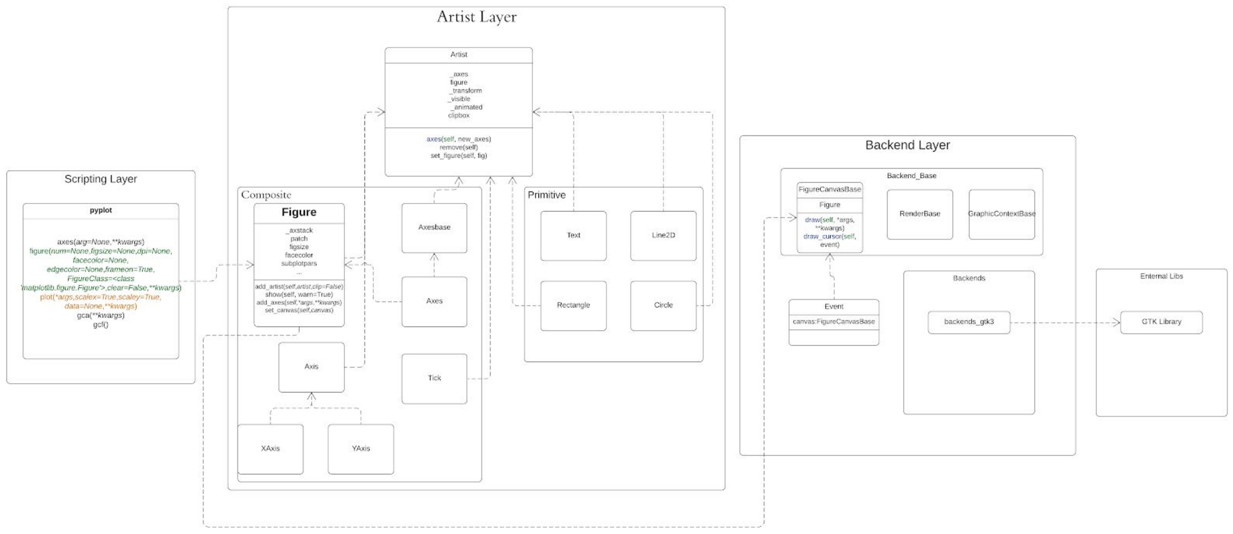
CSCD01 - Team I/O

Deliverable 1: Software Architecture and Development Process

Robert Augustynowicz
Omar Chehab
Jinyang Hu
Dennis Tismenko
Jinming Zhang

Software Architecture

Overall Higher level UML diagram



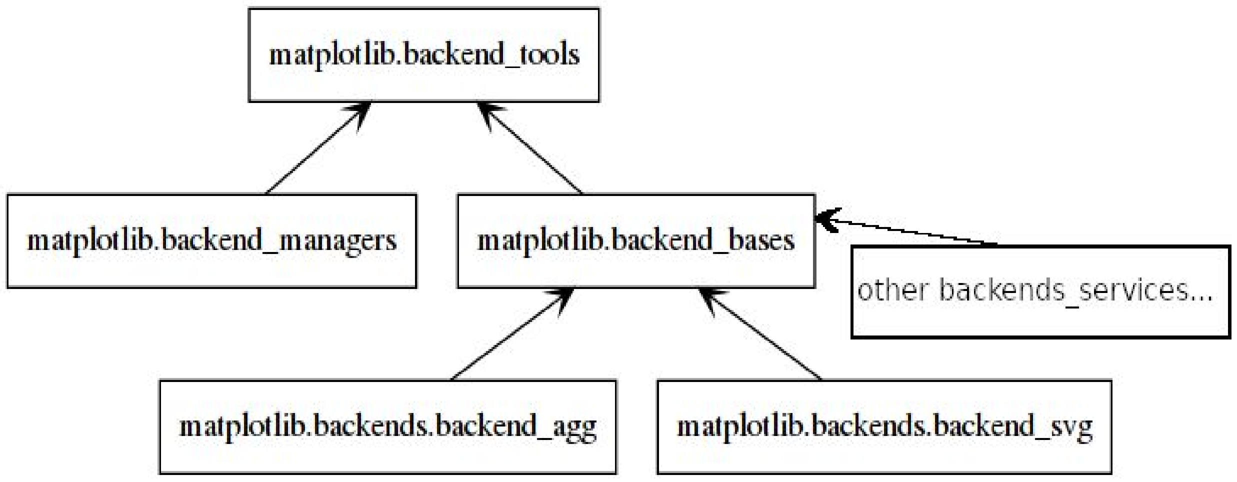
(Original Size)

