International Journal of Technical Research and Applications e-ISSN: 2320-8163,
www.ijtra.com Special Issue 41 (AVALON) (March 2016), PP. 5-9

A COMPARATIVE STUDY OF TEXTUAL PROGRAMMING V/S PROGRAMMING WITH VIRTUAL INSTRUMENTS

Akash Tomar[1], Prerita Deshpande[2], Pallavi Nimbalkar[3], Tanvi Gurav[4], U. B. Mantale[5]
Computer Engineering Department, Mumbai University
Navi Mumbai, India
[1]akashtomar@yahoo.com
[2]deshpandeprerita@ymail.com
[3]pallavinimbalkar05@gmail.com
[4]guravtanvi@yahoo.in
[5]u.b.manthale@gmail.com

Abstract— **In a world where digitalization of day to day activities is increasing rapidly, the strain is on professional developers to fulfil the growing demands. The need for more human centred user interfaces is also rising. Not only should the device perform accurately, it should be self-explanatory to the users. A user friendly, innovative and resourceful Graphical User Interface is required to make a device or software efficient to use. This paper explores the positive and negative outcomes of textual programming and programming using virtual instruments .**

Index terms- **Virtual Instrumentation, Textual Programming, LabVIEW, Evolution of Programming, User Interface.**

I. INTRODUCTION

For more than 50 years, engineers have sought easier and faster ways to solve problems through computer programming. Furthermore, the programming languages chosen by engineers to translate their task have trended towards higher level of abstraction. This paper explores the concepts of programming with virtual tools in addition to G programming.

When software is to be created, often, the first preference of developers is a textual programming language to write the code in. Since students as well as professional developers are well acquainted with languages such as C, C++, Java, their approach is restricted and more inclined towards these textual languages. At an academic level, programmers are less exposed to different programming methods, thus limiting their use to a particular textual programming language.

However at an industrial level, where a project completion deadline is as important as the actual of the project itself, professional developers experiment with various methods to reduce time and development cost. This has lead to the development and evolution of programming with virtual tools, also called as virtual instrumentation.

The comfort level of a developer many vary from different perspectives such as the exposure to various programming methodologies, their proficiency in these languages and the confidence to implement it.

The purpose of this paper is to investigate the pros and cons of programming paradigms that are used for implementation in programming with virtual instruments and textual programming. Based on the analysis, this paper will be

capable to decide which programming approach has potential in minimizing the development cost and other parameters.

II. EVOLUTION OF TEXTUAL PROGRAMMING

At the dawn of modern computer age in the mid- 1950s, a small team at IBM decided to create a more practical alternative to programming the enormous IBM 704 mainframe (a supercomputer in its day) in low level assembly language the most modern language at the time. The result was FORTRAN a more human readable programming language.
FORTRAN, i.e. Formula Translation was intended for high level scientific, mathematical computations. BASIC was developed specifically for timesharing. It was a very stripped-down version of FORTRAN, to make it easier to program. The language C is still the most successful language and many other languages such as C++, Perl and Python are based on C. All of the features of Pascal, including the new ones such as the CASE statement are available in C. C uses pointers extensively and was built to be fast and powerful at the expense of being hard to read.
In the late 1970's and early 1980's, a new programing method was being developed. It was known as Object Oriented Programming, or OOP. Objects are pieces of data that can be packaged and manipulated by the programmer. Bjarne Stroustroup liked this method and developed extensions to C known as "C with Classes." This set of extensions developed into the full-featured language C**++,** which was released in 1983. C++ was designed to organize the raw power

of C using OOP, but maintain the speed of C and be able to run on many different types of computers. C++ is most often used in simulations, such as games.

After C++, Sun Microsystems developed java language. It provides platform independence. We can use the same code on Windows, Solaris, Linux, Macintosh, and so on. Compared with Java, C++ builds are slow and complicated. These languages are purely program based textual languages.

In the times when programming languages were just formed, the focus was on executing the program successfully and getting an accurate output. This, in itself was a difficult task. Thus, GUI and aesthetics were neglected. Many challenges were faced while using textual languages; these are explained in the next section.

III. CHALLENGES FACED IN TEXTUAL PROGRAMMING

One major challenge faced during textual programming is the time consumption. Here, the code is made from scratch and is done manually. All of the libraries have to be referenced manually and the logic must be worked out. Even after the program has be written, it has to be tested for errors, debugged and executed. Error debugging is another challenge faced by the developers. In this type of programming, the developer must study the entire code in detail so as to find the bug and the resolve it. If the code is long and complex, it may be frustrating for the developer to resolve errors.

