XML format for RBA models

BioSys group, INRA Jouy, France

September 30, 2018

Contents

1 Introduction

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).
- parameters.xml (definition of user-defined functions).
- density.xml (definition of density constraints).
- 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).
- targets.xml (definition of necessary to growth and maintenance).

A valid RBA model must contain all these files with these exact names in the same directory to be recognized by the RBApy parser. For every file, we present the nodes that compose the XML structure. For every node, we show a class diagram that shows the node's attributes and the children nodes that it may/must contain. We provide a brief description about the relevance of the node in the RBA model.

2 Conventions

2.1 Naming conventions in XML and RBApy

All XML elements described here 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.

2.2 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.

2.3 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.

3 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.

3.1 RBAMetabolism

The outermost portion of the metabolism file is an instance of class **RBAMetabolism**, shown in Figure 1.

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.

3.2 Compartment

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

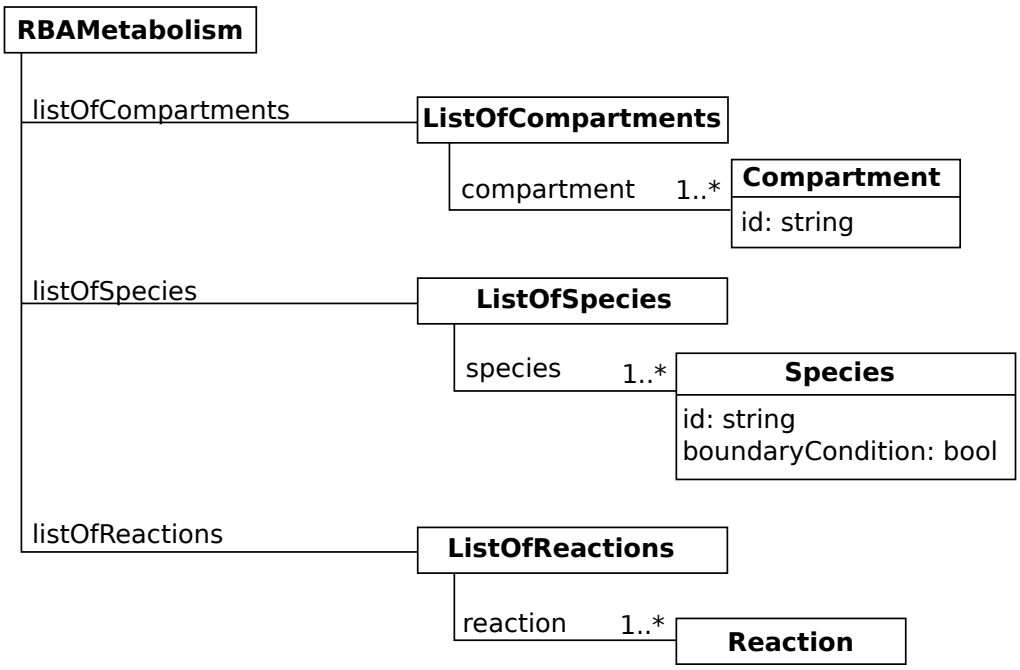

Figure 1: XML structure of metabolism document.

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

3.3 Species

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

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.

3.4 Reaction

The **Reaction** class is used to define metabolic reactions (Fig. 2). Reactants and products are defined using a **ListOfSpeciesReferences**.

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

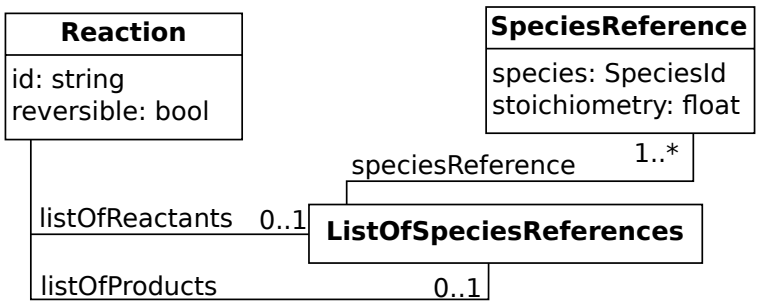

Figure 2: Class storing metabolic reactions.

