VisualFlow Tech Review

Group 33: Connor Sedwick, Behnam Saeedi, Collin Dorsett

Abstract

The purpose of this document is to compare and contrast technologies that may be implemented in the development of our graphical user-interface system. The technologies reviewed range from basic programming languages to third-party APIs developed to aid in GUI design and development. Technologies are reviewed and chosen on how well they fit with the design of our graphical user-interface system.

Contents

1 INTRODUCTION

This document explains some of the reasons to decisions made by VisualFlow team in order to create the WYSIWYG TensorFlow graphical user interface. The topics which this document will go through are:

- development environment for graphical user interface (Behnam Saeedi)
- variable handling (Connor Sedwick)
- function support(Collin Dorsett)
- abstraction (Connor Sedwick)
- commenting and documentation (Collin Dorsett)
- debugging (Collin Dorsett)
- multi-staged programs with multiple executable handling (Behnam Saeedi)
- methods of result representation (Behnam Saeedi)
- GUI framework (Connor Sedwick)

This document will explain the underlying pros and cons to each proposed solution for these nine problems. This document is meant to act as an aid in deciding which software would be most beneficial to building our project while accommodating client expectations for the TensorFlow WYSIWYG graphical user interface.

2 Development Environment for Graphical User Interface

Our project is to create a WYSIWYG interface for machine learning. WYSIWYG (What you see is what you get) is a graphical user interface that helps users accurately visualize how their final project looks. This task is not possible without an elegant graphical user interface. Our software's user interface needs to be easy to develop. Furthermore, our graphical user interface needs to support the underlying library. The back-end of the graphical user interface is in Python since the TensorFlow library is released for Python language. In our solution, Extensible Markup Language, C++'s Open Graphics Library and Python are few of our options in addressing this problem.

XML or Extensible Markup Language is a markup language for programmatically formatting text. XML was designed with having simplicity, generality and usability as its goals. [**?**] This language is very versatile for websites and supports many features that could be helpful for designing our graphical user interface. This tool is both human and machine readable. Furthermore, it takes care of image and graphics formatting making it easy for developers to generate graphical user interfaces using this tool. However, there are a few disadvantages in using XML. The syntax is redundant, large and verbose compared to binary that could be rendered through C++ or Python. [**?**]. Another disadvantage in using this format is the fact that our library is designed for Python. This makes it difficult to create one of the main functionalities of our solution for the WYSIWYG project. That functionality is to run the current project for debugging purposes.

OpenGL is an API for creating 2-dimensional and 3-dimensional vector graphics images. [**?**] This tool allows creation of 2D and 3D shapes, rendering and UI creation. OpenGL has a few benefits over XML including the C++ code base, object orientation and C syntax. [**?**] Furthermore, C/C++ supports system calls. System calls make it possible to execute and run anything that the underlying OS supports. However, it does not completely address the complexity of communicating with Python's TensorFlow API. Furthermore, the system calls are not platform independent and change based on the OS.

The last solution to the GUI for WYSIWYG was picked in order to address the weakness of the first two solutions. Python provides a rich platform for creating graphical user interfaces. Python allows the handling and testing of the TensorFlow API. Furthermore, it is a full programming language similarly to C++. Likewise, Python supports system calls similarly to C/C++.

In conclusion, the graphical user interface is the most important factor for the success of our project. To handle this important part of our project we picked Python. This makes creating the graphical user interface much easier while not sacrificing any of the possible features that the other tools have to offer.