The user interface has to be designed in a way that can be easily comprehended by the user; it has to be consistent and well organized. The aesthetics of the software play an important part in its usage. Though this is not impossible in textual programming, it leads to complicated coding that can be done by professionals and experts.

The software must also be self-explanatory in terms of logic and GUI. In case the user wishes to modify the software, it must be an achievable task. Again, depending on the complexity of the program, this task can be done by professionals alone. The time taken by an individual to study that particular programming language enough to modify the program, increases overall time consumption.

On an industrial level, scientists may not be well-equipped with programming knowledge, but still need to create user interfaces to control hardware operations. This generates an issue wherein professional programmers have to be roped in to do the work.

IV. PROGRAMMING WITH VIRTUAL TOOLS

This module can be classified as programming with visual tools and graphical programming. Here, we use various available tools to create and modify softwares as per user requirements.

A. Programming with visual tools:

Several Integrated Development Environments are available that offer visual tools for designing a software. One such example is Microsoft Visual Studio. It provides an integrated platform for creating web applications, web forms, databases etc. Visual Studio is well equipped with a toolbox that offers the developer readymade tools to design the software. This toolbox consists of tools like buttons, input textboxes, labels, dialog boxes, timers etc. One can also connect the application to database schemas.

Programming in Visual Studio is a form of visual programming [3], i.e. textual programming along with visual tools to design the GUI. However, Intellisense, another feature of visual studio, makes textual programming easier by automatically generating lists of keywords and functions from the inbuilt library to predict what the developer wants to write. This makes it handy for the developers as they do not have to remember the keywords or syntax, also reducing syntactical errors in process.

B. Graphical programming

Graphical programming is purely virtual programming as it involves generating a program using only virtual tools. These tools may be in the form of knobs, buttons, etc. This form of programming is mainly used to create softwares that in turn control hardware machines. A virtual instrument consists of an industry-standard computer or workstation equipped with powerful application software, cost-effective hardware such as plug-in boards, and driver software, which together perform the functions of traditional instruments. Traditional instruments come with limitations in form of actual knobs with pre-specified value range, buttons with specific commands etc. This limits flexibility in implementing operations where values are fluctuating. Also, these instruments cannot be modified, but have to be created again.

Using virtual instruments, one can create GUIs that depict the traditional instruments but can change the value ranges as per requirements. This proves beneficial for the developer as well as the user.

V. INTRODUCTION TO VIRTUAL INSTRUMENTATION

The rapid adoption of computerized applications in the last 20 years catalysed a revolution in the fields of instrumentation and automation. The concept of virtual instrumentation offers several benefits to engineers and scientists who require increased productivity, accuracy, and performance.

Virtual instruments represent a fundamental shift from traditional hardware-centred instrumentation systems to software-centred systems that exploit the computing power, productivity, display, and connectivity capabilities of popular desktop computers and workstations. It can be said that using virtual instrumentation, the traditional hardware applications can be digitalized, thus making software applications that control the hardware machines.

VI. EVOLUTION OF VIRTUAL INSTRUMENTATION

Historically, instrumentation systems originated in the distant past, with measuring rods, thermometers, and scales. In modern times, instrumentation systems have generally consisted of individual instruments. Even complex systems such as chemical process control applications typically employed, until the 1980s, sets of individual physical instruments wired to a central control panel that comprised an array of physical data display devices such as dials and counters, together with sets of switches, knobs and buttons for controlling the instruments. The introduction of computers into the field of instrumentation began as a way to couple an individual instrument, such as a pressure sensor, to a computer, and enable the display of measurement data on a virtual instrument panel, displayed in software on the computer. and enable the display of measurement data on a virtual instrument panel, displayed in software on the computer monitor and containing buttons or other means for controlling the operation of the sensor. Thus, such instrumentation software enabled the creation of a simulated physical instrument, having the capability to control physical sensing components.

VII. ARCHITECTURE OF VIRTUAL INSTRUMENT

A virtual instrument is composed of following blocks:

- Sensor Module
- Sensor Interface
- Information Systems Interface
- Processing Module
- Database Interface
- User Interface

