by seconds (how mRNAs are degraded is defined in processes.xml), and an absolute production flux that exactly compensates degradation (Fig. 20).

```
<RBATargets>
  <listOfTargetGroups>
    <targetGroup id="translation_targets">
      <listOfConcentrations>
        <targetSpecies value="nonenzymatic_proteins_Cytoplasm" species="average_protein_Cytoplasm"/>
        <targetSpecies value="nonenzymatic_proteins_Secreted" species="average_protein_Secreted"/>
        <targetSpecies value="nonenzymatic_proteins_Cell_membrane" species="average_protein_Cell_membrane"/>
      </listOfConcentrations>
    </targetGroup>
    <targetGroup id="transcription_targets">
      <listOfConcentrations>
        <targetSpecies value="mrna_concentration" species="mrna"/>
      </listOfConcentrations>
      <listOfProductionFluxes>
        <targetSpecies value="mrna_degradation_flux" species="mrna"/>
      </listOfProductionFluxes>
    </targetGroup>
                                                    ⋮
    <targetGroup id="maintenance_atp_target">
      <listOfReactionFluxes>
        <targetReaction lowerBound="maintenance_atp" reaction="R_maintenance_atp"/>
      </listOfReactionFluxes>
    </targetGroup>
    <targetGroup id="flagella_activation">
      <listOfReactionFluxes>
        <targetReaction value="flagella_proton_flux" reaction="Th"/>
      </listOfReactionFluxes>
    </targetGroup>
  </listOfTargetGroups>
</RBATargets>
```

Figure 20: targets.xml from a hand-curated model for the model bacteria *B. subtilis*. Large parts of the files where removed for brevity. The first 3 targetSpecies encompass all nonenzymatic proteins that must be produced by the cell for housekeeping purposes. mRNAs have two production constraints: the first contraint defines a flux that must be generated in order to maintain their concentration, the second constraint defines a flux that must be generated in order to compensate degradation (defined later in the file). Finally, there are two example of reaction targets: a lowerBound for maintainance ATP production, and a set bound for the reaction controlling flagella movement.

The third class of targets are targets that reference reactions instead of metabolic species or macromolecules. Maintenance ATPÂăis one of the most common examples, but there are a lot of contextual requirements that can be written in this form, such as flagella movement, response to stresses such as oxidative stresses causing NADPH consumption.

8 parameters.xml

The parameter file contains user-defined parameters and functions.

```
<RBATargets>
  <listOfTargetGroups>
    <targetGroup id="biomass_production">
      <listOfConcentrations>
        <targetSpecies value="target_biomass_production" species="M_biomass_c"/>
      </listOfConcentrations>
    </targetGroup>
  </listOfTargetGroups>
</RBATargets>
```

Figure 21: targets.xml from the minimal model. We pool all non-machine production requirements in a generic biomass production requirement.

8.1 Rationale

The file parameters.xml contains all numerical values occurring in the model (except for stoichiometries). Nearly all other files refer to this file, as we have seen throughout the examples, where numerical values were defined as parameter identifiers. The most common parameters are: total amino acid concentrations, fractions of protein per compartment, percentages of non-enzymatic protein per compartment and cellular machinery, target fluxes for metabolites and macromolecules, efficiencies of enzymes, transporters and molecular machines.

A parameter may be defined as: a constant, a function of growth rate, or a function of an external metabolite concentrations (defined in medium.tsv). Currently, the format supports the following function types: linear, inverse, exponential and Michaelis-Menten. The function types have been chosen to reflect common biochemical functions. Typically, enzyme efficiencies vary linearly with growth rate and transporters have activities that depend on the concentration of metabolites transported according to a Michaelis-Menten function.

A parameter can also be defined as a product of functions, called an "aggregate". For example, the activity of a transporter can be described as the product of a growth rate-dependent maximal activity and a concentration-dependent Michaelis-Menten term. All parameters are allowed to have growth-rate or concentration dependencies. For instance, maximal densities and target fluxes can be defined as constant or growth rate dependent.

8.2 RBAParameters

The outermost part of the parameter file is an instance of class **RBAParameters**, shown in Figure 22.

RBAParameters has no simple attributes. It includes exactly one instance of **ListOf** container classes. All **ListOf** classes do not have own attributes, they are merely used to organize a list of instances from another class.