3 Layers Architecture of matplotlib:

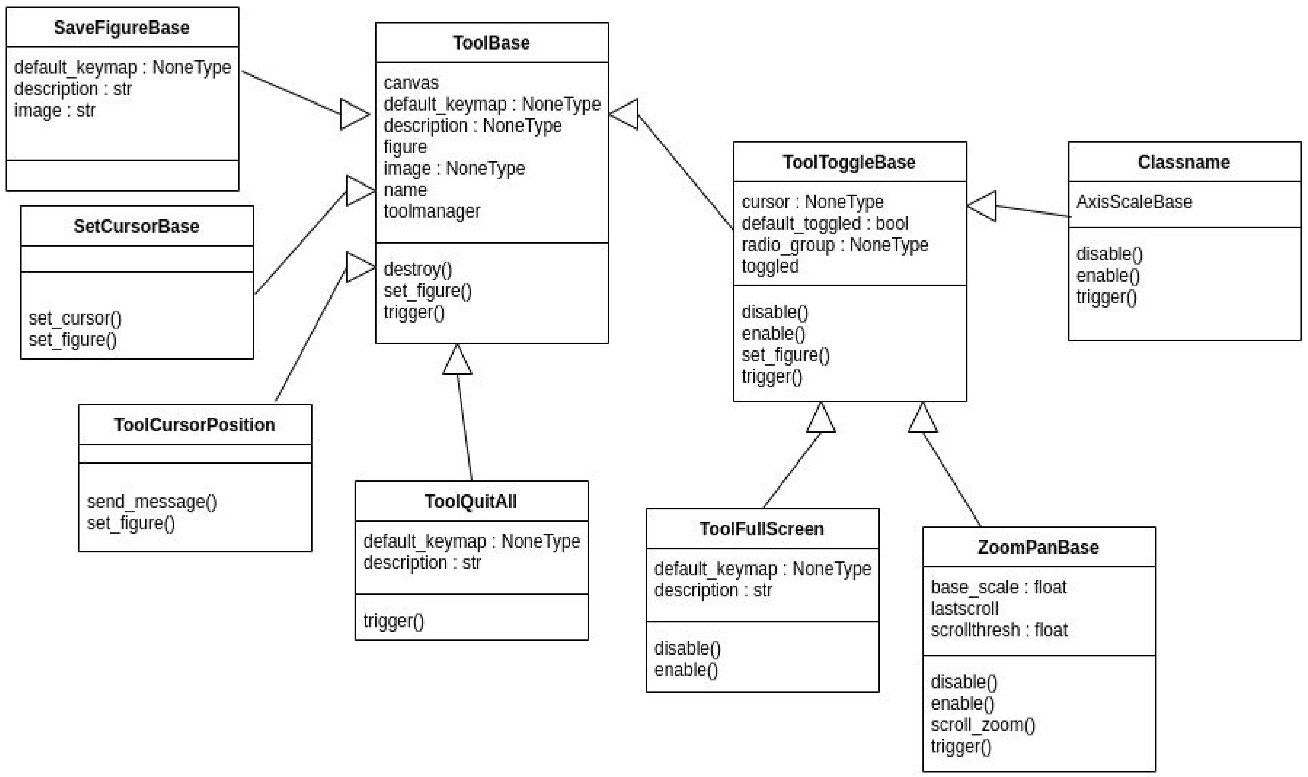
- Backend Layer
- Artist Layer
- Script Layer (pyplot)

Backend layer:

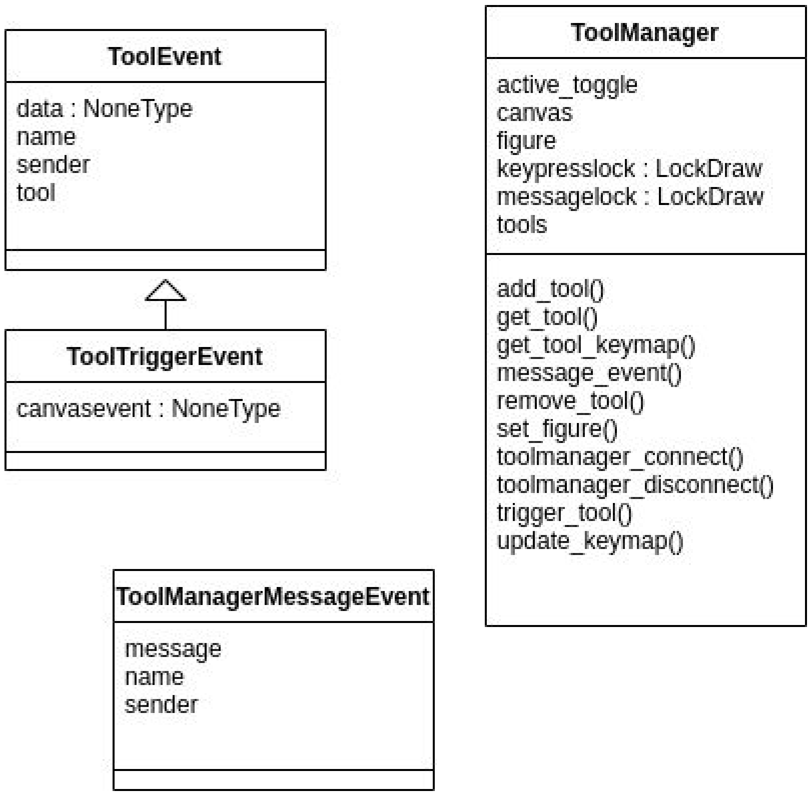
- Provides concrete implementations of the abstract interface classes.
- Responsible for rendering graphics and event handling.
- Can be extended for different purposes, ie, backend for saving drawings in different file formats
- Has three main classes:
    - FigureCanvas: Logical canvas responsible for storing graphical states, cooperate with Rendering operations, listen to GUI events and transfer to Event class
    - Renderer: Responsible for 'drawing' operations(draw_path, draw_image, draw_text…). For example, the backend_agg.py implementation supports Anti-Grain Geometry(agg), a C++ template library for efficient 2D graphic rendering, as well as pixel-mapping for agg generated images and python GUI.
    - Event: Responsible for connecting GUI events to callback event handling functions, embedded in FigureCanvas class
- Generated and Simplified Backend Layer component UML:


matplotlib.backend_tools
matplotlib.backend_managers
matplotlib.backend_bases
other backends_services...
matplotlib.backends.backend_agg
matplotlib.backends.backend_svg