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.

3.5 SpeciesReference

The **SpeciesReference** class is used to refer to a metabolic **Species** and associate with it a stoichiometry (Fig. 2).

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**).

4 parameters.xml

The parameter file contains user-defined parameters and functions.

4.1 RBAParameters

The outermost portion of the parameter file is an instance of class **RBAParameters**, shown in Figure 3.

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.

4.2 Function

The **Function** class is used for user-defined functions and parameters (Fig. 4). The default variable of a function is the growth rate, but it may also be the extracellular concentration of a metabolite. Every function holds a **ListOfParameters**, where **Parameter** are defined according to each type of function.

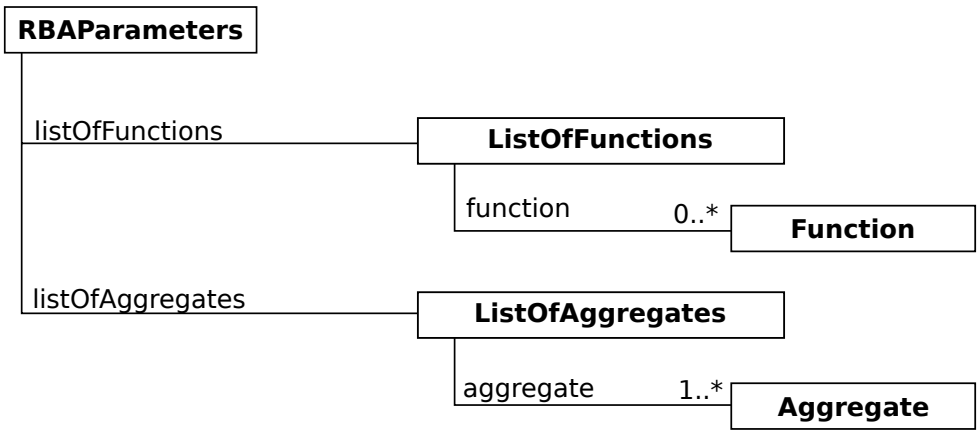

Figure 3: XML structure of parameter document.

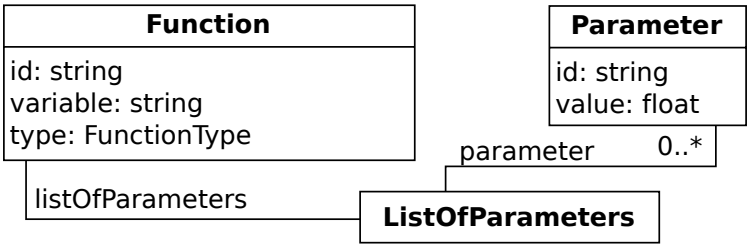

Figure 4: Class used to store user-defined functions.

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 range. The range parameters can be set to infinity by setting them to `("-inf")` or `("inf")`.
- **exponential**. Exponential function with parameter *RATE*.

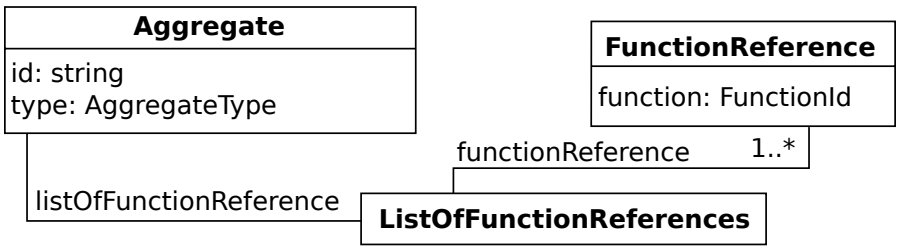

Figure 5: Class used to store user-defined aggregates.

- **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.

4.3 Parameter

The **Parameter** class is used to store the values of function parameters (Fig. 4).

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.

4.4 Aggregate

The **Aggregate** class is used to assemble user-defined functions (Fig. 5). 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.

4.5 FunctionReference

The **FunctionReference** class is used to refer to a user-defined **Function** (Fig. 5).

