OneModel

Please, go to:

`bit.ly/iwbda-onemodel`

and open the Google Colab notebook

ONEMODEL: AN OPEN-SOURCE SBML MODELING TOOL

IWBDA @ IGEM 2022 — WORKSHOP

Fernando N. Santos-Navarro

Wednesday, 26th October 2022

Synthetic Biology and BioSystems Control Lab (SB2CL)
Universitat Politècnica de València (UPV)

1. Introduction
2. PART 1: How to model biological processes using OneModel
3. PART 2: How to reuse and extend previously-defined models
4. Conclusions

INTRODUCTION

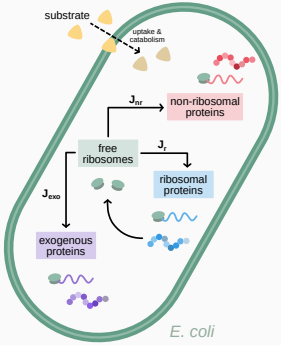
Fernando N. Santos-Navarro — fersann1@upv.es

I have (recently) finished my PhD

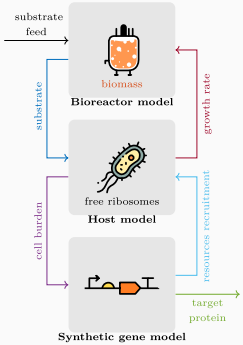
The main topics of my Thesis were:

- How to model competition for free resources in cells — Host-aware model
- The development of tools to facilitate modeling — OneModel

Host-aware model*



Multi-scale model**



*Santos-Navarro, F. N., Vignoni, A., Boada, Y., & Picó, J. (2021). “RBS and Promoter Strengths Determine the Cell-Growth-Dependent Protein Mass Fractions and Their Optimal Synthesis Rates”. ACS Synthetic Biology.

**Santos-Navarro, F. N., Boada, Y., Vignoni, A., & Picó, J. (2021). “Gene Expression Space Shapes the Bioprocess Trade-Offs among Titer, Yield and Productivity”. Applied Sciences.

OneModel

Open-source text-based tool for defining SBML models

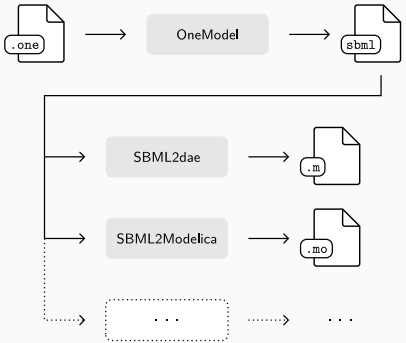
Minimizes the user's programming knowledge requirements

Modularity, accessibility and simplicity

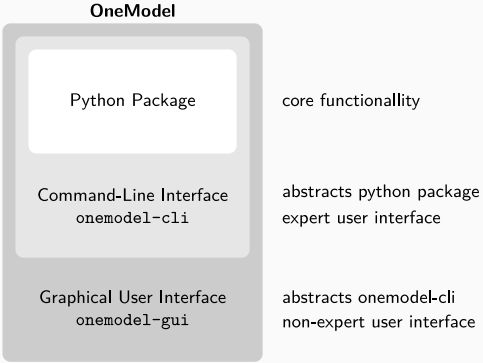
Requirements	Antimony	Little b	BioCRNpyler	OneModel
Reactions	✓	✓	✓	✓
ODE	✓	✓	✓	✓
DAE	–	–	–	✓
Modularity	✓	✓	✓	✓
Accesibility	✓	–	✓	✓
Simplicity	✓	✓	✓	✓
Open source	✓	✓	✓	✓

