SWE621 – HW5

Group:

Venkata Krishna Chaitanya Chirravuri **(G01336659)**

Sai Prashanth Reddy Kethiri **(G01322333)**

Reference System – VisTrails

Additional Systems – Visualization Tool Kit (VTK), matplotlib

VisTrails

1. This is a feature from the GUI part of the system that lets you thoroughly explore the parameter space and quickly identify the desired settings. The programming styles followed by this part of the system are **Hollywood / Inversion Of Control** & for internal implementation it follows **Pipeline** programming style**.** In Hollywood programming style, elements are called indirectly through interfaces. Coming to pipeline programming, the problem is decomposed into multiple functions, where output of one function is taken as an input for another function.

   The **Motivation** for using these programming styles is, while exploring workflows, one critical task is tweaking parameter values to improve simulations or visualizations. This operation of changing parameter values is to be done by a user by interacting through a user interface. The input of the user should be taken and mapped according to the required operation of the user.

   So, the developers came up with a **solution** to design a UI for this operation where a simple user can change the parameter values, without knowing the internal operations and can get multiple different views of the data. From UI level, the user clicks on the Exploration button in the VisTrails toolbar. The Parameter Exploration view starts out with a blank central canvas wherein exploration parameters can be set up. The Set Methods panel contains the list of parameters that can be explored, the Annotated Pipeline panel displays the workflow to be explored and helps resolve ambiguities for parameter settings, and the Spreadsheet Virtual Cell aids users in laying out exploration results in the spreadsheet. To add parameters to an exploration, simply drag the corresponding method from the Set Methods panel to the center canvas. To reduce clutter, this panel only shows the methods for which parameters were assigned values in the Pipeline view. After dragging a method to the exploration canvas, you can, for each parameter, set the collection of values to be explored and the direction in which to explore. To run a parameter exploration, click the Execute button in the VisTrails toolbar or select Execute from the Workflow

menu. All the button clicks and operations performed by the user will indirectly call the objects in the code to execute the function. For example, Set Methods in UI will call the setValue method in the code to set the parameters.

The **constraint** imposed by these programming styles on the system is, there is no global variable since there is a usage of pipeline programming style. This is because there is no shared state in pipeline programming. Vistrail's parameter exploration program consists of following classes and functions to solve the problem,

- class ParameterExploration(DBParameterExploration): Has a copy of default parameters.
- def get_dims(self), def set_dims(self, d): Gets and sets the dimensions.
- def get_layout(self), def set_layout(self, l): Gets and sets the layout.
- def collectParameterActions(self, pipeline): Return a list of action lists corresponding to each dimension

Major **advantage** of having the feature of parameter exploration is, it helps in comparing a single workflow or multiple workflows by trying out different parameters. By making use of this hollywood programming style, it even made it easier for users with not much programming knowledge to tweak the parameter values, for example consider a doctor who would like to visualize an X-ray (scientific visualization), of a human at different levels. This feature would really be helpful in such scenarios. Having a pipeline programming style would help a developer to test the working of the system easily and can also help in achieving the modularity in code.

The **disadvantage** of building a GUI using hollywood programming style is the user must remember the parameters with which to connect to the database. Once all the parameters are set and the workflow is displayed, Spreadsheet does not allow you to name your analogies. Having a pipeline programming style could make it difficult for a developer to debug the code.

Source -
https://github.com/VisTrails/VisTrails/blob/9b42ca9b3550b599467b0c7dfa56f57bbd97a4f2/vistrails/core/paramexplore/paramexplore.py

2. The **programming styles** that we encountered are **Procedural & Things / Object-Oriented styles**. Procedural programming style is the one where a complex problem is divided into multiple procedures (in our case 'Functions') that share a state through a global variable . Things / Object-Oriented programming style is the one where a problem is decomposed into things ('Classes') that consist of operations and have a state to solve a problem.

The **Motivation** for using this programming style is, working with workflows requires connecting to the database for the back-up or to store, retrieve and update previous workflows. MongoDB offers different types of operations like insert, replace, delete, update, aggregate, find, drop, rename, count, group. Writing a big chunk of code in a single class or function does not make sense, because each operation is performed in its own way and provides different outputs.