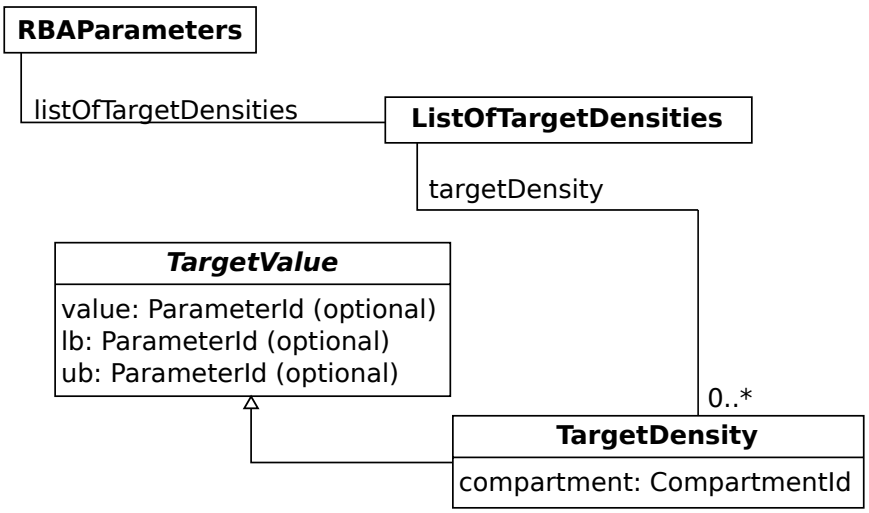

Figure 6: XML structure of density document.

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

5 density.xml

The density file contains density constraints for the RBA model.

5.1 RBADensity

The outermost portion of the density file is an instance of class **RBADensity**, shown in Figure 6.

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.

5.2 TargetDensity

The **TargetDensity** class is used to define density constraints (Fig. 6). 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**.

5.3 TargetValue

The **TargetValue** class is used to define the sign of an additional RBA constraint and the value of its second member (Fig. 6). 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*, *lb* and *ub* attributes Every attribute can be left undefined, or contain the identifier of a **Function** or an **Aggregate**.

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

6 proteins.xml, rnas.xml and dna.xml

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

6.1 RBAMacromolecules

The outermost portion of the protein, RNA and DNA files is an instance of class **RBAMacromolecules**, shown in Figure 7.

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

6.2 Component

The **Component** class is used to define the components of a **Macromolecule** (Fig. 7). 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 **Species**. 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.