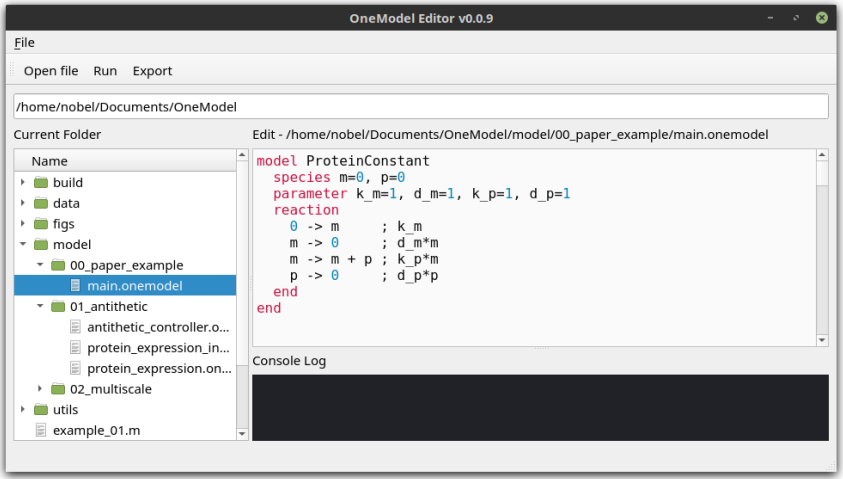
- **Reactions:** models based on reactions
- **ODE:** models based on ordinary differential equations
- **DAE:** models based on differential-algebraic equations
- **Modularity:** to define models incrementally and reuse model parts
- **Accessibility:** low entry barriers for non-expert users, and ease of integration
- **Simplicity:** limited to definition SBML models and the simplicity of tool's implementation
- **Open source:** the source code is freely available

1. Write the model using the OneModel syntax
2. Export the model to SBML
3. Feed the model into SBML-compatible software (simulations, analysis, figures, etc)
4. Validate the model
5. Repeat the loop, but using this model as a base for a new one



- **Python** — open-source, easy-to-learn, compatibility with other programs
- **TatSu** — to build the OneModel syntax parser
- **libSBML** — to export models as SBML files
- **Click** — to build the command-line interface
- **PyQt5** — to build the graphical user interface




OneModel Editor v0.0.9
File
Open file
Run
Export
/home/nobel/Documents/OneModel
Current Folder
Edit - /home/nobel/Documents/OneModel/model/00_paper_example/main.onemodel
Name
build
data
figs
model
00_paper_example
main.onemodel
01_antithetic
antithetic_controller.o...
protein_expression_in...
protein_expression.on...
02_multiscale
utils
example_01.m
model ProteinConstant
species m=0, p=0
parameter k_m=1, d_m=1, k_p=1, d_p=1
reaction
0 -> m ; k_m
m -> 0 ; d_m*m
m -> m + p ; k_p*m
p -> 0 ; d_p*p
end
end
Console Log

PART 1: How to model biological processes using OneModel

PART 2: How to reuse and extend previously-defined models

Please, go to:

`bit.ly/iwbda-onemodel`

and open the Google Colab notebook.

Exercise 00	**2 minutes**

Go to WORKPLACE 00

1. Run the cell to setup OneModel in the Notebook

Note: to run/execute a cell you can press Ctrl+Intro.

PART 1: HOW TO MODEL BIOLOGICAL PROCESSES USING ONEMODEL

OneModel Python Core

```
m = OneModel()

m['mRNA'] = Species(initialConcentration=0)
m['protein'] = Species(initialConcentration=0)

m['k_m'] = Parameter(value=1)
m['d_m'] = Parameter(value=1)
m['k_p'] = Parameter(value=1)
m['d_p'] = Parameter(value=1)

m['J1'] = Reaction()
m['J1']["reactants"] = []
m['J1']["products"] = ['mRNA']
m['J1']["kinetic_law"] = 'k_m'

m['J2'] = Reaction()
m['J2']["reactants"] = ['mRNA']
m['J2']["products"] = []
m['J2']["kinetic_law"] = 'd_m*mRNA'

m['J3'] = Reaction()
m['J3']["reactants"] = ['mRNA']
m['J3']["products"] = ['mRNA', 'protein']
m['J3']["kinetic_law"] = 'k_p*mRNA'

m['J4'] = Reaction()
m['J4']["reactants"] = ['protein']
m['J4']["products"] = []
m['J4']["kinetic_law"] = 'd_p*protein'
```

