# Documentation on PyNSource models.

version 1.7

Jan 2019

Andy Bulka

abulka@gmail.com

# There are two ‘models’ in use.

1. **ParseModel** – this is the parse model, after the raw parsing.
2. DisplayModel for displaying the parse model on the canvas, a mapping of abstract nodes to wxpython OGL shapes done via a **UmlGraph** – a graph of nodes and edges for layout purposes, self.graph

The parse model "classes" are represented by Graph "nodes" where the node.id is the name of the class, and the node x,y,width,height is abstract positioning for layout purposes. Subclasses of GraphNode have attributes and methods for properly representing the parse model "classes" which have been detected from parsing the source code.

## ParseModel

In class CmdFileImportBase inside filemgmt.py the parsing process begins with a call to new\_parser(f) which returns the parsemodel. The parsemodel structure is documented in dump\_pmodel.py thus:

class AbstractParseModel(object):

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

self.**classlist** = {}

self.**modulemethods** = []

self.**errors** = "" # new as of 2019

where

.classlist {classname:classentry, ...} where classname is a string, classentry is a class containing

.ismodulenotrealclass T/F

.classdependencytuples [(fromclass, toclass), ...]

.classesinheritsfrom [class, ...] # todo should be renamed classinheritsfrom (singular)

.attrs [attrobj, ...]

.attrname

.attrtype [] # todo should be renamed attrtypes plural

.compositedependencies # todo (calculated in real time, should precalc)

.defs [method, ...]

.modulemethods = [method, ...]]

There is a useful

dump\_old\_structure(pmodel)

function in there that you can use to dump the parse model. There is nothing in the gui that can dump the parse model out, but that is easily addresses. For example you could modify filemgmt.py to dump the parse model after a File/Import is done e.g.

class CmdFileImportBase(CmdBase): # BASE

def execute(self):

assert self.files

# these are tuples between class names.

self.context.model.ClearAssociations() # WHY DO WE WANT TO DESTROY THIS VALUABLE INFO?

if self.files:

for f in self.files:

#pmodel, debuginfo = old\_parser(f)

pmodel, debuginfo = new\_parser(f)

from parsing.dump\_pmodel import dump\_old\_structure # ADD THIS

print dump\_old\_structure(pmodel) # ADD THIS

self.context.model.ConvertParseModelToUmlModel(pmodel)

**Tip:** You could thus potentially use **dump\_old\_structure()** as a basis for a different loop that generates e.g. UMI.

## DisplayModel which is just the UmlGraph Model

The DisplayModel class reads the parse model and creates a different model of Graph **nodes/edges** mapped to wxpython OGL **shapes** on the umlcanvas. It stores these mappings in self.graph, which contains all the Graph **nodes/edges** which are used for layout purposes.

Each graph node corresponds to a shape in terms of x, y, width, height – but nothing else, because that's all you need for working out layout and overlap.

More specific nodes are subclassed from GraphNode e.g. UmlNode and CommentNode. All GraphNodes live inside a Graph. Our specialised UmlGraph inherites from Graph and has extra methods for persistence related stuff – see the diagram comment nodes - UML class-sequence-combo diagram 2019.pdf

* self.graph = UmlGraph() # graph is full of GraphNode objects, and associations, which are ultimately what we layout.

containing

self.graph.nodes

self.graph.edges

nodes are a list of GraphNode

edges are a list of dictionaries with the keys

edge["source"] which is the from\_node

edge["target"] which is the to\_node

edge["uml\_edge\_type"] one of "generalisation", "composition", "association"

once displaymodel.build\_view() is called, shapes are attached to the graph model thus:

node.shape

edge["shape"]

# Dumping out the Models

## Dumping out the ParseModel

pmodel, debuginfo = parse\_source(source\_code, options={})

print(pmodel.classlist)

print((dump\_old\_structure(pmodel)))

**Fred (is module=0) inherits from ['Mary', 'Sam']**

**class dependencies []**

**modulemethods []**

## Dumping out the DisplayModel

displaymode.Dump()

**vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv**

**DisplayModel (Graph)**

**---------------------------------------------------------------------- self.graph.nodes**

**Node ShapeMenuMgr: x/left,y/top ( 10, 10) w,h ( 202, 354) layoutPosX,layoutPosY (-0.74, -1.10) shape=<gui.uml\_shapes.DividedShape>**