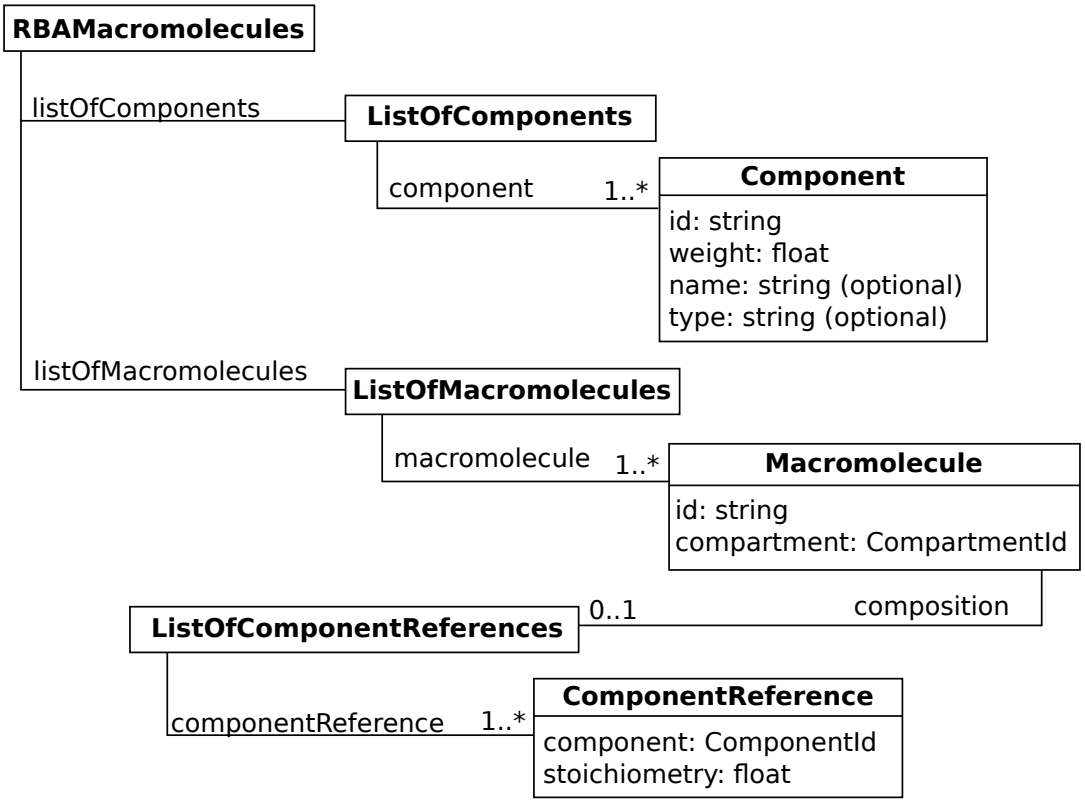

Figure 7: XML structure of macromolecule document.

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).

6.3 Macromolecule

The Macromolecule class is used to define macromolecular species (Fig. 7). 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.

6.4 ComponentReference

The ComponentReference class is used to refer to a Component and associate with it a stoichiometry (Fig. 7).

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).

7 processes.xml

The process file is used to define cellular processes involved in the production or degradation of macromolecules. Processes can be seen as template reactions that specify how Macromolecules are produced and degraded based on their Components. A typical example is translation: a Machinery (the ribosome) assembles proteins based on their amino acid sequence (Components). Every amino acid is assembled using metabolic Species, in particular tRNAs. However, the Species that are used differ from amino acid to amino acid. The ProcessingMap explicits how every Component is processed, i.e. what Species are consumed (e.g. charged tRNAs, GTP) and what Species are produced (e.g. uncharged tRNAs, GDP). Briefly put, Processes define one reaction per Component(), frow which the production/degradation reactions of all Macromolecules can be deduced.

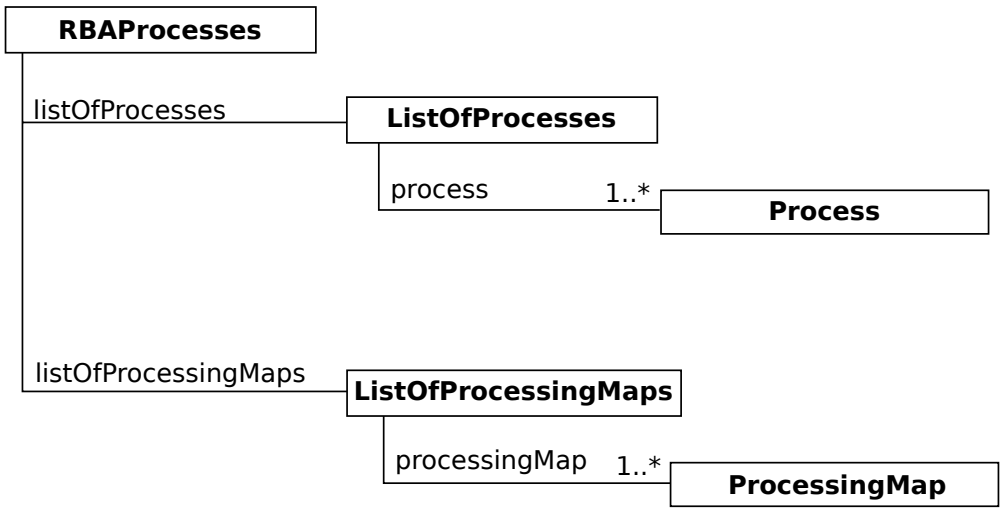

Figure 8: XML structure of process document.