Instead of a code that has everything in one class or a function (Monolithic) , this can be **solved** by making use of both procedural and object-oriented programming styles**.** VisTrails has come up with a solution where each operation has its own function (insert, find, …), such that each program has an input and their output which makes changes to the global variable (in our case 'Module').

The **constraints** that were imposed to engage these programming styles are, there should be a shared state i.e., a global variable. And this global variable must be hidden and can only be accessed using the functions in that code. In this system 'Module' is our global variable, which must be hidden and can only be accessed using the functions. "MongoDatabase, MongoCollection, DropCollection, RenameCollection" are the modules in the program.

The main **advantage** of adopting these programming styles are, it makes the code portable. Making use of classes and functions in the code resolves the problem of writing the code repeatedly, enhancing reusability. Writing code following these styles will also help in memory management, by making use of less resources. The flow of a program can also be tracked easily by a developer, helping in the process of debugging. Having the code with multiple functions and procedures helps to achieve modularity i.e., an efficient code in less number of lines. Having separate functions for each functionality of the database (insert, find, update, delete, …) makes the testing easier for the developer and also increases readability of the code. Storing the workflow to a database can also help in collaboration with other developers.

The only **disadvantage** we observed by adopting these styles is having multiple functions can be a problem while debugging as the developer must have to remember multiple method calls, which could be an additional burden. Having multiple objects can also pose a problem of more memory usage at times, making the system slower to run.

Source - https://github.com/VisTrails/VisTrails/blob/v2.2/vistrails/packages/mongodb/init.py

VTK

1. In VTK we define the pipelines similar to workflows defined in the VisTrails. There are varied possibilities of choosing the correct parameters for defining the pipelines in the VTK to achieve the desired results. We feel this is the closest **equivalent** problem in VTK as in VisTrails. Choosing the outline, contour, planes, lines, polygons are described as parameter selection.

   Hollywood/Inversion of Control and **pipeline** are followed for parameter selection for defining the pipelines where as their functionalities are defined in classes such as:
   - vtkInformation: vtkInformation follows a map-based programming style that supports heterogeneous key-value operations with compile time type checking. When passing information up or down the pipeline (or from the executive to the algorithm) this is the class to use.
   - vtkDataObject: Is a pipeline attribute containing information about a specific pipeline topology
   - vtkAlgorithm: This class follows the lazy river style where the streamed data is transformed using the desired algorithm
   - vtkExecutive: This has used the procedural and lazy river approach in defining the execution of the pipeline and storing their details.

   Similar to the VisTrails,the **solution** is provided by having the GUI developed and has enabled parameter selection by hiding the implementation details of their respective functionalities. Indirectly calling the parameters helps reduce the risk of avoiding unwanted scenarios in altering the main functionality of the classes.

   Constraints imposed by Hollywood programming style on the system are internal objects that are unknown to the end user. This programming style achieves abstraction by hiding the implementation details and providing an interface to the user. Objects from the UI are mapped to the implemented objects making the calls indirect.

   The **pros & cons** of using both Hollywood and Pipeline programming styles is the same as that of our main system.

   Both systems when **compared** followed the same programming style in their respective use cases and solved their problem efficiently.

2. The **equivalent** scenario problem we found is updating, deleting and modifying the pipelines in the VTK. We do this to optimize the pipeline flow to get the desired visualization out of it.

Object Oriented programming styles are followed to overcome this problem. The functions/methods are so defined to be individual existence and have related with linking up to achieve the defined operation.