3 Variable handling

Our software is meant to handle a lot of values. It must also handle many different types of values. For our software to run, it needs to be based off of a programming language that makes it easy to implement variables and supports various data types. As it stands, there are many programming languages and each implements variables in different ways. This section will discuss the Python, C/C++, and Java programming languages and our decision on what language to use for our software and the reasoning for it.
C and C++ offer very traditional syntax when it comes to variables. The C language is also supported on nearly every device. With the requirement of our software being portable there is a definite plus to using this programming language. That being said, let us look at the syntax for it.

```
// C/C++ supports multiple variable types

// To create a variable in C, you must define its type like so:

int integerVariable;
integerVariable = 4;

char characterVariable;
characterVariable = 'c';

float floatingPointVariable;
floatingPointVariable = 4.5;

double veryLargeNumber;
    veryLargeNumber = 10000000;

String stringVariable; //This one is only supported in later versions of C.
stringVariable = "Hello World";

//Example of passing variable to function

int functionEx(int temp, char temp2, float temp3);
functionEx(integerVariable, characterVariable, floatingPointVariable);
```

C/C++ supports a number of variable types and has very simple syntax for defining variables. C/C++ is what is referred to as a strong typed language. This means that the variables must be defined and given a data type in order to be used. [?] Function arguments in C/C++ must also be defined with their corresponding data type.
Like C/C++, Java is also supported on a host of devices. Unlike older versions of C, Java comes packaged with string support. Interestingly enough, Java and C/C++ share much of the same syntax.

```
// Java is similar to C/C++ when it comes to defining variables

// To create a variable in Java, you must define its type like so:

int integerVariable = 11;

char characterVariable = 'c';

//To use floats in Java, they must be type-cast.
float floatingPointVariable = (float)4.5;

//Double also acts as a floating point number.
double veryLargeNumber = 3.4;

String stringVariable = "Hello world";

//Java also supports boolean values
bool qualifier = true;
bool disqualifier = false;

class Fun_example{
    public int exampleFunc(int n);
}
```

Whereas older versions of C (89 or lower) require the user to define and then provide values to variables, Java can do it all in one line. [**?**] C does have the upper-hand in this situation when it comes to defining floating point variables as it does not require the use of type-casting which could prove troublesome when implemented and translated from a GUI. Our GUI must support user-defined objects or classes. Java requires that functions be defined within classes.[**?**] This feature in a sense forces users to create classes for organizational purposes of their functions. Considering that our software is meant to support object-oriented designs this could prove to be a very useful choice over C.
Compared to C and Java, Python widely varies in its implementation guidelines. Python, being a dynamic programming language offers much flexibility for the user. To put it in perspective an example is provided below for the syntax of variable defining in Python.

```
integerVariable = 11

characterVariable = 'c'

floatingPointVariable = 4.5

stringVariable = "Hello world"

qualifier = True;

disqualifier = False;

#function definition
def printinfo( name, age ):
        "This prints a passed info into this function"
        print "Name: ", name
        print "Age ", age
        return;

#call printinfo function
printinfo( age=50, name="miki" )
```

Take note that there is no required defining of variable type before setting the variable to a value. [**?**] This makes variables in Python very easy to implement. Considering that our GUI software will have users creating variable blocks for the flowchart to use in their functions it seems much better to have an overarching ”variable” block. In the case of using C/C++ or Java, separate blocks for each type of variable would need to be created for our menu. Similarly, function parameters in Python are much simpler than Java and C/C++. Instead of setting hard parameters that must be met by the value being passed to a function, Python just requires that a value is passed. [**?**] For these reasons, we advice using Python as the primary language for variable handling.

4 Function Support

The first option we looked at for function support was to allow the user to create their own custom functions, or in this case, their own Block types. These Blocks would be completely customizable for whatever the user's program needs, giving the user more control over their program's design. Our reason for considering this option is that the user may need additional features for their Blocks that are unique to their program. However, this means that it is possible the user may create a Block that simply does not function with the rest of the program, or contains features that other Blocks are not capable of accommodating. Additionally, giving users the freedom of custom Block types adds another layer of complexity and somewhat strays away from our goal to keep the program as simple as possible.

The second option we considered for function support was to have all Block types be built-in prior to the initial installation. These Block types would cover a broad spectrum of features and properties that the user will have access to when designing their program. Our reason for considering this option is that our main goal is to keep our program as simple as possible with regards to the user interaction. This particular option effectively makes using our program much more simple for the user since they will not need to worry about creating their own Blocks and instead are given a list of predetermined Blocks they can use. However, this solution runs into some issues. If the user needs a certain Block for their program and that Block is not in the built-in list, they will not be able to apply such an implementation to their program. Additionally, the user may not need all of the Blocks in the built in list, which could make finding and utilizing these Blocks a hassle for the user.