OneModel Syntax

```
species
  mRNA = 0
  protein = 0
end

parameter
  k_m = 1
  d_m = 1
  k_p = 1
  d_p = 1
end

reaction
  0 -> mRNA ; k_m
  mRNA -> 0 ; d_m*mRNA
  mRNA -> mRNA + protein; d_p*mRNA
  protein -> 0 ; d_p*protein
end
```

The OneModel syntax allows the definition of DAE models:

- Parameters
- Species
- Reactions
- Assigment rules
- Rate rules
- Algebraic rules

```
# This is a comment

parameter k = 10 # This is a parameter

species A = 0 # This is a species

reaction 0 -> A ; k # This is a reaction
```

```
# Single-line parameter definition
parameter k1 = 1

# Multiple parameter definition separated by commas
parameter k2 = -1, k3 = 3.14, k4 = 1e+5, k5

# Parameter definition block
parameter
  k = 1
  d = 0.12
end
```

Parameters are values that do not change during simulation time

```
# Single-line species definition
species S1 = 1 # "1" is the initial condition

# Multiple species definition separated by commas
species S2 = -1, S3 = 1e-1, S4

# Species definition block
species
  mRNA = 0
  protein = 0
end
```

Species are values that change during simulation time:

- reactions
- rules (assigment, rate or algebraic)

Understand them as state variables (can take negative values)

```
# A is produced at rate "k"
reaction 0 -> A ; k

reaction
  # Note we have to explicity write reaction rates
  A -> B + C ; k_A*A

  # We can give a name to a reaction
  my_reaction: A -> 0 ; d_A*A
end
```

```
rule
  # Assigment rule
  S := 10*s

  # Rate rule
  der(x) := S - x

  # Algebraic rule
  y == 10 - x

  # As reactions, we can give them names
  R1: z := x + y
end
```

Exercise 02 **3 minutes**

Go to WORKPLACE 03

1. Execute the first cell to write into `text_model.one`
2. Download `text_model.one` locally
   (it saved the inside `./src/` folder)
3. Open it in your computer using a text editor
4. Run the “cat” cell to preview the file in the Colab Notebook

Note: if the file is not shown,
try to update the directory tree in Colab

Exercise 03 **3 minutes**

Go to WORKPLACE 04

1. Run the first cell to create `src/my_model.one`
2. Run the second cell to load `my_model.one` using OneModel
3. Check the model and the simulation result
4. Make a change in `my_model.one` (parameter values or species initial conditions)
5. Run again the simulation and check the differences

Exercise 04 **8 minutes**

Go to WORKPLACE 05

1. Implement the set of reactions, species and parameter using OneModel syntax in `src/protein_constitutive.one`
2. Load the model (without errors)
3. Run the simulation

PART 2: HOW TO REUSE AND EXTEND YOUR MODELS USING ONEMODEL

```
one> parameter k = 1 "This is my parameter"

one> k
<parameter value=1>

one> show(k)
Name         Value                    Documentation
----------   ----------------------   ---------------
units        'per_second'
value        1
isConstant   True
__doc__      'This is my parameter'

one> k.value = 100

one> k
<parameter value=100>
```

OneModel allows object-oriented programming

```
model ProteinExpression
  species protein = 0

  parameter k = 1, d = 0.1

  reaction
      0 -> protein ; k
      protein -> 0 ; d*protein
  end
end

A = ProteinExpression()
```

```
one> locals()
Name                Value
-----------------   ------------------------------
A                   <object>
ProteinExpression   <model>
...                 ...

one> show(A)
Name      Value
-------   ----------------------------------------
_R1       <reaction eq='protein -> 0 ; d*protein'>
_R2       <reaction eq='0 -> protein ; k'>
d         <parameter value=0.1>
k         <parameter value=1>
protein   <species initialConcentration=0>

one> A.k
<parameter value=1>
```