CRUD operations are very sensitive to be handled, so the methodologies are so defined in a way not to break any dependencies on other functionalities nor crash the whole application. We change the parameters of modifying the contours, planes, lines with the functions defined inside the VTKInformation Object. ANd the updated algorithm can be specified in the VTKAlgorithm and the rest pipeline can be executed as it was. Using the object-oriented functions gives functional independence for each procedure and defining a pipeline execution helps in completing the desired operation. Small changes to the pipelines can be simply made by exempting one of the procedures and including the other required procedure.

https://github.com/Kitware/VTK/blob/master/Wrapping/Python/vtkmodules/util/vtkAlgorithm.py : The VTKAlgorithm class is created and methods are defined with independent existence that does not affect other methods operations.

https://github.com/Kitware/VTK/tree/master/Utilities/Upgrading

Constraints imposed by object-oriented programming style are, problems have to be decomposed into several subproblems and written in classes. Each class has its own functions that perform specific operations and those operations can only be accessed by the objects of that class. There will be a global state variable that is hidden and only accessed by the objects that perform specific operations including that variable.

Advantages of using OO Programming is, it makes the code more reusable and easy to debug for the developers. This also achieves modularity in the code. Code written in this way makes the system work efficiently and can solve the problems effectively.

Disadvantages are that the ObjectOriented approach though has lesser dependency of the final code and provides code reusability. It still has many adverse effects of creating many combinations for each case of updation and modification

(Comparison) VisTrails is far more advantageous than VTK because, VisTrails offers more operations and these operations can be retrieved back as part of the provenance architecture, whereas VTK does not offer such extensibility.

MATPLOTLIB

1. The **equivalent problem** we have found in Matplotlib is pretty much similar to our main system i.e.., choosing the appropriate parameters with re-iterating the compositions of axes, text, tick marks, lines, legend etc in the figure to make the final visualization more appealing, specific and understandable. This selection of parameters vary from user-to-user and scenario-to-scenario based upon the requirements and the input data.

   The **programming style** here is similar to our main system, matplotlib also follows the combination of **Hollywood/Inversion of control** with **pipeline**. But we also observe that matplotlib is structured and developed along with injecting few dependencies from other modules making it follow the **aspect** programming style. There are procedures defined in other modules and have been used as indirectly hiding their implementation details.

   Matplotlib provides a **solution** by following layered architecture where the scripting layer acts as the interface layer interacting with the Artist to render the figure onto the figurecanvas. Mainly, Matplotlib has two APIs to work with one being a MATLAB-style state-based interface and other being a more powerful object-oriented (OO) interface. The former MATLAB-style state-based interface is called pyplot interface and the latter is called Object-Oriented interface. Matplotlib.pyplot provides a MATLAB-style, procedural, state-machine interface to the underlying object-oriented library in Matplotlib. Pyplot is a collection of command style functions that make Matplotlib work like MATLAB. Each pyplot function makes some change to a figure - e.g., creates a figure, creates a plotting area in a figure etc, all these changes can be altered until the user is satisfied with the final version of the visualization.
   draw(): Redraw the current figure. This is used to update a figure that has been altered, but not automatically re-drawn.
   Few of the parameters that could be defined by the user using the pyplot interface would be:
   - **figlegend(*args, **kwargs):** attaching the legend to the figure plot
   - **axes(arg=None, **kwargs):** defining the axes of the figure plot
   - **subplot(*args, **kwargs):** describing the type of subplot and their hyperparameters
   - **xlim(*args, **kwargs):** defining the limits of the x-axes
   - **ylim(*args, **kwargs):** definng the limits of the y-axes
   - **xticks(ticks=None, labels=None, *, minor=False, **kwargs):** describing the xticks of the figure plot

- **yticks(ticks=None, labels=None, *, minor=False, **kwargs):** describing the yticks of the figure plot

- **colorbar(mappable=None, cax=None, ax=None, **kwargs):** describing the color selection of the figure plot

- **clim(vmin=None, vmax=None):** defining the color hue limits of the figure plot

- **figtext(x, y, s, fontdict=None, **kwargs):** Describing the text, which has to be part of the figure plot

And the abstractions of other modules like axes, color, grids, widgets, text, lines etc modules are imported and have been utilized hiding their implementation details explicitly.

Parameters can have bounds and **constraints** and the result is a rich object that can be reused to explore the model fit in detail. makeparams() makes use of following constraints to to build default parameters,

- value : Numerical Parameter value.

- vary : Whether the Parameter is varied during a fit (default is True).

- min : Lower bound for value (default is -numpy.inf, no lower bound).

