VisualFlow Tech Review

Group 33: Behnam Saeedi

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 selected based on how well they fit with the design of our graphical user-interface system.

CONTENTS

1 INTRODUCTION

This document explains some of the reasosns to decisions made by Visual Flow team in order to create the WYSIWYG TensorFlow graphical user interface. The topics which this document will go through are handling multi-staged programs with multiple executables, development environemnt for graphical user interface and lastly methods of result representation. This document will explain the underlying pros and cons to each proposed solution for these 3 problems.

2 HANDLING MULTI-STAGED PROGRAMS WITH MULTIPLE EXECUTABLES

2.1 introduction

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 complete failure of software. Some developers on the other hand prefer to have one executable in order to generate 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 was a support for large multi staged projects. Our client was looking for a way to handle large Machine learning projects that require multiple parts. Three solutions were proposed to address this problem and each have their own advantages and disadvantages.

2.2 Integrated Large System

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 average program that is large. This decreases the flexibility of the program too since it is make it difficult to modify. Finally, in programming, when the code grows large, there is a point in which after that a bug cannot be correctly addressed without introducing one plus epsilon error to the code. At that point the program cannot be corrected without rewriting it.

2.3 Indovidual Project generation

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.

2.4 Onion Approach

Our solution to this problem is designed to reduce the disadvantages of both of these systems while maintaining the advantages. Our solution is introduction 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.

2.5 Conclusion

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 keeping it managed through version control systems such as SVN and Git.

3 DEVELOPMENT ENVIRONEMNT FOR GRAPHICAL USER INTERFACE

3.1 introduction

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 user accurately visualize how their final project looks like. This task is not possible without an elegant graphical user interface. This user interface needs to be easy to develop. Furthermore, our graphical user interface needs to support the underlying library. The backend 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.

3.2 Extensible Markup Language

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 few disadvantages in using this tool. The syntax is redundant and large and verbose comparing 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.

3.3 C++ Open Graphics Library

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

3.4 Python Graphical User Interface

The last solution to 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 full programming language similarly to C++.

3.5 Conclusion

In conclusion graphical user interface is the most important factor in success of our project. To handle this important part of our project we picked Python. This makes creating the graphical user interface much easier.

4 Result Representation

4.1 introduction

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 googles tensorflow library for python. Our graphical user interface will mask away the underlying python code in order to make the development seemless for the user. Our users programming skill ranges 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 3 possible solutions. Our proposed solutions are displaying results in a log file, create executables or output scrip.

4.2 GUI Representation of Data

Our first solution was 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.

4.3 Generating Executables

Our second solution is to create an executable for users 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. 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.

4.4 Python Script

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 interpret them. This means that a python script file can be provided by a simple makefile to be executed. Therefore, python script is still easy to execute for users with low python experience. Also, more experienced programmers, now can 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 restrict 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.

4.5 Conclusion

In conclusion, in order to solve this problem, we decided to generate a file that contains users 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.

5 CONCLUSION

In conclusion there were 3 main dicisions discussed in this document. The three issues were handling multi-staged programs with multiple executables, development environemnt for graphical user interface and lastly methods of result representation. These problems were througly analyized by the Visual flow team many possible solutions were generated for each problem. Based on the group's analysis, for handling Multi-staged projects we introduced the concept of layers. Furthermore, for the graphical user interface development environment we selected python. Lastly our group decided to generate python script files as output for extraction of the project.