from \_\_future\_\_ import print\_function, division, absolute\_import

from six import iteritems, string\_types

from cobra import Metabolite as Component

from cobrame.util import dogma

class MEComponent(Component):

"""

COBRAme component representation. Inherits from

:class:`cobra.core.metabolite.Metabolite`

Parameters

----------

id : str

Identifier of the component. Should follow best practices of child

classes

"""

def \_\_init\_\_(self, id):

Component.\_\_init\_\_(self, id)

def remove\_from\_me\_model(self, method='subtractive'):

"""

Remove metabolite from me model along with any relevant

:class:`cobrame.core.processdata.ProcessData`

Parameters

----------

method : str

- destructive: remove metabolite from model and remove reactions

it is involved in

- subtractive: remove only metabolite from model

"""

if self.id in self.\_model.process\_data:

self.\_model.process\_data.remove(self.id)

# If cannot import SymbolicParameter, assume using cobrapy

# versions <= 0.5.11

try:

from optlang.interface import SymbolicParameter

except ImportError:

self.remove\_from\_model(method=method)

else:

if method.lower() == 'subtractive':

self.remove\_from\_model(destructive=False)

elif method.lower() == 'destructive':

self.remove\_from\_model(destructive=True)

else:

raise AttributeError("method must be subtractive or "

"destructive")

class Metabolite(MEComponent):

from \_\_future\_\_ import print\_function, division, absolute\_import

import re

from six import iteritems

from warnings import warn

from cobra import Model, DictList

import numpy as np

from scipy.sparse import dok\_matrix

from cobrame.core.reaction import (SummaryVariable, MetabolicReaction,

TranscriptionReaction, TranslationReaction)

from cobrame.core.component import (Constraint, ProcessedProtein, Complex,

TranslatedGene, TranscribedGene)

from cobrame.core import processdata

from cobrame.util import mu

class MEModel(Model):

def \_\_init\_\_(self, \*args):

Model.\_\_init\_\_(self, \*args)

self.global\_info = {}

self.process\_data = DictList()

# create the biomass/dilution constraint

self.\_biomass = Constraint("biomass")

self.\_biomass\_dilution = SummaryVariable("biomass\_dilution")

self.\_biomass\_dilution.add\_metabolites({self.\_biomass: -1})

self.add\_reactions([self.\_biomass\_dilution])

self.\_biomass\_dilution.upper\_bound = mu

self.\_biomass\_dilution.lower\_bound = mu

# maintenance energy

self.\_gam = 0.

self.\_ngam = 0.

# Unmodeled protein is handled by converting protein\_biomass to

# biomass, and requiring production of the appropriate amount of dummy

# protein

self.\_unmodeled\_protein\_fraction = None

def add\_biomass\_constraints\_to\_model(self, biomass\_types):

for biomass\_type in biomass\_types:

if '\_biomass' not in biomass\_type:

raise ValueError('Biomass types should be suffixed with '

'"\_biomass"')

constraint\_obj = Constraint(biomass\_type)

summary\_variable\_obj = SummaryVariable("%s\_to\_biomass" %

biomass\_type)

summary\_variable\_obj.add\_metabolites({constraint\_obj: -1,

self.\_biomass: 1})

self.add\_reactions([summary\_variable\_obj])

@property

def unmodeled\_protein(self):

return self.metabolites.get\_by\_id("protein\_dummy")

@property

from \_\_future\_\_ import absolute\_import, print\_function, division

from collections import defaultdict

from six import iteritems, string\_types

from warnings import warn

import cobra

from cobrame.core.reaction import GenericFormationReaction, ComplexFormation

from cobrame.core.component import GenerictRNA, GenericComponent

from cobrame.util.massbalance import elements\_to\_formula

from cobrame.util import dogma

class ProcessData(object):

"""Generic class for storing information about a process

This class essentially acts as a database that contains all of the

relevant information needed to construct a particular reaction. For

example, to construct a transcription reaction, following information must

be accessed in some way:

- nucleotide sequence of the transcription unit

- RNA\_polymerase (w/ sigma factor)

- RNAs transcribed from transcription unit

- other processes involved in transcription of RNAs (splicing, etc.)

ME-model reactions are built from information in these objects.

Parameters

----------

id : str

Identifier of the ProcessData instance.

model : :class:`cobrame.core.model.MEModel`

ME-model that the ProcessData is associated with

"""

def \_\_init\_\_(self, id, model):

self.id = id

self.\_model = model

# parents need to be updated every time the process is updated

# a parent must have an update method

self.\_parent\_reactions = set()

model.process\_data.append(self)

@property

def model(self):

"""

Get the ME-model the process data is associated with

Returns

-------

:class:`cobrame.core.model.MEModel