4-2002

A Comparison of Programming Languages for Graphical User Interface Programming

Phillip Kevin Reed
University of Tennessee - Knoxville

UNIVERSITY HONORS PROGRAM

SENIOR PROJECT - APPROVAL

Name: Kevin Reed

College: A. Arts and Sciences Department: Computer Science

Faculty Mentor: Brad Vander Zanden

PROJECT TITLE: A Comparison of Programming Languages for Graphical User Interface Programming

I have reviewed this completed senior honors thesis with this student and certify that it is a project commensurate with honors level undergraduate research in this field.

Signed: Brad Vander Zanden, Faculty Mentor

Date: 4/24/02

Comments (Optional):

Excellent job!

A Comparison of Programming Languages for Graphical User Interface Programming

by

Kevin Reed

Project Faculty Mentor: Brad Vander Zanden

Kevin Reed

A Comparison of Programming Languages for Graphical User Interface Programming

Introduction

This paper is the final component of my Senior Honors Project for the University of Tennessee Honors Program. The entire project involves taking a program that I wrote for a class last semester and rewriting the program in two other programming languages and then doing a comparison of the three languages. The purpose of the Senior Honors Project is to provide students with an opportunity to explore an area of interest on their own with the guidance and advice of a faculty mentor. I took this as a chance to learn more about graphical user interface (GUI) programming. The program I chose to rewrite is a program I wrote for a class on graphical user interfaces that I took during the Fall semester of my senior year. The class, taught by my faculty mentor Brad Vander Zanden, used the Java programming language. This provided an excellent opportunity to learn about Java programming, but I was more interested in learning how to write GUI programs in C++, so I decided to do a senior project that would give me an opportunity to do this.

The original idea that I submitted to Dr. Vander Zanden was to rewrite the program only in C++ and then do a comparative paper, focusing on my experiences writing the two programs. Dr. Vander Zanden suggested that I extend the project to include a scripting language. For the final project, the programs have been written in

Java, C++, and Python. The program is a simple puzzle game called Sokoban, which uses all the basic elements of GUI programming. All three programs create and display a game board, using a graphical display and a text view, with pieces that can be interacted with using the mouse or keyboard. The program also provides a simple menu for loading levels, resetting the game, and quitting. The original Java version of the program also had several more advanced GUI features, but due to lack of time, these have not been implemented in the C++ and Python versions of the program.

General Overview of Java, C++, and Python

Before describing my experiences using each of the three programming languages, I will provide a general description and the common uses of each language, beginning with Java. One defining feature of the Java language is that it is entirely object-oriented. By forcing programs to be written in an object-oriented way, programs written in Java have to be highly modularized. The idea behind this is to make code reusable and easier to update and maintain.

Another important feature of Java is that Java programs are designed to be platform independent. All Java programs are run through a Java virtual machine. These virtual machines work the same no matter what type of computer or operating system they are on. This allows code compiled on one type of computer to be run on any other type of computer. This platform independence makes Java well suited for internet-based applications where there is no way to determine what type of system a program will be executed on. The virtual machine also provides a level of security between programs and a host computer by restricting a programs access to memory and system devices. This is

another feature that makes Java attractive to developers and users of Internet applications. Java programs can generally be executed without fear that the program will accidentally or maliciously damage a host computer.

One final feature of the Java language that stands out is that it is designed to be a very high level programming language. Java is designed to allow developers to create programs quickly and easily, without having to deal with many of the low-level details of the program. Java handles memory management automatically using a garbage collector. It also allows for the use of anonymous classes that can be easily defined on the fly. Java also comes with a great deal of packages and utilities to make development faster and easier. The standard Java distribution comes with classes for common data structures like stacks and queues, event handling, window management, graphics, and several others.

C++ is also an object-oriented language, but unlike Java it is not strictly object oriented. The C++ language is actually a superset of the C language, and therefore can be used for both object-oriented programming and procedural or function based programming. This gives C++ a great amount of flexibility. C++ is also a very powerful and very fast language. It is less of a high level language than Java and lacks some of the built in features that Java offers, often trading ease of development for speed and efficiency. It also allows direct access to memory and system devices, also giving the language speed and a bit more power than Java, but less security. Unlike Java, C++ code must be compiled directly for the type of system that it will be executed on, since C++ code does not run through a virtual machine. Executing the code directly instead of through a virtual machine layer also speeds up the execution time of the code, but does not allow for platform independence.

C++ is used for a wide variety of programs due to its power, speed, and flexibility. Many operating systems, utility programs, commercial applications, scientific and industrial software, and video games are written using C and C++. It is one of the most widely used, possibly the most widely used, programming language today. C++ is very well suited to any program where speed and efficiency is important and works well for almost any type of programming problem.