Exercise 05 **3 minutes**

Go to WORKPLACE 05

1. Use the keyword `model` to define `ProteinConstitutive` model
2. Use `ProteinConstitutive()` to define two proteins `A` and `B`
3. Change the value of `d_p` of protein `A`
4. Check the change in the simulation

protein_expression.one

```
model ProteinExpression
  species protein = 0
  parameter k = 1, d = 0.1
  reaction
      0 -> protein ; k
      protein -> 0 ; d*protein
  end
end
```

other_model.one

```
# Imports works like in Python
import protein_expression

A = protein_expression.ProteinExpression()

# The most useful one is from ... import ...
from protein_expression import ProteinExpression

B = ProteinExpression()
```

```
model ProteinExpression
  species protein = 0
  parameter k = 1, d = 0.1
  reaction
      0 -> protein ; k
      protein -> 0 ; d*protein
  end
end

model ProteinInduced
  extends ProteinExpression

  species TF = 0, k
  rule k := TF/(TF + 1)
end

B = ProteinInduced()
```

```
one> locals()
Name               Value
-----------------  ------------------------------
A                  <object>
ProteinInduced     <model>
ProteinExpression  <model>
...                ...

one> show(A)
Name     Value
-------  ----------------------------------------
_R2      <assignment-rule eq='k := TF/(TF + 1)'>
TF       <species initialConcentration=0>
_J1      <reaction eq='protein -> 0 ; d*protein'>
_J0      <reaction eq='0 -> protein ; k'>
d        <parameter value=0.1>
k        <species initialConcentration=0>
protein  <species initialConcentration=0>
__doc__  ''
```

Exercise 06 **3 minutes**

Go to WORKPLACE 06

1. Import `ProteinConstitutive` using:

   ```
   from protein_constitutive import ProteinConstitutive
   ```

2. Define a new model `ProteinInduced`
3. Extend `ProteinConstitutive` in `ProteinInduced` using:

   ```
   extends ProteinConstitutive
   ```

4. Code the new parts for inducible expression (check the information in the Notebook)

CONCLUSIONS

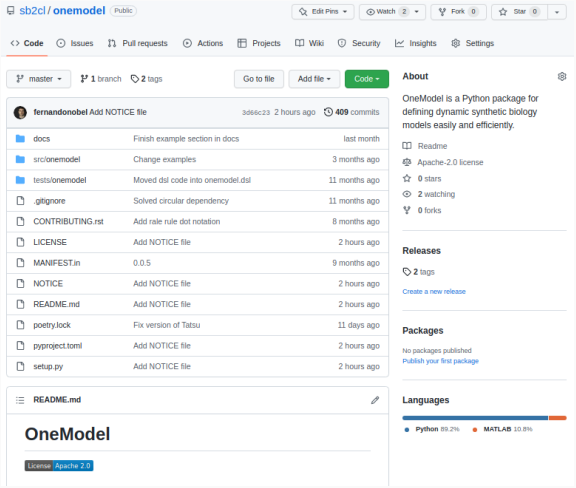
Now, you know...

- How to model biological processes using OneModel
- How to reuse and extend previously-defined models

Accesibility, simplicity, and modularity

Availiable at:

- GitHub (Source Code)
- PyPI (Python Package Index)
- Read the Docs (Documentation)



- Full-SBML compatibility
- Class-based vs. prototype-based
- A standard built-in library for OneModel

If you find OneModel useful, please consider giving a star in the GitHub repository: `github.com/sb2cl/onemodel`

Please do not hesitate to contact me

And, most importantly, all feedback is welcome! :-)

Questions?

fersann1@upv.es

ONEMODEL: AN OPEN-SOURCE SBML MODELING TOOL

IWBDA @ IGEM 2022 — WORKSHOP

Fernando N. Santos-Navarro

Wednesday, 26th October 2022

Synthetic Biology and BioSystems Control Lab (SB2CL)
Universitat Politècnica de València (UPV)