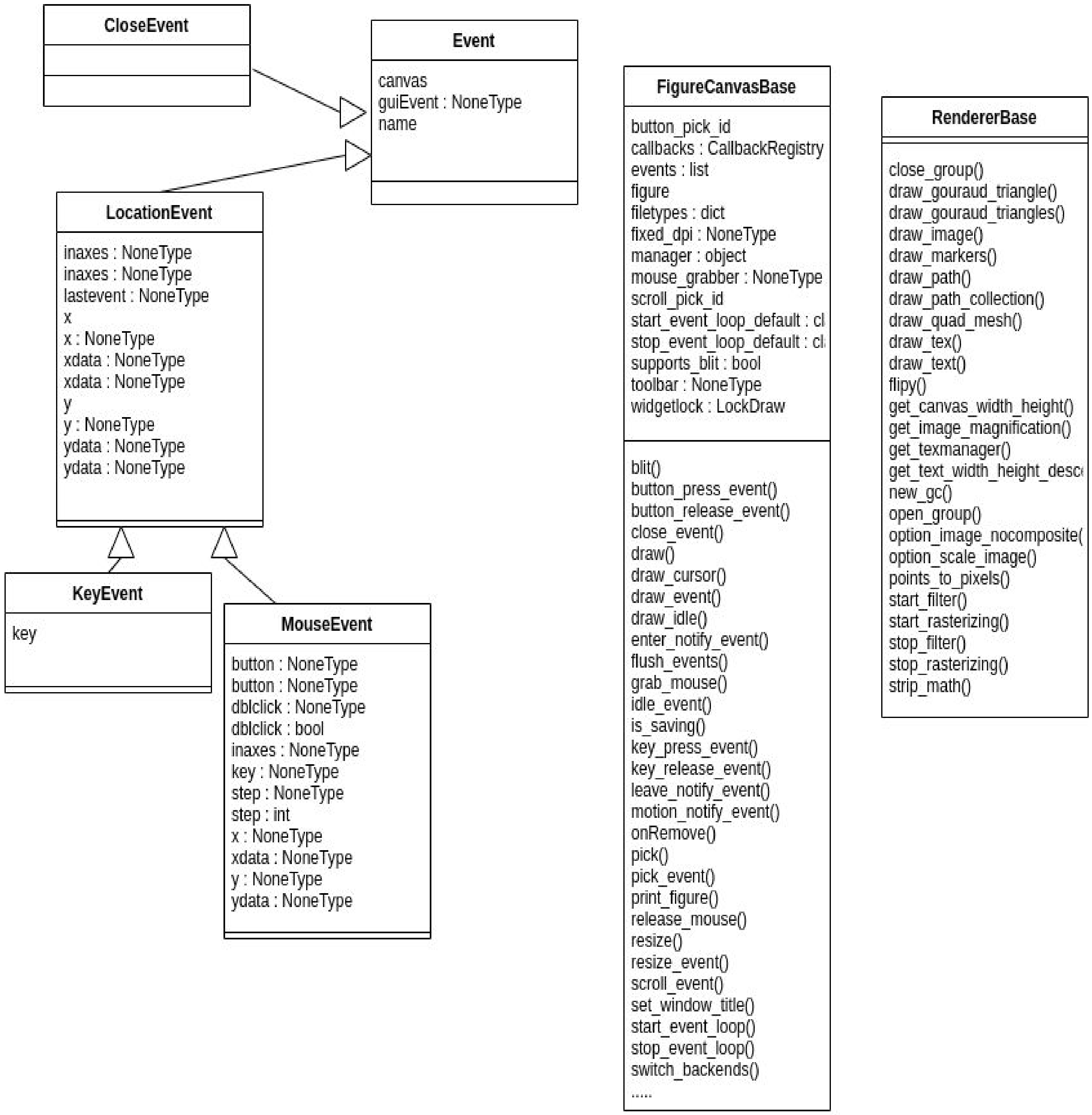
- Generated and Simplified UML for backend_tools:



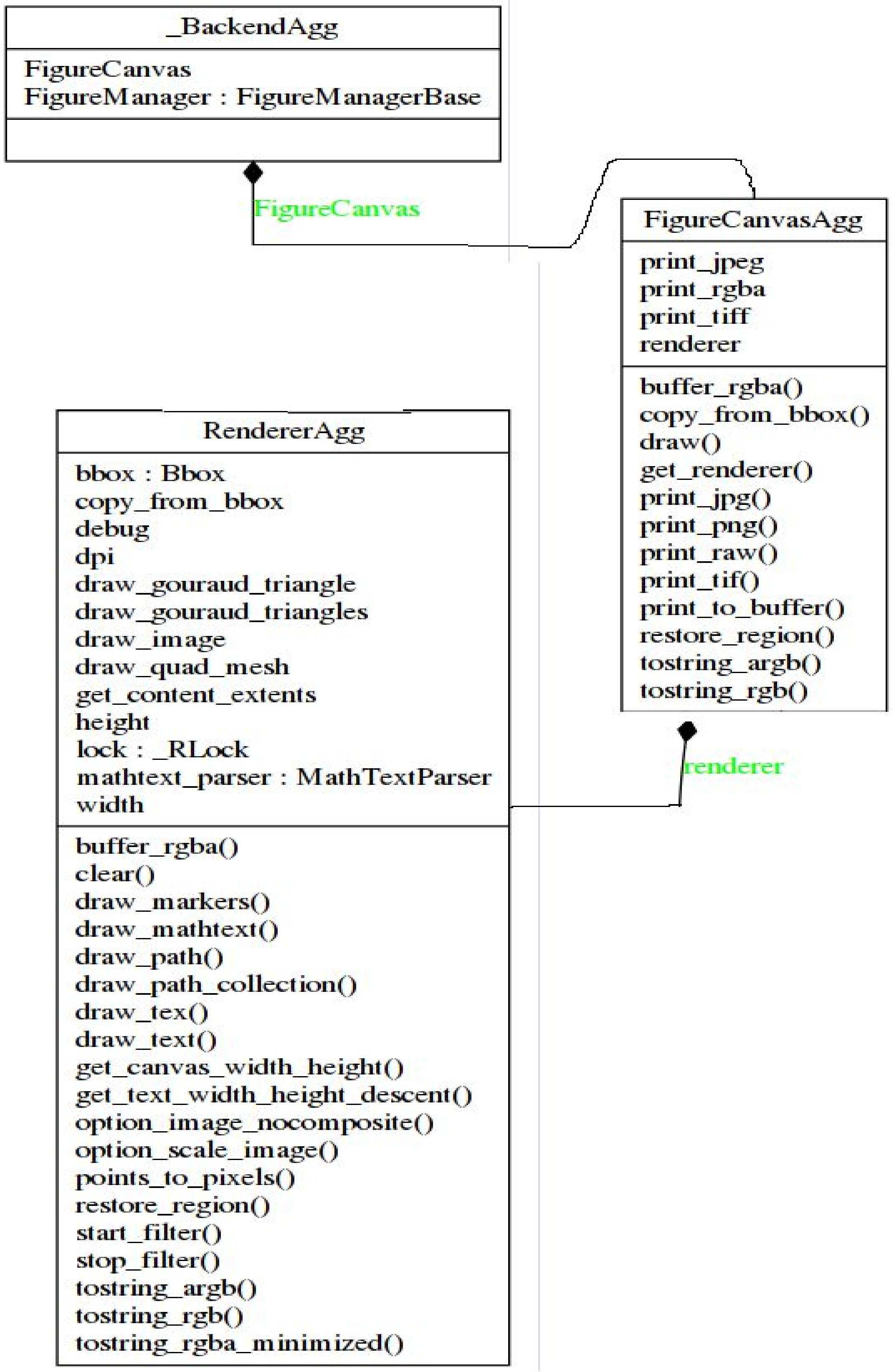
- Generated and Simplified UML for backend_managers:



- Generated and Simplified UML for backend_bases:

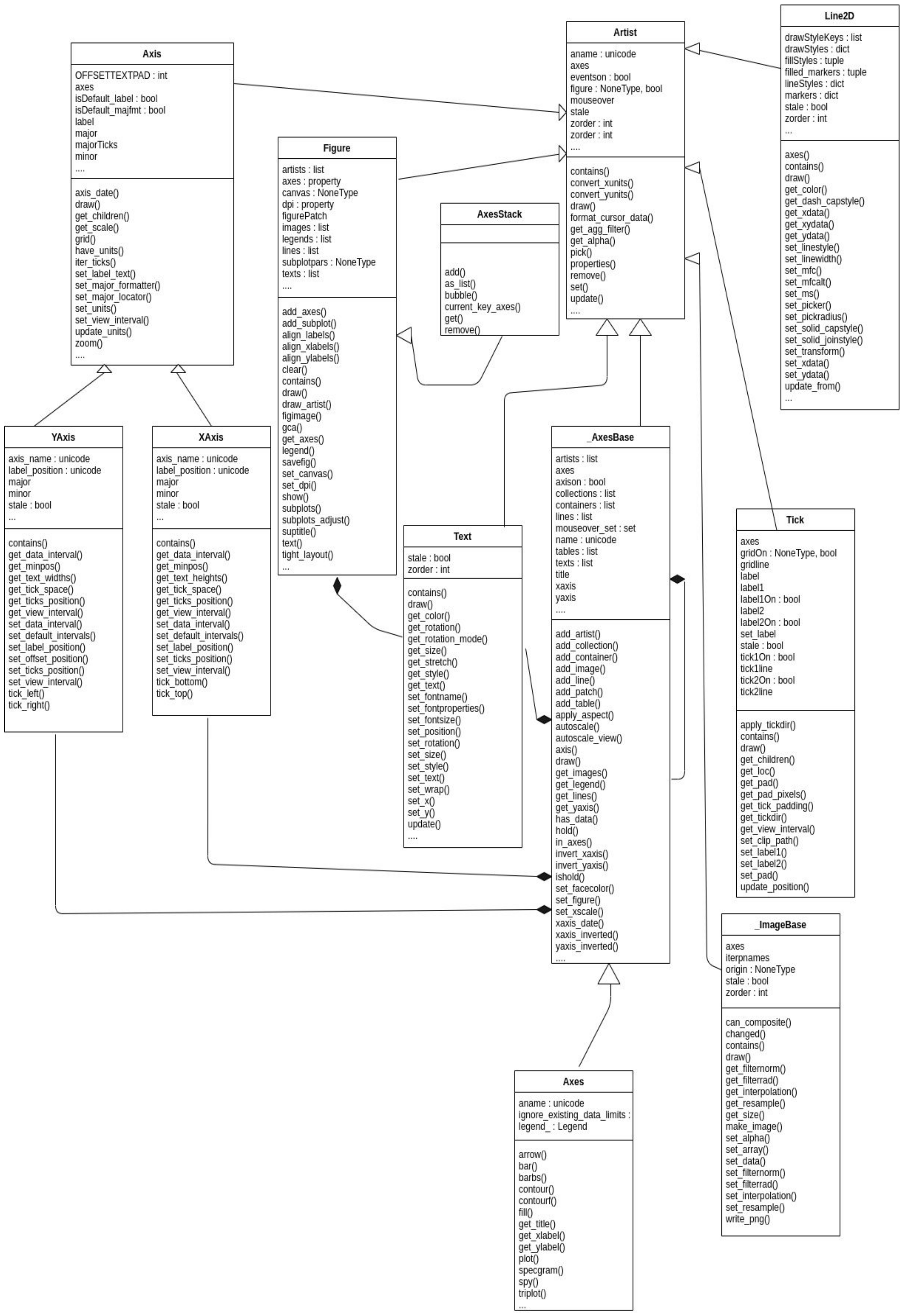


- Generated and Simplified UML for Backend Layer AGG implementation:



Artist layer:

- Responsible for passing drawing commands to backend layer (backend interprets the drawing command and apply it regarding the service needed. Ie,pdf, png, agg…)
- Everything in a matplotlib Figure is an Artist instance: title, lines, tick labels, images
- Interact with Backend layer through ‘draw’ method
- Has two types of Artist:
  - Primitive Artists: Basic shapes comprises a graph, ie, line2D, texts...
  - Composite Artists: A collection of Artist, ie, Axis, Tick, Axes, and Figure. Each composite artist can also contain other composite artists and primitive artists.
- Simplified UML for Artist layer:


