Developing PBK models in SBML

A guide to implement models in Antimony and convert them to SBML

J Minnema A Zwartsen Others

Contents

1. Introduction

‘Since the advent of SBML (the Systems Biology Markup Language) computer models of biological systems have been able to be transferred easily between different labs and different computer programs without loss of specificity. But SBML was not designed to be readable or writable by humans, only by computer programs, so other programs have sprung up to allow users to more easily create the models they need.’[1]

Antimony is such a program. ‘Antimony is designed to interconvert between the SBML format and a shorthand form that allows editing without the structure and overhead of working with SBML directly. In

Antimony, users can easily define substance species, reactions, compartments, events, and other elements of a biological model.' [1] Essentially, Antimony is a relatively easy way of defining models that can be subsequently converted into SBML files using Tellurium. Thus, Antimony provides a way of harmonizing the way in which PBK models are defined within the PARC project.

In this guidance document, we will describe how to use antimony, provide basic coding examples and discuss more complex elements that may be present in your PBK models. The goal of this guidance is not to provide a detailed overview of all functionalities of Antimony, as that can already be found here. Instead, this guidance focus on distilling the aspects of Antimony that may be relevant when building your own PBK models within PARC.

2. Basics of antimony

2.1 Reactions

Antimony uses reactions to define the transfer of a substance to one compartment to the other. Hence, instead of using differential equations to describe the rate of change within a given compartment, modelers should specify the rate with which compounds migrate from one compartment to the other. Consider this two compartment model below (Fig. 1):

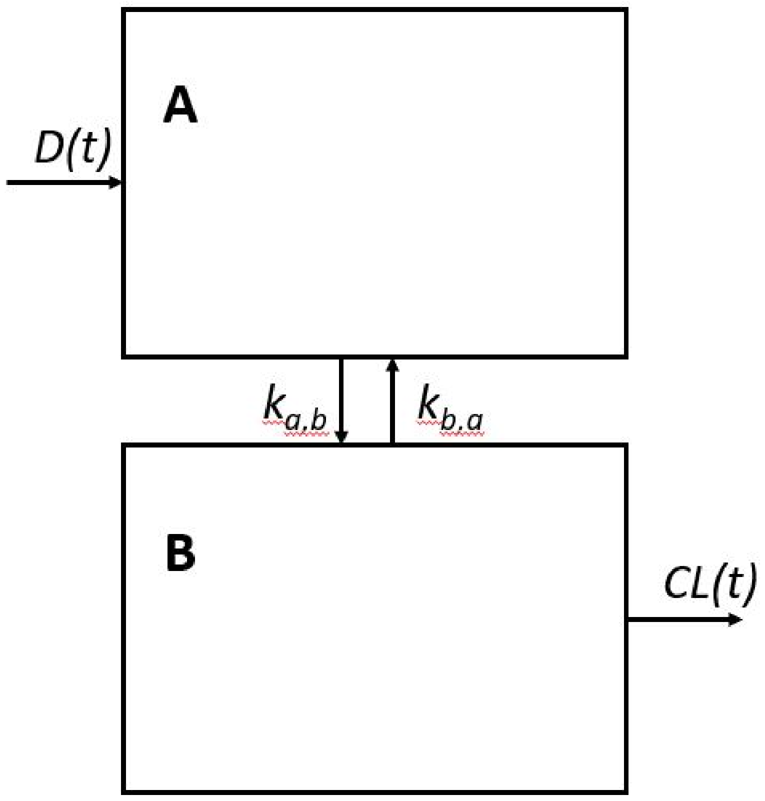

Figure 1: Example of a two-compartment model with metabolism. D(t) represents the dose into compartment A per unit time, k represents the transfer rates from and to compartments A and B. Finally, CL(t) represent the clearance of the compound from compartment B.

In terms of differential equations, this model would be defined as:

$$\frac{dA_A(t)}{dt} = D + k_{B,A} \times B(t) - k_{A,B} \times A(t)$$
$$\frac{dA_B(t)}{dt} = k_{A,B} \times A(t) - k_{B,A} \times B(t) - CL$$

In contrast, in antimony this model is described with the following reactions:

```
-> A; D
A -> B; k_{A,B} * A
B -> A; k_{B,A} * B
B -> ; CL
```

Basically, instead of specifying the rate at which the amount in a compartment changes, modelers should define the ‘arrows’ representing the transfer of a compound.