Python, the last language I'll examine in the paper, is a scripting language. This means that the code is not directly compiled, but instead executed though an interpreter. This makes it easy to develop and test code quickly, without having to wait on a piece of code to be recompiled every time a change is made. Executing the code through an interpreter also allows Python code to be platform independent. Similar to the Java virtual machine, the Python interpreter provides a layer between the program and the operating system, so that the same program can run on any system since the interpreter will execute the code in the same way no matter what kind of system it's on.

Also, similar to Java, Python is designed to be very high level and make development quick and easy. It handles memory management and provides many useful packages similar those provided by Java. It also does dynamic variable typing. Like C++, Python has support for both procedural and object oriented programming, making it easy to develop programs best suited for one or the other type of programming. Because of its ability to do quick testing and rapid development, Python is often used for developing and testing program prototypes. It is also often used to quickly develop simple utility programs.

My Experience with Java, C++, and Python

The original Sokoban program that I wrote was done in Java. At first, when I found out that the GUI class where I wrote the program was going to be taught in Java, I was a little upset. C++ is the language I'm more familiar with, and generally the one I prefer. Also, I didn't like the fact that Java was entirely object oriented and that it was generally slower and less efficient than C++. Now that I've actually written some programs in Java, I still dislike these things but I have found several things that I do like about Java. First of all, I do like Java's syntax for the most part. It is very similar to C++ syntax and therefore very easy for a C++ programmer to make the adjustment to Java. I also thought that the documentation for Java provided on the Sun website was very good, better than the documentation provided for Microsoft's Visual C++ (the version I used for this project) or Python. I found it easy to find the information I needed, including detailed information about the language's objects and their methods and also examples of their use.

As far as actually writing the program went, I thought that Java was very well suited to GUI programming. The language provides a very good interface to the windowing system. This interface works the same for any platform. Java makes it very easy to create and manage windows and add components to those windows, giving a large number of layout types to easily arrange items within a window. The Java language also includes a wide variety of predefined widgets that are typically used in GUI programs. It is very easy to add menus, dialogs, buttons, and other interactive objects to the program. Java also makes it easy to create custom widgets by extending various pre-existing classes, adding event handling code and custom graphics. I thought that Java's

built in 2D graphics utilities were very straightforward and easy to use. Also, they include support for loading and displaying several common image formats. I did, however, run into a problem trying to draw only onto a specific part of a component without clearing the entire component and redrawing it. Most of the class, in fact, discovered this problem. When the entire component is cleared and then redrawn, however, everything works fine, and aside from this problem, the graphics code works very well. I also thought that the event handling utilities provided were easy to use and worked very well. The Sokoban program needs input from both the mouse and keyboard, and Java's event handling system makes it easy to access these devices and obtain the necessary information from them.

The primary problems that I had with Java are mostly performance and efficiency issues. Java has more overhead than programming languages that do less to handle lower level details like memory management and security issues and because of this, Java programs generally run slower and have a higher load time. The platform independence added by the Java virtual machine also adds to these problems. For the Sokoban program, however, these effects are not extremely noticeable. There is almost no discernable difference in the execution speed of the Java and C++ program, and the load time, while noticeably longer than the C++ version, is not enough to make much difference. For the Sokoban program, I thought that Java worked very well, and in cases like this one, where the tradeoffs in speed and efficiency are fairly small, Java's convenience may be worthwhile.

The next version of Sokoban is written in C++. In my opinion, this is the most important part of my project and the motivation behind it. By rewriting Sokoban using

C++, I got a chance to learn how to implement in C++ some of the GUI concepts I had previously learned in Java. Unlike Java, C++ is not platform independent and there are many different C++ compilers available. There are also no standard libraries for accessing an operating systems windowing system. For this project, I wanted to develop code for the Microsoft Windows operating systems. I used Microsoft's Visual C++ as a compiler and development environment and the Microsoft Foundation Class (MFC) libraries as an interface to the windowing system.

As I said before, C++ is the programming language that I am most familiar with and the language that I generally like best. I think it's an excellent language that provides a great deal of flexibility and allows for the development of very fast and efficient programs. One more thing that I noticed about C++ that I had never really paid much attention to before this project is the usefulness of header files. They make it much easier to define constants and make references to variables declared in other files. They also provide a great way to see the functions defined in a file or the methods defined for a certain object without having to look through all the code where they are implemented.