Axis
OFFSETTEXTPAD : int
axes
isDefault_label : bool
isDefault_majfmt : bool
label
major
majorTicks
minor
....
axis_date()
draw()
get_children()
get_scale()
grid()
have_units()
iter_ticks()
set_label_text()
set_major_formatter()
set_major_locator()
set_units()
set_view_interval()
update_units()
zoom()
....
Figure
artists : list
axes : property
canvas : NoneType
dpi : property
figurePatch
images : list
legends : list
lines : list
subplotpars : NoneType
texts : list
....
add_axes()
add_subplot()
align_labels()
align_xlabels()
align_ylabels()
clear()
contains()
draw()
draw_artist()
figimage()
gca()
get_axes()
legend()
savefig()
set_canvas()
set_dpi()
show()
subplots()
subplots_adjust()
suptitle()
text()
tight_layout()
...
AxesStack
add()
as_list()
bubble()
current_key_axes()
get()
remove()
Artist
aname : unicode
axes
eventson : bool
figure : NoneType, bool
mouseover
stale
zorder : int
zorder : int
....
contains()
convert_xunits()
convert_yunits()
draw()
format_cursor_data()
get_agg_filter()
get_alpha()
pick()
properties()
remove()
set()
update()
....
Line2D
drawStyleKeys : list
drawStyles : dict
fillStyles : tuple
filled_markers : tuple
lineStyles : dict
markers : dict
stale : bool
zorder : int
...
axes()
contains()
draw()
get_color()
get_dash_capstyle()
get_xdata()
get_xydata()
get_ydata()
set_linestyle()
set_linewidth()
set_mfc()
set_mfcalt()
set_ms()
set_picker()
set_pickradius()
set_solid_capstyle()
set_solid_joinstyle()
set_transform()
set_xdata()
set_ydata()
update_from()
...
YAxis
axis_name : unicode
label_position : unicode
major
minor
stale : bool
...
contains()
get_data_interval()
get_minpos()
get_text_widths()
get_tick_space()
get_ticks_position()
get_view_interval()
set_data_interval()
set_default_intervals()
set_label_position()
set_offset_position()
set_ticks_position()
set_view_interval()
tick_left()
tick_right()
XAxis
axis_name : unicode
label_position : unicode
major
minor
stale : bool
...
contains()
get_data_interval()
get_minpos()
get_text_heights()
get_tick_space()
get_ticks_position()
get_view_interval()
set_data_interval()
set_default_intervals()
set_label_position()
set_ticks_position()
set_view_interval()
tick_bottom()
tick_top()
Text
stale : bool
zorder : int
contains()
draw()
get_color()
get_rotation()
get_rotation_mode()
get_size()
get_stretch()
get_style()
get_text()
set_fontname()
set_fontproperties()
set_fontsize()
set_position()
set_rotation()
set_size()
set_style()
set_text()
set_wrap()
set_x()
set_y()
update()
....
_AxesBase
artists : list
axes
axison : bool
collections : list
containers : list
lines : list
mouseover_set : set
name : unicode
tables : list
texts : list
title
xaxis
yaxis
....
add_artist()
add_collection()
add_container()
add_image()
add_line()
add_patch()
add_table()
apply_aspect()
autoscale()
autoscale_view()
axis()
draw()
get_images()
get_legend()
get_lines()
get_yaxis()
has_data()
hold()
in_axes()
invert_xaxis()
invert_yaxis()
ishold()
set_facecolor()
set_figure()
set_xscale()
xaxis_date()
xaxis_inverted()
yaxis_inverted()
....
Tick
axes
gridOn : NoneType, bool
gridline
label
label1
label1On : bool
label2
label2On : bool
set_label
stale : bool
tick1On : bool
tick1line
tick2On : bool
tick2line
apply_tickdir()
contains()
draw()
get_children()
get_loc()
get_pad()
get_pad_pixels()
get_tick_padding()
get_tickdir()
get_view_interval()
set_clip_path()
set_label1()
set_label2()
set_pad()
update_position()
_ImageBase
axes
iterpnames
origin : NoneType
stale : bool
zorder : int
can_composite()
changed()
contains()
draw()
get_filternorm()
get_filterrad()
get_interpolation()
get_resample()
get_size()
make_image()
set_alpha()
set_array()
set_data()
set_filternorm()
set_filterrad()
set_interpolation()
set_resample()
write_png()
Axes
aname : unicode
ignore_existing_data_limits :
legend_ : Legend
arrow()
bar()
barbs()
contour()
contourf()
fill()
get_title()
get_xlabel()
get_ylabel()
plot()
specgram()
spy()
triplot()
...