Macromolecules can go through several **Process**es: for example, proteins may undergo translation, folding and translocation. The **Processings** element is used to specify what macromolecule goes through which process. The overall production/degradation reaction of a **Macromolecule** reflects all the **Process**es it traverses.

7.1 RBAProcesses

The outermost portion of the process file is an instance of class **RBAProcesses**, shown in Figure 8.

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

7.2 Process

The **Process** class is used to define cellular processes (Fig. 9).

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

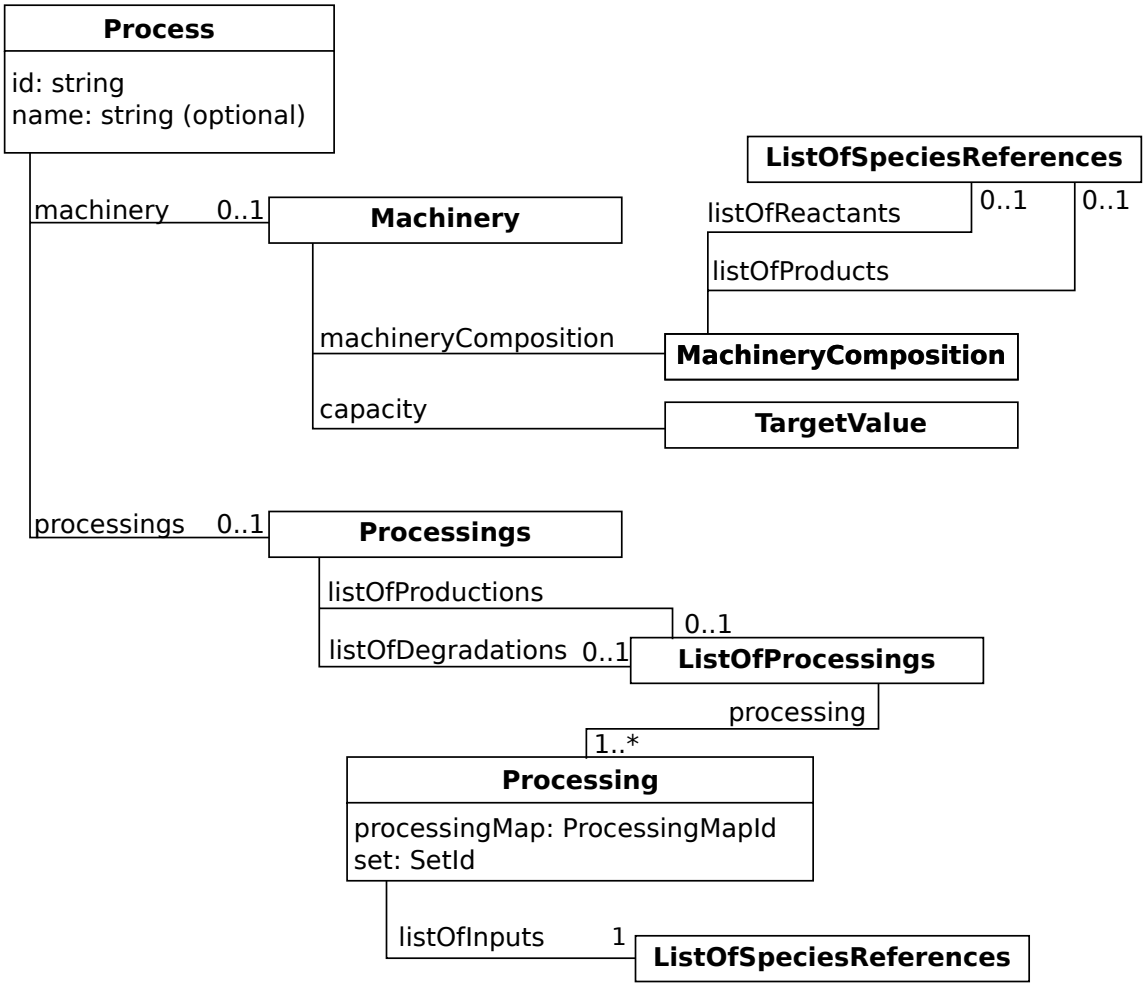

Figure 9: Class used to store processes.

