Design Document
Biomolecular Subsystems Interaction Modeling Instrument (BioSIMI) Python toolbox

Key -
TextInBlueUnderline - libSBML object/class/member functions
(Python API - http://sbml.org/Software/libSBML/5.13.0/docs/python-api/index.html)
TextInRed - New Class names (not in libSBML)
TextInGreen - Member functions of a class (not in libSBML)

Class Name: CreateSubsystem

The main subsystem class. Creates an SBML model with the use of different member functions. The argument is a SBMLDocument object. This SBMLDocument object is used to create a model (inside the document) as desired, and the compartments, species, reactions can be added to the model using the various member functions. Using this class, new SBML models can be created and written to XML files. The class facilitates simple creation of models without having to write long libSBML code always. The system interconnection member function is also a part of this class. It also has the simulate and plot function to plot species amounts using bioscrape. Uses the check function to validate all libSBML function calls. Uses the check2 function to validate all other function calls.

Attributes:

1. NewDocument(SBMLDocument object)

Functions:

1. __init__(self, NewDocument)
2. getNewDocument(self)
3. setNewDocument(self, SBMLDocument object)
4. getFromXML(self, filename)
5. setDocToXML(self, filename)
6. createNewModel(self, timeUnits, extentUnits, substanceUnits):
7. createNewUnit(self, unitID, unitKind, unitExponent, unitScale, unitMultiplier)
8. createNewCompartment(self, compartmentId, compartmentName, compartmentSize, compartmentUnits, compartmentConstant)
9. createNewSpecies(self, speciesId, speciesName, speciesCompartment, speciesInitialAmount, speciesConstant, speciesBoundaryCondition, speciesSubstanceUnits, speciesHasOnlySubstanceUnits)
10. createNewParameter(self, parameterId, parameterName, parameterValue, parameterConstant, parameterUnit)
11. createNewReaction(self, reactionId, reactionReversible, reactionFast)
12. connectInteraction(self, [ListOfInputSubsystems], [ListOfOutputSubsystems], connectionLogicMap)

13. createNewDocument(newLevel, newVersion)
14. plotSbmlWithBioscrape(filename, initialTime, timepoints, ListOfSpeciesToPlot, xlabel, ylabel, sizeOfXLabels, sizeOfYLabels)
15. check(value, message)
16. check2(value, message)

Status - Code written and documented with comments. Available at - Github

Class Name: NewSubsystem

To create a new copy of a given system model in SBML format. Creates a new subsystem model in SBML format with the new given name in the arguments. The new name given is used to name all the species/reactions/parameters etc. to give a unique identity. Use the same name in the NewName string to make copies of SBML models when needed. Uses the check function to validate all libSBML function calls. Uses the check2 function to validate all other function calls.

Attributes:

1. Subsystem(SBMLDocument object)
2. NewName(String)

Functions:

1. __init__(self, Subsystem)
2. getSubsystem()
3. setSubsystem(self, SBMLDocument object)
4. getNewName()
5. setNewName(self, String)
6. createNewSubsystem(self, NewName)

Status - Code written. Documentation and bug fixes pending. Available at - Github

Class Name: NewReaction

The NewReaction class is a level below the CreateSubsystem class. The NewReaction class takes libSBML Reaction object as its argument. The member functions can be used to create a new reaction or modify existing ones. Reaction rates and parameters can also be created. Uses the check function to validate all libSBML function calls. Uses the check2 function to validate all other function calls.

Attributes:

1. Reaction (Reaction object)

Functions:

1. __init__(self, Reaction)
2. getReaction(self)
3. setReaction(Reaction)
4. createNewReactant(self, reactantSpeciesId, reactantConstant, reactantStoichiometry)

5. createNewProduct(self, productSpeciesId, productConstant, productStoichiometry)
6. createRate(self, math_ast)
7. createMath(self, formulaString)

Status - Code written and documented with comments. Available at - Github

Class Name: ModelReduce

From the libSBML model object and a list of reactions object, the ModelReduce class can find the reactions from the given list which are set as "fast" in the corresponding attribute of reaction class in the model. The other member functions in this class can be used to return a reduced order model where the fast reactions have been replaced by their equilibrium amounts. Uses the check function to validate all libSBML function calls. Uses the check2 function to validate all other function calls.

Attributes:

1. Model(Model object)
2. ListFastReactions(ListOfReactions object)

Functions:

1. __init__(self, Model)
2. getListFastReaction(self)
3. setListFastReaction(self, ListOfReactions)
4. reduceModel(self, reduceMap)

Status - Code pending.