The third and last option we examined for function support was to offer a minimal amount of Block types during installation, and to include prepackaged libraries with additional Block types. These prepackaged Block types would apply much more complex and sophisticated functions compared to the Block types already installed. Our reason for looking at this option is to give the user a basic amount of Blocks to build their program with, enough so that any user can be adequately accommodated. If a particular user needs different Block types, they can simply install one of the related libraries and continue building their program. Of course, this option does run into some glaring issues. Once the user has installed the packaged library, those Block types will remain in their list of installed Blocks. These Blocks could take up space and make it more difficult for the user to find less complex Blocks. On the other hand, this option could allow for the user to simply uninstall those packaged libraries once they have finished with their program. Compared to the other options for flexibility, this particular one seems to allow for a diverse range of simplicity while keeping room for more complex operations. This option would function in a manner similar to Cygwin, where a default set of packages is installed during the initial setup and additional modules can be seamlessly downloaded as needed by the user. [**?**]

5 Abstraction

Given that the GUI that we are creating is meant to provide an abstracted view of a user's project to them, the language we use should mirror our GUI closely. Python, C/C++, and Java all support object oriented programming. Java does especially due to it requiring every function be defined within a class. All languages require constructors in order for classes to be invoked. The main difference, as usual, is the syntax:

```
#Python
>>> class Dog:
def __init__(self, name, type):
        self.name = name
        self.type = type

>>> x = Dog('Millie', 'Boston Terrier')
>>> x.name
'Millie'
>>> x.type
'Boston Terrier'
```

```
//Java
public class Bicycle{

int gear;
int cadence;
int speed;

        public Bicycle() {
                gear = 1;
                cadence = 10;
                speed = 0;
        }

        public static void showParts(){
                System.out.println("Gear = " + gear);
                System.out.println("Cadence = " + cadence);
                System.out.println("Speed = " + speed);
        }
}

Bicycle mybike = new Bicycle();
mybike.showParts();
```

```
//C++
class Line {
        public:
                void setLength( double len );
                double getLength( void );
                Line();   // This is the constructor

        private:
                double length;
        };

Line::Line(void){
length = 20;
```

```
        cout << "Object is being created" << endl;
        }

    Line::getlength( void ){
        return length;
    }

    void main(){
        Line myLine;
        cout << "Length is " << myLine.getlength() << endl;
    }
```

Constructors in Python are simple in that they are defined as "__init__()". C++ by contrast requires that constructors be defined with the same name as the class. [?] Java requires that the user call the constructor when they are initializing an object whereas in C++ it only need be defined as that object type as the constructor is implicitly called each time a variable of that type be created. [?]

Simpler implementations are preferred for our implementation. In C++, for example, unless a variable is set as public within a class it cannot be displayed unless a function is called that returns it's value. [?] This provides assurance that values cannot be deliberately changed unless through function calls that do so. Python allows users to reference values just by referring to the "self." variables set in the constructor. [?] C++ classes could require that we implement more features to our GUI to accommodate public and private values as well as special formatting for public versus private functions.

C++ and Java are very similar in their syntax. Both require functions to access and display class variables. Likewise, they support public and private variable types. [?] [?] By contrast, Python does not use private variables. This simplifies accessing values and effectively decreases the amount of code required to access each individual variable. If a user were to create a class in our GUI using the C++ implementation it would require that they create a function for every single private variable of that class. Depending on the object they are trying to model that could require numerous functions just to access basic variable values. This compounded with the pieces representing variables alone would easily overpopulate the interface.

Keeping in mind that functions defined for classes in C++ must also include scope prefixes. This would require that we have a way of storing the class name as a prefix to append to class functions. Python instead requires all functions used by a class be defined and written under the class header. [?] Here it merely comes down to a styling choice. C++ does a better job of organizing the parts that make up each class into sections containing public and private function definitions and variables. Python makes each class a giant listing of functions and variables. The other important aspect to consider is whether our software can support a deep learning API. Considering the previously discussed benefits and the fact that Google's TensorFlow API is written to support the Python programming language it is obvious that Python would be best suited for our purpose.