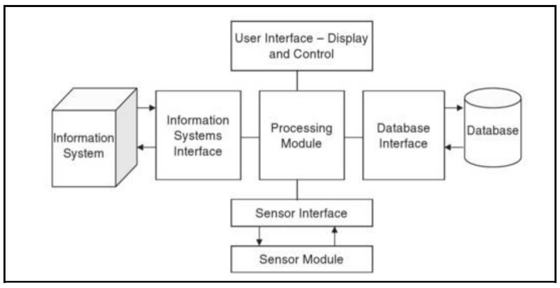


Fig. 1.1. Architecture of a Virtual Instrument

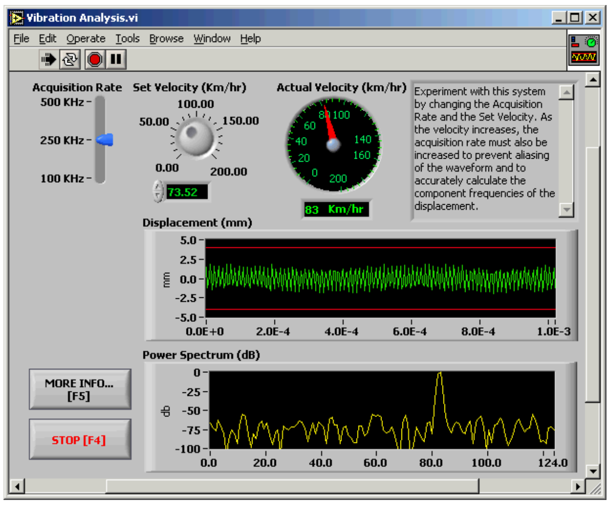
Fig.1.1 shows the general architecture of a virtual instrument. The sensor module detects a physical signal and transforms it into a digital form. Through a sensor interface, the sensor module communicates with a computer. Once the data are in a digital form on a computer, they can be manipulated or stored. Then the data is displayed or converted back to analog form.

VIII. G PROGRAMMING CASE STUDY: LABVIEW

LabVIEW[1] (Laboratory Virtual Instrument Engineering Workbench) is a programming environment that features a dataflow-based VPL (called G) which was designed to facilitate development of data acquisition, analysis, display and control applications. Moreover, one of LabVIEW's marketing claims is that LabVIEW is so usable that it is an effective tool not only for trained programmers, but also for certain types of end users. In particular, LabVIEW is described as usable by scientists and engineers who possess limited programming experience, yet who need software to interact with laboratory equipment.

You can customize front panels with knobs, buttons, dials, and graphs to emulate control panels of traditional instruments, create custom test panels, or visually represent the control and operation of processes. The similarity between standard flow charts and graphical programs shortens the learning curve associated with traditional, text-based languages.

Fig 1.2 shows the front panel of LabVIEW that displays the user interface component. Several knobs, buttons, graphs, level adjusting switches etc. can be seen in this panel. Fig 1.3. shows the Block panel of the front panel. Here the entire working of the application is revealed. Loops, variables, stop switches are all presented in form of blocks.



Experiment with this system by changing the Acquisition Rate and the Set Velocity. As the velocity increases, the acquisition rate must also be increased to prevent aliasing of the waveform and to accurately calculate the component frequencies of the displacement.

Fig 1.2: Front Panel of LabVIEW

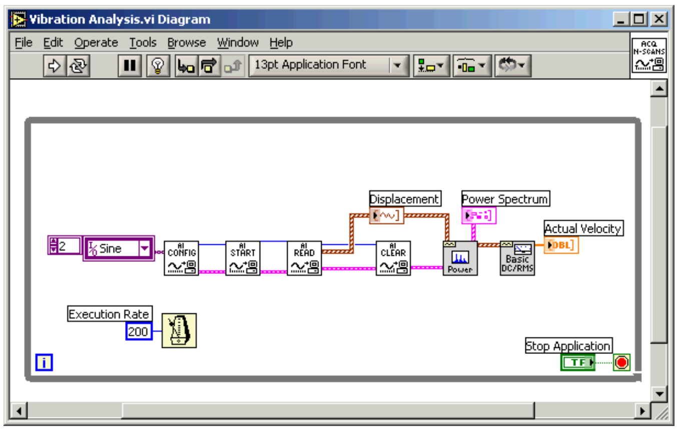


Fig 1.3: Block Panel of LabVIEW

CHALLENGES IN GRAPHICAL PROGRAMMING

Though this is a more advanced and convenient technique of programming, it has limited options when it comes to virtual tools. A common developer has to create programs using only the tools available in the virtual toolbox. It may be possible for an expert to create his own tools.

Moreover, looking isn't always seeing. Different individuals have different perspectives. A person not familiar with the software may interpret the symbols incorrectly. Conventions have to be followed in graphical programming and specific tools have specific representations. They must be used for that particular task [4].

Also, as compared to textual programming where the code goes on in a flow, graphical programming has many scattered components not necessarily organized properly. Thereby, it may prove problematic for someone other than the developer to get the gist of the program.

The comfort level of the developer also decides the difficulty in graphical programming. A professional from a computer field may be more comfortable writing long codes in textual languages, instead of using graphical tools.