Class Name: Compartmentalize

From the libSBML model object and a list of compartments, the Compartmentalize class can check the list of species in the model for their compartments. The member functions can be used to return a compartmentalized model with list of species in different compartments. The compartmentalized model ensures common species not in same compartments to be different and vice versa. Uses the check function to validate all libSBML function calls. Uses the check2 function to validate all other function calls.

Attribute:

1. Model(Model object)
2. ListCompartments(ListOfCompartments object)

Functions:

1. __init__(self, Model)
2. checkCompartment(self, ListCompartments, ListOfSpecies)
3. getCompartmentalizedModel(self)
4. setCompartmentalizedModel(self, ListCompartments)

Status - Code pending.

Class Name : Model
libSBML class - http://sbml.org/Software/libSBML/5.13.0/docs/python-api/classlibsbml_1_1_model.html

Function:

- setModel(self, Model)
- getModel(self)
- addCompartment(self, c)
- addSpecies(self, s)
- addReaction(self, r)
- createSpecies(self, s)

And many more...

Class Name: Species
libSBML class - http://sbml.org/Software/libSBML/5.13.0/docs/python-api/classlibsbml_1_1_species.html

Functions:

- __init__(name, initialAmount = 0)
- getName()
- setName(name)
- getId()
- setId(id)
- getInitialAmount()
- setInitialAmount(initialAmount)

And many more...

Class Name: Parameter
libSBML class - http://sbml.org/Software/libSBML/5.13.0/docs/python-api/classlibsbml_1_1_parameter.html

Functions:

- __init__(name, value = 0)
- getName()
- setName(name)
- getValue()
- setValue(value)
- getConstant()
- setConstant(isConstant)

And many more...

Other important libSBML classes -
http://sbml.org/Software/libSBML/5.13.0/docs/python-api/classlibsbml_1_1_list_of.html

Class Name: Delay

Warning - Currently, bioscrape does not support SBML with delay. The Delay class can be used to set delays between different reactions.

Attributes:

- type (string)
- value (parameter)
- parameterInvolved(parameter)

Functions:

- __init__(type, value, [parameters])
- getType (self)
- setType(self, type)
- getMath(self)
- setMath(self, parameter)
- getParameters(self)
- setParameters(self, [ListOfParameters])