Script layer (pyplot):

- Thin wrapper layer around Artist layer, providing an interface to simplify the process of plotting diagrams
- Passing arguments to corresponding classes in Artist/Backend layer by using *args and **kwargs feature in python
- When pyplot module is loaded, it parses a local user preference configuration file for a default backend. (AGG, AtAgg, At window)
- During plotting, check internal data structures for current Figure instance and plotting to the proper Axes.
- Can force the Figure to render using default GUI backend in configuration file and raise any figures created to the screen.

Software Development Process

Chosen Process

For our software development process, our team chose to work with the plan-driven **Waterfall** methodology. When considering the product, Matplotlib is a large, open-source project with robust features for generating and exporting charts using Python. As such, the requirements for the project are highly unlikely to change within the allocated time frame of this project. Consequently, the fixed and stable nature of the requirements make Waterfall an ideal choice for this project (as opposed to incremental, agile processes). Furthermore, when analyzing the structure of the project itself, the deadline based "deliverable" system employed by the CSCD01 course makes a plan-driven methodology such as waterfall more effective, since the plan is primarily outlined by the deliverable instructions themself. Similarly, the rigidity of the deliverable deadlines emphasize the necessity for excellent planning and software modelling.

By selecting Waterfall, we will be following the multi-step process accordingly:

1. **Specification - Requirements Gathering, Use-case documentation (where applicable)**
    - The deliverable requirements are analyzed. Based on the requirements, a feature/bug is documented (ie. bug description or high-level use-case document) in markdown
2. **Specification - Software modelling, Architecture**
    - Appropriate UML diagrams are created using draw.io for the relevant components affected by the bug/feature, as described by the documentation created in the previous phase
3. **Development**
    - Code is written according to the specifications previously documented
4. **Code Review & Testing**
    - Test cases are written by each developer for their own code
    - Written code is reviewed by at least 1 other peer. Reviewer may add on to the test suite if he deems it necessary
5. **Release**
    - The deliverable is submitted along with the pull requests

Modifications & Tools

One of the downsides of Waterfall is the lack of a development visualization tool which occurs standardly in many of the incremental software development processes such as scrum burndown charts or kanban boards. To combat this issue our team decided to use Gantt Charts, which are additional tools to help plan out and visualize the waterfall process. This will help the team better understand their own expected duties as well as the teams overall plan, and will be done through a collaborative Google Sheets medium. Also, in order to fulfill the documentation requirements for the Waterfall style of development, two tools will be used. The first is draw.io which has Google Drive integration which allows faster collaboration than github, and provides a good space for designing UML diagrams. In order to write requirements and other such documentation markdown files will be used. This is still fast and simple, and allows for formatting which will be seen in GitHub that a plain text file cannot offer.

As per the constraints of this project, waterfall will be modified as to have the same team working on all 5 phases of development, unlike in the usual work environment where it would be passed on between teams. Also, due to the short nature of this project timeline, we will scale it down so that it is not as intensive with regards to the amount of required documentation, requiring just simple requirements be made as well as uml diagrams when necessary. Also due to time restrictions if any changes are necessary then documentation will be edited retroactively rather than starting the process again in a waterfall way.

Other Processes

Incremental Agile

As per the course resources, incremental development (agile) is best suited for applications with constantly changing requirements, as was the case in both CSCB07 and CSCC01. However, this is not the case for CSCD01. Likewise, one of the major proponents of agile development is involving the client within the iterations of the software development lifecycle, allowing for quicker feedback. While this can be critical for the satisfaction of a client, contributing to a FOSS project does not involve a client at any point in the development process.

Reuse-oriented

There are a number of reasons why this project will not be using a Reuse-Oriented process model. The primary reason comes from the type of project this is, as the goal is to fix bugs and implement a completely new feature for the FOSS. Also, since this project is desktop python based, rather than being a mobile or web based the utility of reuse-oriented software development diminishes. Finally, cost should not be a concern when developing a school project. Due to the potential pricey nature of acquiring COTS, a reuse-oriented process should not be considered.