2.2 Parameters

In the above mentioned model equations, four parameters are used. These parameters need to be defined such that Antimony knows what value to use. Defining parameters can be done by simply setting the parameter equal to a value, e.g.

```
D = 3.4
k_{A,B} = 2.3
k_{B,A} = 0.43
CL = 0.9
```

In many cases, however, the value of a parameter depends on the values of other parameters in the model. For example, the dose intake `D` may depend on the body weight `bw`. In that case, we can instinctively define a body weight and define the dose as a function of `bw`:

```
bw = 70
D = 0.05 * bw
```

2.3 Compartments

In order to define the transfer of a compound to and from a compartment, compartments need to be defined in Antimony. This can be done by using the `compartment` keyword. For example, using the model in Fig. 1, the compartments can be defined as follows

```
compartment A = 2
compartment B = 3
```

Here with defined two compartments, A and B, which have volume of 2 and 3, respectively.

2.4 Species

Typically PBK models are build for a single compound. However, sometimes, compound are metabolized which usually requires building a second PBK model (i.e., a second set of differential equations) that is connected to the first PBK model. Instead of defining a second set of differential equations, one can define multiple species in Antimony. In this context, species refer to a compound that resides within compartment. Hence, in Antimony, for each compound and each compartment, a separate species is defined. To illustrate this, we slightly extend the PBK model shown before (Fig. 2

In this extended PBK model, metabolism within compartment B is included. For this PBK model, four species can be defined that are associated with a certain compartment:

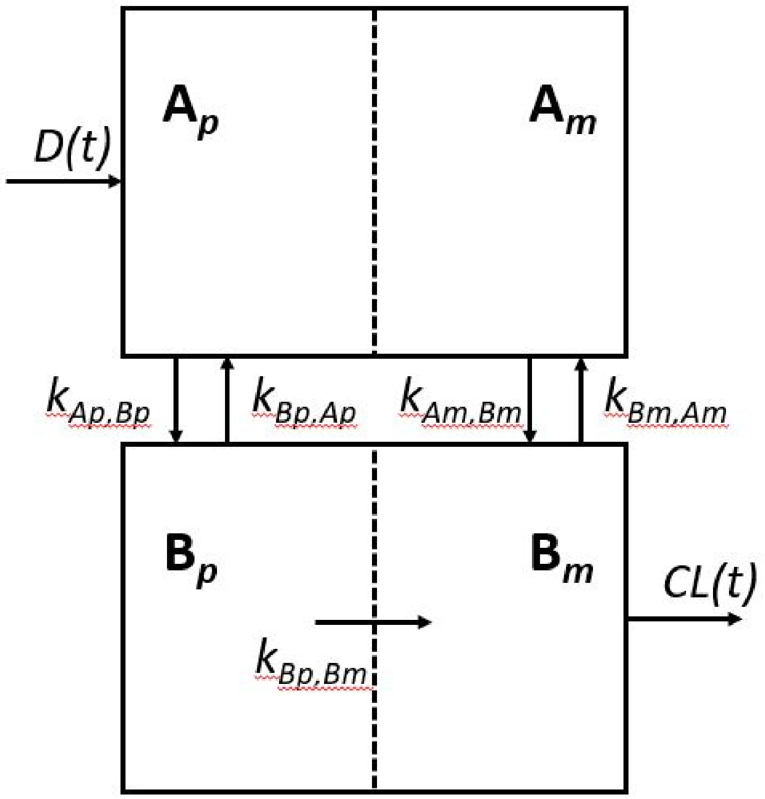

Figure 2: Example of a two-compartment model with metabolism. Ap represents the parent compound in compartment A, while Am represents the metabolite in compartment A. Similarly, Bp represents the parent compound in compartment B, while Bm represents the metabolite in compartment B. D(t) represents the dose (parent compound) into compartment Ap per unit time, k represents the transfer rates from and to compartments A and B, as well as the metabolization rate in compartment B. Finally, CL(t) represent the clearance of the metabolite from compartment B.

```
species Ap in A
species Am in A
species Bp in B
species Bm in B
```