6 Comments and Guides

The first option we considered for commenting and guides is to include comment blocks within our code. Our design of these comment blocks would consist of including helpful comments scattered throughout our code that would give the user some guidance on how the inner workings of the program functions. These comments would be placed in areas where the user will be fully able to understand just what is happening in their program. The reason we chose to consider this option is that it will keep more technical information in the background, away from the user and allow them to focus on the more simple interface of our program. Compared to the other two options, this one remains rather simple because the user has the option to seek out these comment blocks while also able to run their program as intended. However, these comment blocks may not be easily accessible by the user if they are not aware of how to access them. Additionally, if the user is able to access these comment blocks, they are at risk of changing some of the program's code, either accidentally or on purpose. This could prove to be a major issue since it may impact the results of their program.

Our second option we looked at for commenting and guides was to include reference datasheets. These datasheets would function similar to man pages, and would give the user an overview of certain information. Some examples of information included in datasheets would be various types of Blocks, how a certain method operates, or what the properties of a specific class are. Our reasoning for looking into reference datasheets is that it will keep all relevant information pertaining to a particular topic in a single place. This means that the user will not need to go searching through code for specific information. Due the the general overview of datasheets and the large amount of data they can carry, the user may still have a difficult time locating the information they are searching for. Creating reference datasheets also requires the use of certain typesetting. A program that could help with this is nroff, a Unix text formatting program. This particular program is used in the Unix help system to format and display man pages. [**?**]

The last option we considered for commenting and guides is to include helpful comments inside Blocks. These comments will help the user understand what is happening within the Block, without interrupting or changing their program. Similar to the reference datasheets, these Block comments would include information such as Block type, methods used, and variables passing through the Block. Our reason for looking at this particular option is that we want to integrate helpful information into the program itself, rather than having the user search through the code or comb through datasheets. Compared to the other options, this one allows the user to see the information on the go and it will always be relevant information. Additionally, this also means that the user will not be searching through the code for comment blocks, and thus will prevent them from accidentally changing any code. Compared to the other options for commenting and guides, this particular option keep our program simple and straightforward. Similar to the probe option in the debugging section, this option could make use of either Tkinter or Pyglet modules to give the user the information they need.

7 DEBUGGING

The first option we have selected for debugging purposes is the concept of probes and probing. Our design of the probe would be to allow the user to place a special Block type which displays values of variables and data passing through that particular Channel. In addition to simply displaying the content of the Channel, we would also allow users to modifying existing variables at that point in the Channel. Essentially, this means that the user can feed in certain variables and data directly to other Blocks during runtime. The reason we considered this particular design for debugging is so that we can give the user a simple but highly effective means to debug their program. Compared to other debugging options we have examined, the probe is by far the most informative and give the user more control of their program. The ability to change variables and data on the fly as well as determining their output allows the user to quickly view mistakes and make subsequent fixes. On the other hand, the probe does present additional issues that are not prevalent in the other options. One issue is that the user may accidentally forget to turn off or remove a probe. This means that whatever variable they have modified at that point will be fed into the program, and could very well result in incorrect results. One piece of technology we could use for this option is the Tkinter module for Python. This particular module was created specifically for use in GUI application development, and is available on most Unix and Windows platforms. Additionally, Tkinter documentation is very thorough due to many developers utilizing it. [**?**]

Our second option for debugging is to simply print out variables and data for the user to view. Like the probe as mentioned previously, this design would allow the user to view different variables and data passing through that particular Channel. Unlike the probe, using this option means that the user would be unable to modify variables and data during runtime. The reason that we chose this particular design option is that it remains fairly simple in that it gives users the option to view variables and data passing through a Channel. Compared to other debugging options we have examined, this one remains informative while limiting the user's access to controlling the program. At first this may seem like a downside, but our goal is to stay simple, even if it removes the option for variable and data manipulation. Similar to the probe option, this particular option could also utilize the Tkinter module. Another module could also prove equally as useful, the Pyglet module. Pyglet's simplicity makes it much more easy to use than some other modules, and it has also been shown to be quite fast with regards to processing speed. Despite this, Pyglet's documentation has suffered from a lack of maintenance online, where many tutorials and documents remain out of date, and may not be usable. [**?**]