**Node wx.Menu: x/left,y/top ( 522, 374) w,h ( 69, 24) layoutPosX,layoutPosY ( 0.74, 1.10) shape=<gui.uml\_shapes.DividedShape>**

**---------------------------------------------------------------------- self.graph.edges**

**from ShapeMenuMgr ---> wx.Menu (composition) shape=None**

**MISC Scaling etc info**

**scale and radius 2 10**

**world\_size (1024, 728)**

**layout (MinX/MinY)(MaxX/Maxy) (-0.74,-1.10) (0.74,1.10)**

**layout width height 1.49 2.19**

**factorX factorY 343 166**

**line-line intersections 0**

**node-node overlaps 0**

**line-node crossings 0.0**

**bounds (522, 374)**

This can be dumped at any time in the gui. Just run rungui.bat to ensure you get the console window and hit ‘d’ in the GUI when there is something in the workspace.

If you look at class CmdDumpUmlWorkspace(CmdBase): in diagnostics.py you will see this is called by pressing 'd' whilst the gui is running. It dumps the model using self.context.model.Dump()

Sometimes I see nothing dumped out in generalisation relationship diagnostic dump – not sure why that is – needs to be investigated.

**Tip:** You could potentially change the this dump routine to generate UMI or other formats, as a prototyping technique. Just print to console and hit ‘d’ to see the result – easy development! Later the algorithm can be properly integrated into PyNSource.

# Persistence format

.pyns files

All formats are ‘custom’ and based on arrays, dictionaries and tuples. The ocassional class/object is also used.

{'type':'node', 'id':'CmdBase', 'x':557, 'y':270, 'width':87, 'height':31, 'attrs':'', 'meths':''}

{'type':'node', 'id':'CmdDeselectAllShapes', 'x':650, 'y':14, 'width':230, 'height':83, 'attrs':'', 'meths':'execute|undo'}

{'type':'edge', 'id':'CmdFileSaveWorkspace\_to\_CmdBase', 'source':'CmdFileSaveWorkspace', 'target':'CmdBase', 'uml\_edge\_type':'generalisation'}

{'type':'edge', 'id':'CmdFileLoadWorkspaceFromQuickPrompt\_to\_CmdFileLoadWorkspaceBase', 'source':'CmdFileLoadWorkspaceFromQuickPrompt', 'target':'CmdFileLoadWorkspaceBase', 'uml\_edge\_type':'generalisation'}

## 2019 update, the 1.2 format

Added the comment type e.g.

# PynSource Version 1.2

{'type':'meta', 'info1':'Lorem ipsum dolor sit amet, consectetur adipiscing elit is latin. Comments are saved.'}

{'type':'umlshape', 'id':'MyPrintout', 'x':21, 'y':164, 'width':136, 'height':174, 'attrs': 'canvas|log', 'meths': '\_\_init\_\_|OnBeginDocument|OnPreparePrinting|HasPage|GetPageInfo|OnPrintPage|IncreasePrintAreaSize'}

{'type':'umlshape', 'id':'wx', 'x':74, 'y':30, 'width':27, 'height':24, 'attrs': '', 'meths': ''}

{**'type':'comment'**, 'id':'C8004', 'x':203, 'y':42, 'width':200, 'height':100, 'comment': 'aW5pdGlhbCBjb21tZW50CnRoaXMgaXMgYSBjb21tZW50'}

{'type':'edge', 'id':'MyPrintout\_to\_wx', 'source':'MyPrintout', 'target':'wx', 'uml\_edge\_type': 'generalisation'}

# Appendix

## Differences between old parser model used in pynsource and GitUML alsm

OLD PARSE MODEL GITUML

USED BY PYNSOURCE ALSM TYPE

----------------- ----------- -----------------------

pmodel.classlist alsm.classes : Dict[str, ClassEntry]

ClassEntry

.name .name

.defs .methods : List[?]

.attrs .attributes : List[?]

.classesinheritsfrom .classes\_inherits\_from : List[str]

.classdependencytuples .class\_dependency\_tuples : List[str]

.ismodulenotrealclass no equivalent cos an alsm

represents a module

+ other differences not documented yet.