8.3 Function

The **Function** class is used for user-defined functions and parameters (Fig. 23). The default variable of a function is the growth rate, but it may also be the extracellu-

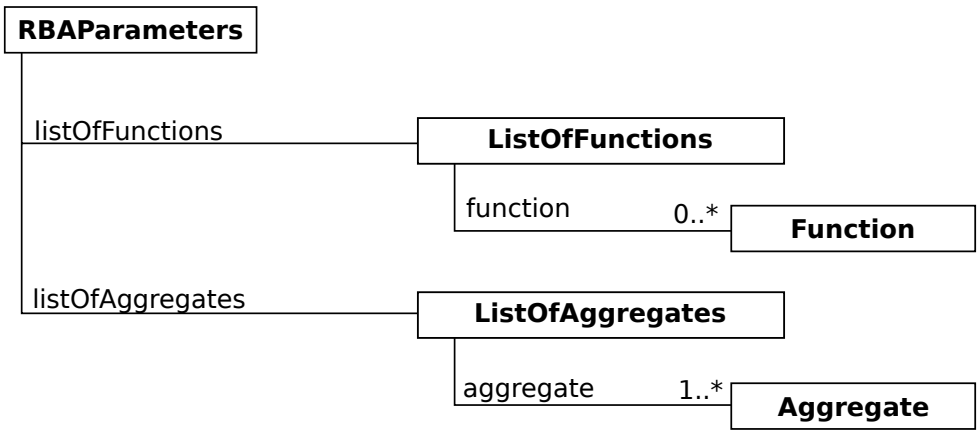

Figure 22: XML structure of parameter document.

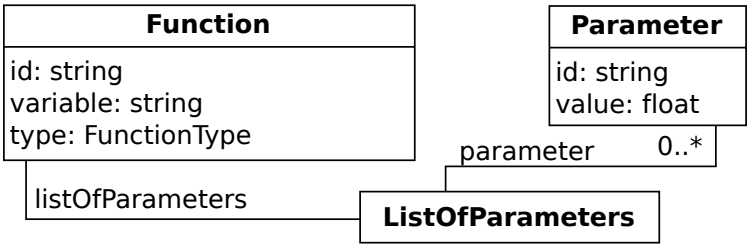

Figure 23: Class used to store user-defined functions.

lar concentration of a metabolite. Every function holds a **ListOfParameters**, where **Parameter** are defined according to each type of function.

The *id* attribute The **id** attribute is a string defining the identifier of the function.

The *variable* attribute The **variable** attribute is a string defining the variable of the function. If empty or set to `growth_rate`, the variable is the current growth rate. Alternatively, the variable may the prefix of a metabolite.

The *type* attribute The **type** attribute is a string that must match a known function type. Currently, the supported types are:

- **constant**. Constant function with parameter *CONSTANT*.
- **linear**. Linear function with parameters *LINEAR_CONSTANT*, *LINEAR_COEF*, *X_MIN*, *X_MAX*, *Y_MIN*, *Y_MAX*. The 4 last parameters are used to saturate the function. The computation is done in three steps. First, if the variable (e.g. growth rate) is outside of the [X_MIN, X_MAX] range, it is set to the closest value in that range. Second, the function is computed. Finally, if the return value is outside of the [Y_MIN, Y_MAX] range, it is set to the closest value in that

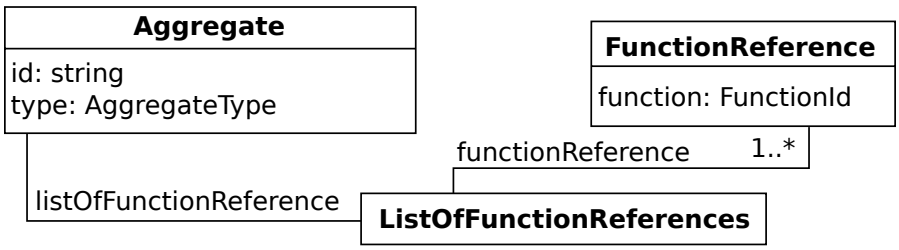

Figure 24: Class used to store user-defined aggregates.

range. The range parameters can be set to infinity by setting them to ("-inf") or ("inf").

- **exponential**. Exponential function with parameter *RATE*.
- **indicator**. Indicator function with parameters *X_MIN* and *X_MAX*. This function returns one if the variable (growth rate) is in the [X_MIN, X_MAX], zero otherwise.
- **michaelisMenten**. Irreversible Michaelis Menten function with parameters *kmax*, *Km* and *Y_MIN* (optional). If Y_MIN is defined, any return value lower than Y_MIN will be set to Y_MIN.

8.4 Parameter

The **Parameter** class is used to store the values of function parameters (Fig. 23).

The *id* attribute The **id** attribute is a string that should match a valid parameter identifier. The list of valid parameters for each type of **Function** is listed above.

The *value* attribute The **value** attribute is a real number representing the value of the attribute.

8.5 Aggregate

The **Aggregate** class is used to assemble user-defined functions (Fig. 24). Every aggregate holds a **ListOfFunctionReferences**, where each **FunctionReference** refers to a previously defined function.

The *id* attribute The **id** attribute is a string defining the identifier of the aggregate.

The *type* attribute The **type** attribute is a string that must match a known aggregate type. Currently, the supported types are:

- **multiplication**. The result is the multiplication of the values returned by the function listed in the aggregate at current growth rate.

8.6 FunctionReference

The **FunctionReference** class is used to refer to a user-defined **Function** (Fig. 24).

The *function* attribute The **function** attribute is a string that must match the identifier of a user-defined **Function**.

8.7 Examples

parameters.xml is one of the longest files in the model as it contains all numerical values in the model. All parameters follow the same rules, no matter whether they define a density constraint, a target concentration or a catalytic activity.