Our third and last option for debugging is to not display any variables or data to the user. Our reasoning for considering this option is to keep our program even more simple than the previous options. The other options we have examined is to give the user variables and data that can be manipulated to change the results of the program. However, the other options run into issues that can be solved by not displaying any sort of variables or data. This solution removes any issue where the user may forget to leave a probe in their program or become overwhelmed with the assortment of variables and data. This ensures that everything is kept in the background, out of the user's reach. However, this also means that the user will miss out on certain aspects of aforementioned options. Obviously, choosing this option would not require the implementation of any new technology.

8 Handling Multi-staged Programs with Multiple Executables

There are many ways to implement a large project. Large projects usually have many different parts that work together in order to create the desired result. Some of these parts could be independent to prevent a complete failure within the software. [**?**] Some developers prefer to have one executable in order to generate their final results. This review will explain what approach is best fit for the purpose of multi-stage programs that rely on complex data handling. One of the requirements for the WYSIWYG project is a support for large multi-staged projects. Our client is looking for a way to handle large machine learning projects that require multiple parts. Three solutions are proposed to address this problem and each have their own advantages and disadvantages.

One of the approaches is to integrate all of the functionalities into one single project. This forces the project to be handled at once when the project is interpreted. There are many advantages and disadvantages to this approach. This approach is very simplistic and makes the project easy to move. On the other hand, the project will not be reusable, difficult to debug, patch, read or understand. Furthermore, it is much easier to create multiple small programs that work together than to create one single program that is large. This decreases the flexibility of the program too since it becomes difficult to modify. Finally, in programming, when the code grows large, there is a point at which a bug cannot be correctly addressed without introducing one plus epsilon error to the code. [**?**] At that point the program often cannot be corrected without rewriting the entire program.

Another approach to this problem is to create different projects using our solution for each stage of the system. These independent projects are treated as if each one of those is a complete project. This means that each project is going to have its own project folder with all of its necessary files in the folder. This approach helps with reusability, ease of updating and flexibility of the program. On the other hand, it is difficult to keep track of all of the related programs in the same project. Another issue with this approach is version control.

Our third solution to this problem is designed to reduce the disadvantages of both of previous systems while maintaining the advantages. Our solution introduces the of concept of layers. Each layer is independent of others and occurs in the same project. This will help with easier version control while keeping it reusable and easy to update. Some of the limitations to this solution is the complexity and management of project files.

In conclusion, our proposal to this problem is designed to improve user experience in creating large complex multi-staged projects. This solution allows users to take advantage of our tool to improve their program and make their projects easily updatable and reusable while managing their project through version control systems such as SVN and Git.

9 Result Representation

Our project, WYSIWYG, needs to be able to perform a unique task. It needs to provide a graphical user interface for generating machine learning applications in Python using the Google's TensorFlow library for Python. Our graphical user interface will mask away the underlying Python code in order to make the development seamless for the user. Our users' programming skills range from low (math and statistic majors) to high (software developers). The problem that we face is to how to represent the result of our project to the user and maximize their benefit from the projects they create using our GUI. To solve this problem we proposed three possible solutions. Our proposed solutions are displaying results in a log file, create executables or output script.

Our first solution is to only generate the result of the system. Our solution shows the output of the program ignoring the executable and code. This approach makes our tool easy to use by the users with low understanding of programming. Furthermore, our graphical user interface will have more control over the organization of the code since the code will be completely masked away. On the other hand, this approach provides little to no control over the code to the users that are more advanced in Python programming language. Likewise, moving the project from one platform or computer to another requires moving the entire project file and installing our GUI on the new computer. This problem makes the code virtually non-distributable.