For the sake of completeness, the associated transfer equations would be as follows:

```
-> Ap; D
Ap -> Bp; k_{Ap,Bp} * Ap
Bp -> Ap; k_{Bp,Ap} * Bp
Bp -> Bm; k_{Bp,Bm} * Bp
Am -> Bm; k_{Am,Bm} * Am
Bm -> Am; k_{Bm,Am} * Bm
Bm ->   ; CL
```

By default, species in Antimony are defined as concentration. Since PBK models, commonly use species amounts rather than concentrations, the keyword `SubstanceOnly` can be used. This can be simply done by starting with the keyword when defining a particular species.

```
substanceOnly species Ap in A
substanceOnly species Am in A
substanceOnly species Bp in B
substanceOnly species Bm in B
```

Initial species concentrations (or amounts) can be defined with a simply equal sign:

```
Ap = 0
Am = 0
Bp = 0
Bm = 0
```

2.5 Comments

Single-line comments in Antimony can be created using the `#` or `//` symbols, and multi-line comments can be created by surrounding them with `/* [comments] */`.

```
k1 = 1 # Example comment 1
k2 = 2 // Example comment 2

/* Example
comment 3
*/
k3 = 3
```

3 Advanced antimony

In the previous section you have learned how to create a basic PBK model and its associated transfer rates. Now, you will learn how to use more advanced functionalities of antimony that may be necessary when defining complex PBK models.

3.1 Assignment rules

In the previous sections we have learned that parameters can be defined as functions of other parameters. Hence, the definitions below would lead to a dose `D` of `0.05 * 70 = 3.5`.

```
bw = 70
D = 0.05 * bw
```

After loading an Antimony model using roadrunner, these definitions are evaluated. In addition, users can manually change parameter values with roadrunner. However, when manually changing the definition of a parameter, other depending parameters are not automatically updated.

This means that changing the value of `bw` would not automatically update the value of `D`, which can lead to severe errors in the model. To make sure these values are automatically updated, we need to use so-called assignment rules. These assignment rules essentially make sure that a parameter value is re-evaluated at every point of the model. An assignment rule can be defined by simply adding a colon before the equals sign.

```
bw = 70
D := 0.05 * bw
```

This above definition would make sure that the value of `D` is updated if the value of `bw` is changed.

3.2 Piecewise assignment

A special kind of assignment rule is the piecewise assignment. The piecewise assignment can be use to define rules that cannot expressed by a single function. The piecewise assignment is similar to an if-else statement used in many other programming languages.

The syntax for using a piecewise assignment is the following:

```
bw = 70
D := piecewise(5, bw < 40, 10, bw => 40 & bw < 80, 20)
```

The above piecewise call will return 5 if bw < 40, it will return 10 if bw is between 40 and 80, and it will return 20 otherwise. The piecewise function has this general “do this if this is true, else ... ” pattern, and can be extended to include as may conditions as needed.

3.3 time-dependent parameters Another species kind of assignment rule is the time-dependent assignment. If you want to define a parameters that changes in time, you can use the keyword `time` within the assignment rule.

For example, you may want to define a body weight `bw` depending on time:

```
bw := 3 + 4*time - 0.1*time^2 + 0.001*time^3
```

3.4 Events

Sometimes, when developing a PBK model, one wants to model an event representing a discontinuity in the simulation. For instance, the dosing or intake of a particular substance at a particular time point can be modeled as an event. To model an event, one can simply use the keyword `at` followed by a conditional expression that specifies when the event should take place. For example,

```
at (x > 10): y = 7, x=r+2
```

In the above expression, whenever x transitions from being less than or equal to 10 to being greater to 10, y will be assigned the value of 7, and x will be assigned the value of r+2, using whatever value r has at that moment. For more detailed explanation of events, look here