For this project, my primary goal was to learn to use MFC to develop GUI programs for Windows operating systems. One interesting aspect of MFC that I discovered while programming Sokoban is that it is very strongly integrated with Visual C++'s development environment, which automatically creates code for the basic framework of an MFC program based program. This code handles the creation of the main window and separate document and view handling code, with an interface between the two. Visual C++ will also automatically add event-handling code, such as mouse and keyboard events, and it also allows menu's and dialogs to be created and edited visually.

The automatic code creation and visual editing capabilities of Visual C++ have both advantages and disadvantages in my opinion. It does speed up development time and make it easier to produce MFC applications, but it can create unnecessary code and also makes it easy for a programmer to create code without knowing exactly what it is doing. Also, the code for creation and management of windows is more complicated with MFC than with Java and would make program development more complicated if it were done manually and not through the automated features of the Visual C++ development environment. The actual libraries provided by MFC worked fairly well for most things, although they lacked a great number of conveniences provided by Java. MFC has a few built in widgets – file loading dialogs, basic menus, and scroll bars, for example - but not quite as many as Java. MFC's built in 2D drawing functions also weren't quite as good as Java's. They worked well, but I thought they were less intuitive and straightforward than Java's, although font and graphical text manipulation was actually easier. MFC also lacked the built in ability to easily load and display common image formats.

Finally, I thought that the documentation provided with Visual C++ and MFC could have been a lot better. The biggest problem I had with the documentation, Microsoft's MSDN library, is that it contains not only information on C++, but also information about several other Microsoft development tools. The huge amount of information provided made it difficult to find C++ and MFC specific documentation and examples. Also, I was not able to find many good examples showing the use of MFC objects.

The final version of Sokoban that I wrote for this project was done in the Python scripting language. Like C++, Python doesn't really have it's own utilities for creating windows and graphics, but it does come with Tkinter, an interface onto the Tk toolkit that does handle standard GUI functions. As I said earlier, I had originally planned to only do this project using C++ and Java, but my faculty mentor suggested that I increase the scope of my project to include a scripting language and suggested that Python was a good one to use. Since I had never done any programming in Python, and wasn't too familiar with scripting languages in general, I agreed that it probably would be good to further enhance my education from this project by again implementing Sokoban using Python. And of course, my experimentation with Python for this project has indeed taught me something, mainly that if at all possible I will never use Python again for anything.

I stated in my general overview of Python earlier in this paper that one of the main purposes of Python, and most other scripting languages, is to make code development and debugging quick and easy, primarily for the purposes of prototyping a program. I found the exact opposite to be true. My development of the program using Python was slow and frustrating. First of all, the documentation for Python and Tkinter was not very good. Generally there were few or no examples given and often the documentation for objects did not adequately explain how those objects were used. One of the first problems I ran into - due mainly to a lack of good documentation - was with the import statement, which is needed to load and use pieces of code defined in other files. There are two ways that the import statement can be used, and both seem to work differently. I don't know what the exact difference is, but sometimes one works and at other times the other works, but usually not both.

I also had a problem getting the interpreter to recognize changes in any files outside of the main program file. One of the big advantages to scripting languages is supposed to be that changes to a program can be tested immediately without having to recompile part, or all, of the program. Instead of having to recompile, though, I found myself having to exit and restart the interpreter and reopen the script files whenever I made changes, which can be almost as time consuming as recompiling code and is just generally annoying. Other things I disliked about Python are its syntax and the way in which the language implements object oriented programming. I thought that the syntax of the language made it more difficult to read and organize than C++ and Java programs. For Python's implementation of objects, I didn't like having to reference every method and variable within an object by preceding it with self, which Python's implementation of objects requires. This produced cluttered, difficult to read code and also resulted in frequent errors.

In addition to all the problems I had developing Python code, the final program ran noticeably more slowly than either the C++ or Java version. The only good thing I have to say about Python is that does have a pretty good library of GUI tools, using the Tk toolkit. It has a good number of built in widgets, easy window creation, good event handling methods, and reasonably good, simple drawing methods. Tkinter's GUI tools are certainly no better than Java's, however, and only slightly better than C++'s using MFC at best. I personally don't see any advantages to using Python for GUI programming or development.

Conclusions

My conclusion regarding my overall project is that it has been very successful. I accomplished my goal of extending what I learned in the GUI class into other programming languages. My conclusions for the actual comparison of my experiences with these programming languages are that Java and C++ are both excellent languages, each with their own advantages and disadvantages. For web-based applications or applications where platform independence is important, Java is most likely the best choice. Java seems to have the best built in support for GUI programming, however, C++ using the MFC libraries has more than adequate tools for GUI development and may be a better choice when speed and efficiency are important.