Example demonstrations -

1. Double Phosphorylated system modeling and simulation

Create a new SBML model - Double Phosphorylation subsystem and simulate it using Bioscrape. The following code describes the method to create the SBML model using libsbml but using the classes and member functions defined above, which simplify the process of creating a model to a great extent. Using the object structure defined, the SBML model is simply created by using one line commands to create species, reactions, reactants, reaction rates etc. Finally, the model is simulated using bioscrape.

```
from modules.CreateSubsystem import *
from modules.NewReaction import *

# Create a new SBML Document to hold the subsystem model
sbmlDoc1 = createNewDocument(3,1)
sbmlDoc = CreateSubsystem(sbmlDoc1)

# Create a new model inside the document, arguments - units
model = sbmlDoc.createNewModel("seconds","mole","count")

# Create a unit arguments - id, unitKind, exponent, scale, multiplier
```

```
per_second = sbmlDoc.createNewUnit('per_second',UNIT_KIND_SECOND,-1,0,1)

# Create compartment
# createNewcompartment arguments - compartment ID, Name, Size, Units, isConstant
comp = sbmlDoc.createNewCompartment('cell','cell',1,'litre',True)

# Create all species
# createNewSpecies arguments - id, name, compartment, initial amount, isConstant,
BoundaryCondition, Substance units, HasOnlySubstance
inp_DP1 = sbmlDoc.createNewSpecies( 'inp_DP1','inp_DP1','cell',50,False,False,'count',False)
X_DP1 = sbmlDoc.createNewSpecies( 'X_DP1','X_DP1','cell',50,False,False,'count',False)
C1_DP1 = sbmlDoc.createNewSpecies( 'C1_DP1','C1_DP1','cell',0,False,False,'count',False)
Xp_DP1 = sbmlDoc.createNewSpecies( 'Xp_DP1','Xp_DP1','cell',0,False,False,'count',False)
E_DP1 = sbmlDoc.createNewSpecies( 'E_DP1','E_DP1','cell',50,False,False,'count',False)
C2_DP1 = sbmlDoc.createNewSpecies( 'C2_DP1','C2_DP1','cell',0,False,False,'count',False)
C3_DP1 = sbmlDoc.createNewSpecies( 'C3_DP1','C3_DP1','cell',0,False,False,'count',False)
out_DP1 = sbmlDoc.createNewSpecies( 'out_DP1','out_DP1','cell',0,False,False,'count',False)
C4_DP1 = sbmlDoc.createNewSpecies( 'C4_DP1','C4_DP1','cell',0,False,False,'count',False)

# Create all parameters, arguments - id, name, value, isConstant, unit
k1f = sbmlDoc.createNewParameter( 'k1f','k1f',1,False,'per_second')
k1r = sbmlDoc.createNewParameter( 'k1r','k1r',1,False,'per_second')

k2f = sbmlDoc.createNewParameter( 'k2f','k2f',1,False,'per_second')

k3f = sbmlDoc.createNewParameter( 'k3f','k3f',1,False,'per_second')
k3r = sbmlDoc.createNewParameter( 'k3r','k3r',1,False,'per_second')

k4f = sbmlDoc.createNewParameter( 'k4f','k4f',1,False,'per_second')

k5f = sbmlDoc.createNewParameter( 'k5f','k5f',1,False,'per_second')
k5r = sbmlDoc.createNewParameter( 'k5r','k5r',1,False,'per_second')

k6f = sbmlDoc.createNewParameter( 'k6f','k6f',1,False,'per_second')

k7f = sbmlDoc.createNewParameter( 'k7f','k7f',1,False,'per_second')
k7r = sbmlDoc.createNewParameter( 'k7r','k7r',1,False,'per_second')

k8f = sbmlDoc.createNewParameter( 'k8f','k8f',1,False,'per_second')

# Create all reactions
# Arguments - id, isReversible, isFast
```

```
# Arguments - species id, isConstant, Stoichiometry

r1 = NewReaction(sbmlDoc.createNewReaction('r1',True,False))
sref1_inp_DP1 = r1.createNewReactant('inp_DP1',False,1)
sref1_X_DP1 = r1.createNewReactant('X_DP1',False,1)
sref1_C1_DP1 = r1.createNewProduct('C1_DP1',False,1)
math_r1 = r1.createMath('k1f * inp_DP1 * X_DP1 - k1r * C1_DP1')
r1_rate = r1.createRate(math_r1)

r2 = NewReaction(sbmlDoc.createNewReaction('r2',False,False))
sref2_C1_DP1 = r2.createNewReactant('C1_DP1',False,1)
sref2_inp_DP1 = r2.createNewProduct('inp_DP1',False,1)
sref2_Xp_DP1 = r2.createNewProduct('Xp_DP1',False,1)
math_r2 = r2.createMath('k2f * C1_DP1')
r2_rate = r2.createRate(math_r2)

r3 = NewReaction(sbmlDoc.createNewReaction('r3',True,False))
sref3_E_DP1 = r3.createNewReactant('E_DP1',False,1)
sref3_Xp_DP1 = r3.createNewReactant('Xp_DP1',False,1)
sref3_C2_DP1 = r3.createNewProduct('C2_DP1',False,1)
math_r3 = r3.createMath('k3f * E_DP1 * Xp_DP1 - k3r * C2_DP1')
r3_rate = r3.createRate(math_r3)

r4 = NewReaction(sbmlDoc.createNewReaction('r4',False,False))
sref4_C2_DP1 = r4.createNewReactant('C2_DP1',False,1)
sref4_E_DP1 = r4.createNewProduct('E_DP1',False,1)
sref4_X_DP1 = r4.createNewProduct('X_DP1',False,1)
math_r4 = r4.createMath('k4f * C2_DP1')
r4_rate = r4.createRate(math_r4)

r5 = NewReaction(sbmlDoc.createNewReaction('r5',True,False))
sref5_inp_DP1 = r5.createNewReactant('inp_DP1',False,1)
sref5_Xp_DP1 = r5.createNewReactant('Xp_DP1',False,1)
sref5_C3_DP1 = r5.createNewProduct('C3_DP1',False,1)
math_r5 = r5.createMath('k5f * inp_DP1 * Xp_DP1 - k5r * C3_DP1')
r5_rate = r5.createRate(math_r5)

r6 = NewReaction(sbmlDoc.createNewReaction('r6',False,False))
sref6_C3_DP1 = r6.createNewReactant('C3_DP1',False,1)
sref6_out_DP1 = r6.createNewProduct('out_DP1',False,1)
sref6_inp_DP1 = r6.createNewProduct('inp_DP1',False,1)
math_r6 = r6.createMath('k6f * C3_DP1')
```