In the first example (Fig. 25), we show the definition of a parameter related to the density constraint, protein_concentration. This parameter reflects measured protein densities, later used to define a growth-dependent maximal density bound. The measures showed that the density of proteins decreases with growth rate: the density bound becomes smaller for larger growth rates. We also show the definition of a transporter catalytic rate. A transporter can be defined in two parts: its base efficiency (potentially growth-rate dependent, here modeled as a constant), and transport factors related to the concentration of the transported molecule and potential cofactors.

In the example from the minimal model (Fig. 26), we see how quickly parameters accumulate: we need parameters for catalytic activities, process machines, target concentrations and density constraints.

```
<RBAParameters>
  <listOfFunctions>
    <function id="protein_concentration" type="linear" variable="growth_rate">
      <listOfParameters>
        <parameter id="LINEAR_COEF" value="-0.0048302"/>
        <parameter id="LINEAR_CONSTANT" value="0.031256"/>
        <parameter id="X_MIN" value="0.25"/>
        <parameter id="X_MAX" value="1.6"/>
        <parameter id="Y_MIN" value="-inf"/>
        <parameter id="Y_MAX" value="inf"/>
      </listOfParameters>
    </function>
                                        ⋮
    <function id="TptsG_base_efficiency" type="constant" variable="growth_rate">
      <listOfParameters>
        <parameter id="CONSTANT" value="1334026.4422"/>
      </listOfParameters>
    </function>
    <function id="TptsG_transport_factor_0" type="indicator" variable="M_glc_e">
      <listOfParameters>
        <parameter id="X_MIN" value="1e-05"/>
        <parameter id="X_MAX" value="inf"/>
      </listOfParameters>
    </function>
    <function id="TptsG_transport_factor_1" type="michaelisMenten" variable="M_glc_e">
      <listOfParameters>
        <parameter id="Km" value="0.8"/>
        <parameter id="kmax" value="1.0"/>
      </listOfParameters>
    </function>
                                        ⋮

  </listOfFunctions>
  <listOfAggregates>
                                        ⋮
    <aggregate id="TptsG_transport_efficiency" type="multiplication">
      <listOfFunctionReferences>
        <functionReference function="TptsG_base_efficiency"/>
        <functionReference function="TptsG_transport_factor_0"/>
        <functionReference function="TptsG_transport_factor_1"/>
      </listOfFunctionReferences>
    </aggregate>
                                        ⋮

  </listOfAggregates>
</RBAParameters>
```

Figure 25: parameters.xml from a hand-curated model for model bacteria *B. subtilis*. Large parts of the file were removed for brevity. Parameters are defined either as constants, functions (e.g. linear or Michealis-Menten), or aggregates, representing the multiplication of functions. Functions can be defined in terms of growth rate (the default) or concentration of external metabolites. Aggregates may contain an arbitrary number of functions, and functions in an aggregate can be defined according to different variables (growth rate, concentration of different metabolites).

```
<RBAParameters>
  <listOfFunctions>
    <function id="kcat_transport_base" type="constant" variable="growth_rate">
      <listOfParameters>
        <parameter id="CONSTANT" value="10"/>
      </listOfParameters>
    </function>
    <function id="kcat_precursor" type="constant" variable="growth_rate">
      <listOfParameters>
        <parameter id="CONSTANT" value="10"/>
      </listOfParameters>
    </function>
    <function id="kcat_biomass" type="constant" variable="growth_rate">
      <listOfParameters>
        <parameter id="CONSTANT" value="10"/>
      </listOfParameters>
    </function>
    <function id="zero" type="constant" variable="growth_rate">
      <listOfParameters>
        <parameter id="CONSTANT" value="0"/>
      </listOfParameters>
    </function>
    <function id="transport_factor" type="michaelisMenten" variable="M_carbon_source_e">
      <listOfParameters>
        <parameter id="Km" value="0.5"/>
        <parameter id="kmax" value="1"/>
      </listOfParameters>
    </function>
    <function id="maximal_cytosol_density" type="constant" variable="growth_rate">
      <listOfParameters>
        <parameter id="CONSTANT" value="1"/>
      </listOfParameters>
    </function>
    <function id="target_biomass_production" type="constant" variable="growth_rate">
      <listOfParameters>
        <parameter id="CONSTANT" value="1"/>
      </listOfParameters>
    </function>
    <function id="ribosome_capacity" type="constant" variable="growth_rate">
      <listOfParameters>
        <parameter id="CONSTANT" value="100"/>
      </listOfParameters>
    </function>
  </listOfFunctions>
  <listOfAggregates>
    <aggregate id="kcat_transport" type="multiplication">
      <listOfFunctionReferences>
        <functionReference function="kcat_transport_base"/>
        <functionReference function="transport_factor"/>
      </listOfFunctionReferences>
    </aggregate>
  </listOfAggregates>
</RBAParameters>
```

Figure 26: parameters.xml from the minimal model. For simplicity, all parameters were defined as constants, except for transport terms, which adopt the traditional Michaelis-Menten.