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.

7.3 Machinery

The **Machinery** class defines the machinery used by a process (Fig. 9). **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.

7.4 MachineryComposition

The **MachineryComposition** class defines the assembly of a complex molecular machinery (Fig. 9). **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.

7.5 Processings

The **Processings** relates **Macromolecule**s production/degradation to some given **Process**es (Fig. 9). **Processings** has no simple attributes. It may contain two **ListOfProcessings**, one for production and one for degradation.

7.6 Processing

The **Processing** class defines how **Macromolecule**s are produced/degraded (Fig. 9). **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.

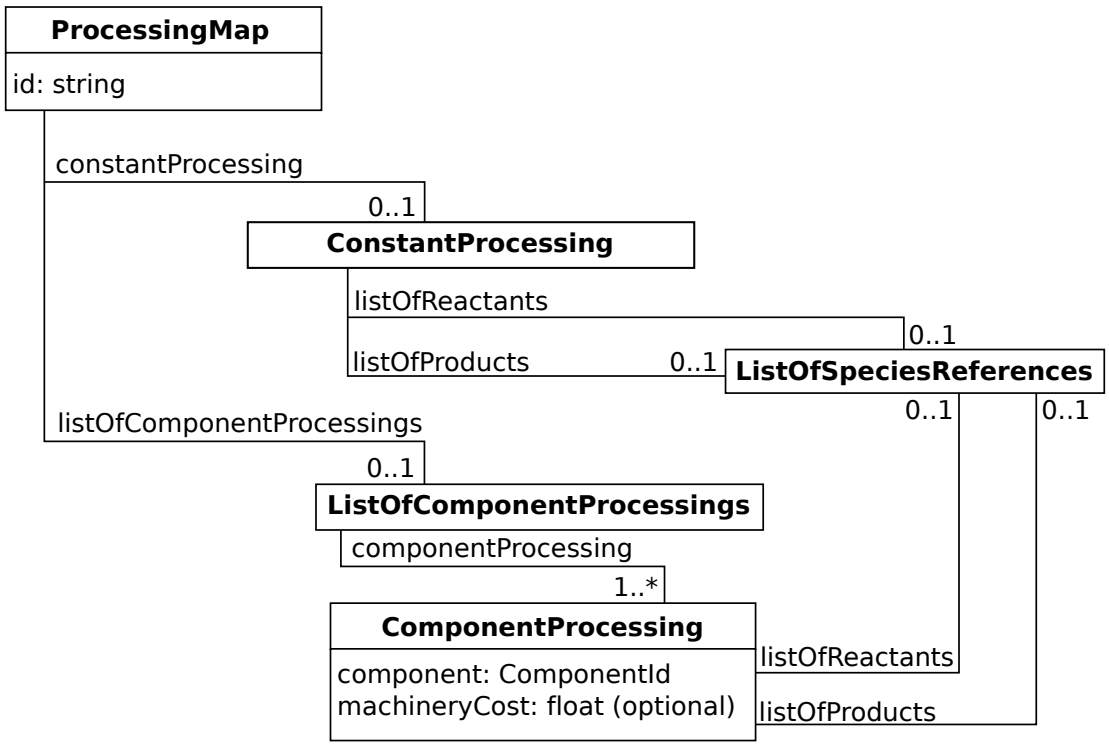

Figure 10: Class used to compute production/degradation of macromolecules.

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.

7.7 ProcessingMap

The **ProcessingMap** class is used to convert **Macromolecule**s in metabolic and machinery costs (Fig.10).

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.

7.8 ConstantProcessing

The **ConstantProcessing** class defines metabolites consumed and byproducts generated by an assembly process (Fig.10). 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**.

7.9 ComponentProcessing

The **ComponentProcessing** class defines metabolites consumed and byproducts generated when assembling a specific **Component** (Fig.10). 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).

8 enzymes.xml

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.

8.1 RBAEnzymes

The outermost portion of the metabolism file is an instance of class **RBAEnzymes**, shown in Figure 11.