3.5 Probabilistic modelling and random sampling from distribution

In some cases, one may want to define parameters as a distribution, instead of a point value. In Antimony, this can be done with the following syntax:

```
A.mean = x
A.stdev = x  (or A.standardDeviation = x)
A.coefficientOfVariation = x
A.kurtosis = x
A.median = x
A.mode = x
A.sampleSize = x
A.skewness = x
A.standardError = x
A.variance = x
A.confidenceInterval = {x, y}
A.credibleInterval = {x, y}
A.interquartileRange = {x,y}
A.range = {x,y}
A.distribution = function()
A.externalParameter = x || {x,y} || function()
```

Where `A` may be any symbol in Antimony with mathematical meaning; `x` and `y` may both be either a symbol or a value (i.e. `A.mean=2.4`; `A.confidenceInterval={S1, 8.2}`); `function()` may be any mathematical formula.

—- Write something about how this information can be used in roadrunner/python. I'm not sure yet about this. —-

3.6 Custom function definitions

In some cases, it may be useful to define a custom function, for example when there is an expression that is repeated often in your model. You can do this, to some extent, by using the `function` keyword. These function must be basic single equations, e.g.:

```
function quadratic(x, a, b, c)
  a*x^2 + b*x + c
end
```

4. Model annotation

Before converting the Antimony file to SBML, the model should be annotated. This annotation is essential in order to harmonize parameters, compartments and units in Antimony models developed by a variety of researchers. To this end, we created a template XXX.xlsx. Further documentation on how to fill in this Annotation file can be found here (to be developed!).

5. Conversion to SBML using automated Github workflow

An automated workflow has been developed to convert the developed Antimony model, including annotations, to SBML. Specifically, a Github workflow was developed, in which the Antimony model is converted to SBML upon pushing the model to the repository. Some understanding of Github is assumed here. If you are not yet familiar with using Github, please take a look at the documentation, or contact us for specific help.

To use the automated workflow, you simply save the following code chunk as a file (build.yml) in a directory *.github/workflows/* in your Github repository.

```
name: Create and annotate SBML
on: [push, workflow_dispatch]

jobs:
  create-and-annotate-sbml:
    uses: jwkruisselbrink/sbml-pbk-workflow/.github/workflows/build.yml@main
    with:
      model-name: your_model_name
    permissions:
      contents: write
    secrets: inherit
```

Note that you should replace *your_model_name* with the name of the model you want to convert.

The automated workflow will check the validity of the model and its annotations. If all checks are passed, the Antimony model will be automatically converted into SBML.

6. Running simulations and testing your model using tellurium

Both Antimony and SBML models can be loaded and used to perform simulations using Tellurium. Tellurium is a Python package that can deal with these data types and run model defined in either Antimony or SBML. In the following subsections, a few Tellurium functionalities will be described that may help you to check whether your model is correctly implemented.

6.1 Installation In order to use tellurium, you need Python (version 3.7, 3.8 or 3.9) and you need to install Tellurium:

```
$ pip install tellurium
```

6.2 Loading model

Loading a model with tellurium is fairly simple and can be done as follows:

```
import tellurium as te
my_model = te.loada(path_to_my_model.ant)           # For Antimony models
my_model = te.loadSBMLModel(path_to_my_model.sbml)  # For SBML models
```

6.2 Run simulations

Once a model (either Antimony or SBML) has been loaded using tellurium, one can perform a model simulation. For example, a simple simulation can be performed using the `simulate` function.

```
simulation = my_model.simulate(0,24,48)
```

With the above statement, we perform a model simulation from t=0 to t=24 with 48 steps, with the model as was loaded from Antimony or SBML

6.3 Change parameter values

Often times, one needs to perform model simulations for various sets of parameters. For example, one may be interested in performing a PBK model simulation of individuals with varying physiological characteristics, such as a higher body weight. Varying these parameter values does not require constantly changing the Antimony and/or SBML files. Instead, Tellurium offers the functionality of changing parameter values after loading the model. Specifically, after loading the initial model, one can access and change parameter values as follows:

```
# Accessing body weight parameter (bw)
my_model.bw

# Changing parameter value
my_model.bw = new_value
```

7. Tips and tricks

7.1 Common errors

7.2 Separation of model and its dosing

8 References

[1] https://tellurium.readthedocs.io/en/latest/antimony.html