- max : Upper bound for value (default is numpy.inf, no upper bound).

- expr : Mathematical expression used to constrain the value during the fit.

This allows you to set not only a default initial value but also to set other parameter attributes controlling bounds, whether it is varied in the fit, or a constraint expression.

The **pros** of using both Hollywood and Pipeline programming styles is the same as that of our main system. Upon these, using Aspect programming style provided an added advantage of plugins, where we can add required service without knowing the internal implementation. This is done by mpld3 that has the ability to add plugins to your plots. This enables several plugins by default, namely the Reset button, the Zoom button, and the BoxZoom button.

Even the **cons** of using Hollywood and Pipeline programming styles match with our main system, adding the aspect programming style has

bought additional burden. Having a lot of plugins in matplotlib made it heavily reliant on other packages, causing dependency problems.

(Comparison) I see both systems have their own set of advantages and disadvantages, and have been implemented according to their use cases. Bsystems have defined their better solutions

Source - https://github.com/matplotlib/matplotlib/blob/main/lib/matplotlib/pyplot.py

2. The **equivalent problem** we have found in Matplotlib is similar to our main system i.e.., operations on the figure plots like updating, saving, deleting, subplotting etc. The main motive is that the figure plot changes accordingly with the changes in the data. Not all operations are required and be needed for a simple change in the data, so each operation should perform independently of the other operation and try not to alter the other designs/functionalities of the plot, else sometimes making unnecessary changes causes unwanted outcomes.

   Matplotlib is designed as an **object-oriented** library, making it following the object oriented programming style. Following this paradigm provides a means of structuring so that properties and behaviors are bundled into individual objects. This particular programming style is characterized by the identification of classes of objects closely linked with the methods (functions) with which they are associated. It also includes ideas of inheritance of attributes and methods.

   The best **solution** is provided by defining the operations/methods explicitly as objects, which enables reusing the defined code in the later part of the codebase whenever needed, reduces risk and complexity by hiding certain parts of the code from the user, preventing unintentional modifications.

   All of these dedicated methods/operations are defined across various classes in different modules. Some of them are defined in the figure.py where the operations are :
   - figaspect(arg): defined to alter the dimensional aspects of the figure.
   - savefig(self, fname, *, transparent=None, **kwargs): Saving the figure in the specified formats

   Other methods defined at lines.py:
   - update_from(self, other): copies the figure properties to self and alters the changes accordingly.
   - set_linestyle(self, ls): sets the line styles.
   - set_lineewidth(self, ls): sets the line width.

Other methods defined are:

- draw_idle() method, request a widget redraw once the control returns to the GUI event loop.
- canvas.draw() method is defined to represent the figure after enabling all the changes/updating the parameters.

The only **constraint** imposed by this programming style, state has to be hidden and can only be accessed by the methods of that class. Consider the class FigureBase, which has only one global variable “artist”. This is only being accessed and shared by the methods of that class, which include _get_draw_artist(), get_children(), add_artist.

The **advantages** on the system by following this programming style are, it allows users to create numerous and diverse plot types. Each operation, like creating subplot, saving the plot, updating an existing are independent operations and having separate procedures and methods for these operations could make the code more readable and easier for the developer to debug. This also helps in parallel development, where one developer can work on one set of operations and another developer can develop another code for a different operation. Having separate procedures for the operation also ensures modularity and is easier to maintain.

The **disadvantage** of writing code following this programming style is, having separate procedures for each operation can make the code lengthier. This results in high run time of the code i.e., slower execution. This makes the code inefficient.

Comparing both the systems, VisTrails has the feature of saving and retrieving from the database. Whereas in matplotlib which in general is a plugin, requires working with other plugins to achieve this feature. This brings the problem of dependency.

Source - https://github.com/matplotlib/matplotlib/blob/main/lib/matplotlib/figure.py

References

- https://www.vistrails.org//index.php/Main_Page
- https://github.com/VisTrails/VisTrails
- https://vtk.org/doc/nightly/html/
- https://github.com/Kitware/VTK
- https://matplotlib.org/stable/
- https://github.com/matplotlib/matplotlib