Our second solution is to create an executable for a finished project. This approach has many similar advantages to the previous solution. This makes the experience easy for users that have little understanding of Python programming language. Likewise, the GUI will have a lot of control over the code organization since the underlying code is still masked away from the user. Furthermore, it solves the portability and distributability problem that the first solution had. Also, it does not need Python installed to work after the executable is generated. On the other hand, in this approach users with high level of programming skills will still not have access to the code after the project is finished. Another issue that is unique to this solution is the fact that the compiled result might not function on other platforms with different processors or operating systems.

Our final solution to this problem is to output a script. For this solution the graphical user interface provides a Python script file with .py extension. This Python script file will contain the Python code version of the project that user created. In order to solve this problem, it is important to understand and exploit some of the properties of Python programming language. Python does not compile the Python script stored in .py files, but rather interprets them. This means that a Python script file can be provided by a simple makefile to be executed like an executable. Therefore, Python script is still easy to execute for users with low Python experience. Also it allows more experienced programmers to modify the output Python script even after the project is created and finished. Since Python is an interpreter the .py files will still be portable between different systems with different processors and operating systems. This solution still has few disadvantages. The graphical user interface now needs to follow strict organizations that are human readable and human understandable with proper commenting. Finally, the user needs to install Python interpreter in order to be able to execute their final project.

In conclusion, in order to solve this problem, we decided to generate a file that contains user's project in form of a Python script. This solution will take advantage of the fact that Python interprets its script rather than compiling it. Furthermore, if we chose that it is important to give the option of generating an executable rather than a script file to the user, it would be a simple addition to our graphical user interface.

10 GUI FRAMEWORK

For our software to work as described in our requirements document and in order to satisfy our client's expectations we must choose a framework that supports our design requirements. The framework we choose must support drag and drop functionality and multiple pages (preferably a tabbing function). Considering that Python is best suited for our software we should look to using a framework for our GUI that is supported by Python. Python has many libraries that are suited for our application. A few main frameworks that are cross-platform are libavg, Kivy, and wxPython. All three choices listed are supported by Windows, MacOS and Linux systems. All three APIs also support buttons whether they appear as images or text.

Both Kivy and libavg support touch and drag functions due to them primarily being designed for mobile application and game development. [**?**][**?**] Libavg, specifically, supports multitouch functionality. While our software is meant as a desktop application, having the ability to use it on a tablet would also increase its practicality. If individuals working on a project wanted to be able to collaborate and build a solution together, supporting multitouch would also be an interesting feature to consider. wxPython supports drag and drop functionality as well. wxPython's documentation also provides examples of the implementation of a drag and drop feature. [**?**]

To support a build space function in our software, there is a need to ensure that the framework we use can track coordinates of blocks. libavg itself has what is called a hierarchy of coordinate systems. [**?**] This means that users can create divs or cut out portions of the interface that have their own relative coordinate system. This could prove useful for our build space design as it can allow us to set a relative portion of the interface to be used exclusively for node placement. Kivy supports relative coordinates as well and the website for it provides much more detail on the matter. [**?**] wxPython does not have a specific feature built in for dealing with relative coordinates. [**?**] Instead, it requires that the developer do some arithmetic in order to ensure values fall into the correct coordinate plane.

In order to make efficient use of one of the mentioned frameworks we must have some form of documentation to aid us in using it. wxPython itself has multiple books written on it, and its API manual page is host to extensive documentation. The website for Kivy also hosts a great deal of well-organized API documentation. libavg, does provide documentation on its website, but is very sparse and unorganized. Provided the timeframe we have to work on this project it would be beneficial to use an API that is well documented. Its manual should also be easy to refer to (i.e. have a search feature).

Obviously, it is advantageous to use an API that guarantees efficiency in the development process. With this in mind, using wxPython would be very time consuming as it would require much time spent developing algorithms for coordinate tracking. libavg and Kivy would appear better choices as they have support for relative coordinate systems built into them. However, when it comes to documentation, libavg comes up short when compared to Kivy and wxPython. Taking these factors into consideration it would appear most beneficial to use the Kivy API for our GUI development.