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

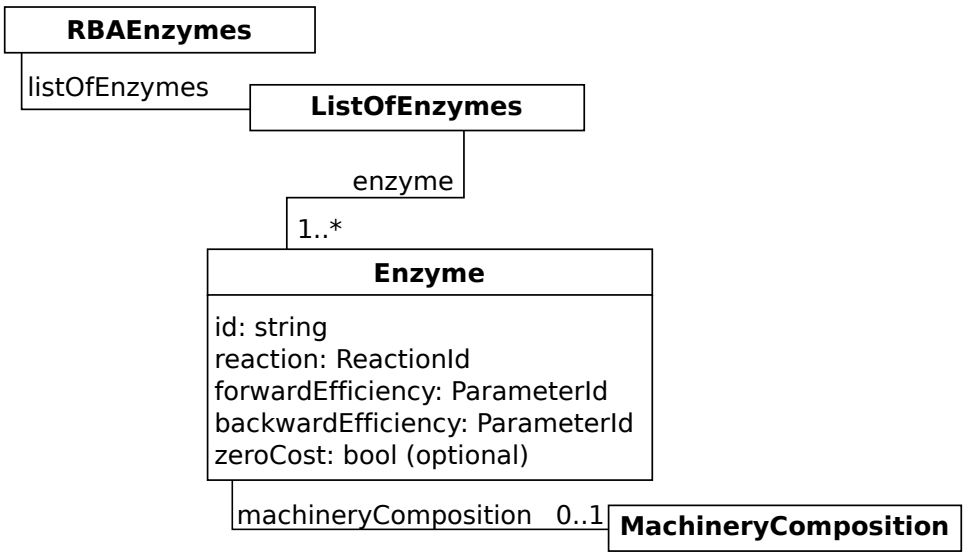

Figure 11: XML structure of enzyme document.

8.2 Enzyme

The **Enzyme** class is used to define enzymes (Fig. 11).

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

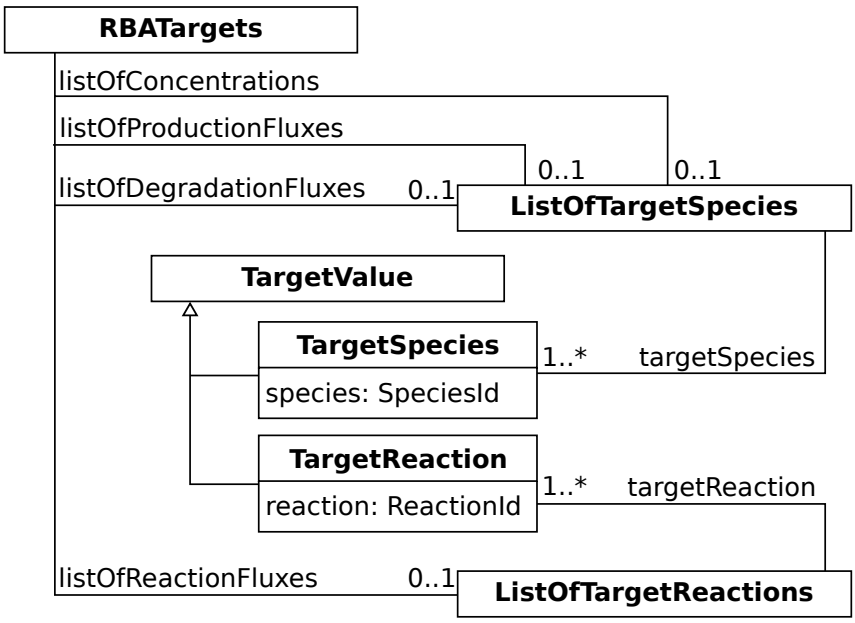

Figure 12: XML structure of target document.

9 targets.xml

The process 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. Constraints can be specified as target fluxes or as target concentrations.

9.1 RBATargets

The outermost portion of the process file is an instance of class **RBATargets**, shown in Figure 12.

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.

9.2 TargetSpecies

The **TargetSpecies** class defines constraints for a species flux (Fig. 12). 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.

9.3 TargetReaction

The **TargetReaction** class defines constraints for a reaction flux (Fig. 12). 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**.