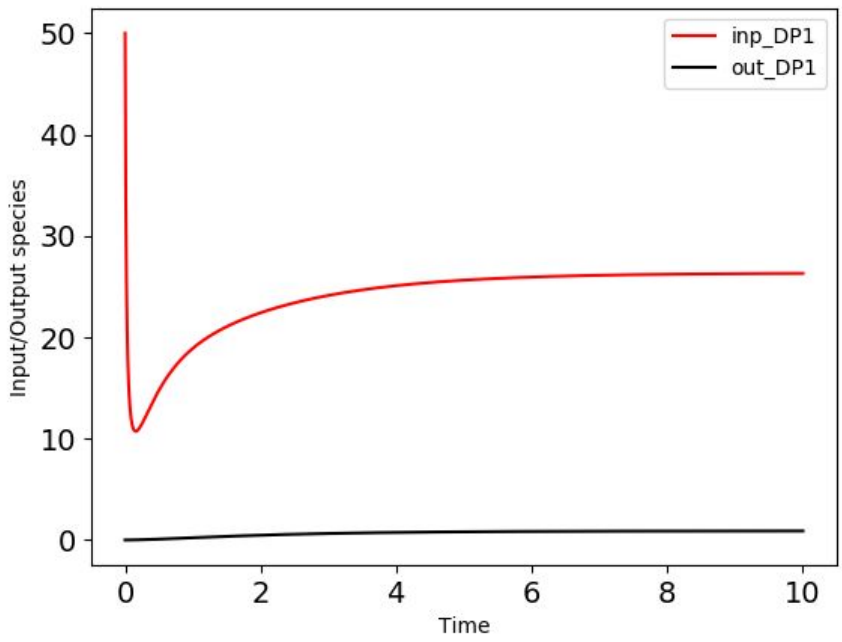
```
r6_rate = r6.createRate(math_r6)

r7 = NewReaction(sbmlDoc.createNewReaction('r7',True,False))
sref7_E_DP1 = r7.createNewReactant('E_DP1',False,1)
sref7_out_DP1 = r7.createNewReactant('out_DP1',False,1)
sref7_C4_DP1 = r7.createNewProduct('C4_DP1',False,1)
math_r7 = r7.createMath('k7f * E_DP1 * out_DP1 - k7r * C4_DP1')
r7_rate = r7.createRate(math_r7)

r8 = NewReaction(sbmlDoc.createNewReaction('r8',False,False))
sref8_C4_DP1 = r8.createNewReactant('C4_DP1',False,1)
sref8_Xp_DP1 = r8.createNewProduct('Xp_DP1',False,1)
sref8_E_DP1 = r8.createNewProduct('E_DP1',False,1)
math_r8 = r8.createMath('k8f * C4_DP1')
r8_rate = r8.createRate(math_r8)
# Write to XML file
writeSBML(sbmlDoc.getNewDocument(),'models/DP1_sbml.xml')

# Simulate and plot using bioscrape
timepoints = np.linspace(0, 10, 1000)
plotSbmlWithBioscrape('models/DP1_sbml.xml',0,timepoints,['inp_DP1','out_DP1'],'Time','Input
/Output species',14,14)
```

Result -



Similarly, Incoherent Feedforward Loop (IFFL) or any other model can be simulated.

2. Interaction between two DP and IFFL subsystems

Using the above method, two DP and one IFFL subsystem models were created. The interaction between the double phosphorylated species is modeled in this example. The double phosphorylated system output activates the expression of proteins in the incoherent feedforward loop system.

```
import numpy as np
from libsbml import *
from modules.CreateSubsystem import *
from modules.NewReaction import *

# read all SBML models (for this example - Two DP systems and one IFFL system)
reader = SBMLReader()
doc_DP1 = reader.readSBML('models/DP1_sbml.xml')
DP1_subsystem = CreateSubsystem(doc_DP1)

doc_DP2 = reader.readSBML('models/DP2_sbml.xml')
DP2_subsystem = CreateSubsystem(doc_DP2)

doc_IFFL = reader.readSBML('models/IFFL_sbmlNew.xml')
IFFL_Subsystem = CreateSubsystem(doc_IFFL)

# create a blank document for the final connected system
final_sbml_doc = createNewDocument(doc_IFFL.getLevel(),doc_IFFL.getVersion())
check(final_sbml_doc.createModel(),'creating model of final doc')
Final_subsystem = CreateSubsystem(final_sbml_doc)

# user specifies how the systems interact by defining the following map
connection_logic = {}
connection_logic["out_DP1"] = "pA_IFFL"
connection_logic["out_DP2"] = "pB_IFFL"

# Call the connect function by specifying the input and output subsystems and the logic map
Final_subsystem.connectInteraction([DP1_subsystem, DP2_subsystem],[IFFL_Subsystem],
connection_logic)

# Write the connected document to SBML file

writeSBML(Final_subsystem.getNewDocument(),'models/DP_IFFL_connected.xml')
```

```
# Simulate
timepoints = np.linspace(0,100,1000)
plotSbmlWithBioscrape('models/DP_IFFL_connected.xml',0,
timepoints,['inp_DP1','inp_DP2','out_IFFL'],'Time',
'Input and Output Species',14,14)
```

Result -