IX. COMPARISON BETWEEN TEXTUAL PROGRAMMING AND PROGRAMMING WITH VIRTUAL TOOLS

1) *Integrated Development Environment*

The major thing with programming with virtual tools is that it requires an IDE (Integrated Development Environment). The entire implementation of the program revolves around this IDE. On the other hand, textual programming is independent of such development environment.

2) *Time*

The use of virtual tools minimizes amount of time since majority of work is handled by the IDE even we are not sure about the syntax of the language. In case of textual programming due to the absence of IDE, the developer is bound to have some skill of the language syntax master in language syntax in order to speed-up
the development process
.

3) *Debugging*

During debugging, a virtual tool points out errors and puts forth alternative functions or properties to resolve them unlike a non-virtual tool language.

4) *Interfacing tools*

A virtual tool provides an ease to develop a program just by introducing features such as drag and drop, adding menu lists, dialogue boxes, drawing elements graphically, etc. In case of a textual programming there is no way out than to write down the entire code manually which is a much tedious task.[3]

5) *Modifications*

A lay man/beginner can easily modify the interface with the help of virtual tool paradigms, whereas, modifying a language without a virtual tool becomes a complex job.

6) *Speed*

G represents an extremely high-level programming language whose purpose is to increase the productivity of its users while executing at nearly the same speeds as lower-level languages like FORTRAN, C, and C++.

XI. EXAMPLES OF TEXTUAL PROGRAMMING V/S PROGRAMMING WITH TOOLS

1) Example of C v/s LabVIEW

For Loop in LabVIEW

A For Loop executes a sub diagram a set number of times. Fig.1.4. below shows an empty For Loop in LabVIEW. A For loop executes its sub diagram n times, where *n* is the value wired to the count (N) terminal. The iteration (i) terminal provides the current loop iteration count, which ranges from 0 to *n*-1.

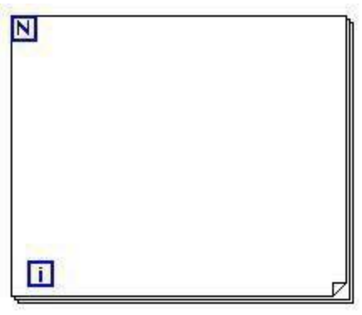


Fig. 1.4.: For Loop in LabVIEW

For loop in C: Fig. 1.5. shows a flowchart depicting a For Loop in C language.

- The **init** step is executed first, and only once.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.
- After the body of the 'for' loop executes, the flow of control jumps back up to the **increment** statement.

The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

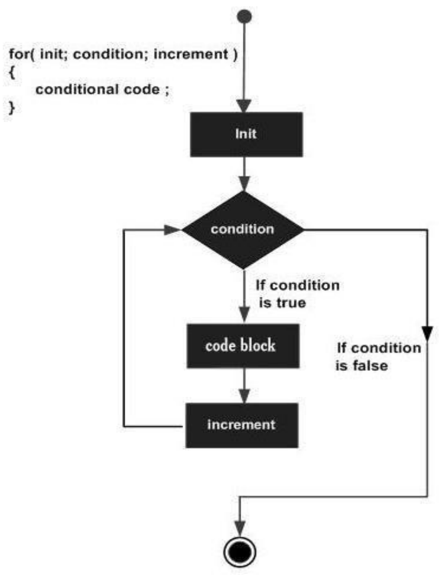


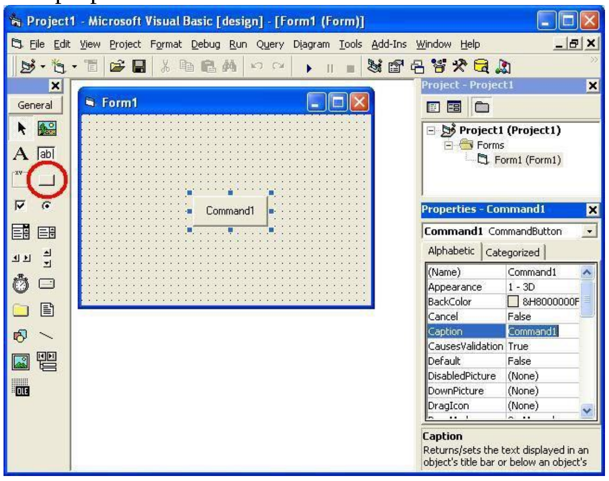
Fig. 1.5.: For Loop in C

2) Example of HTML v/s Visual Basic

Creating a command button in HTML:
<!DOCTYPE html>
<html>
<body>
<button type = “button” onclick= “alert(‘Hello world!’)”>Submit</button>
</body>
</html>

Creating a button in VB:

- Open your form. Figure out where you want the button to appear.
- Select the command button tool on the toolbox to your right.
- Draw the button on the form to the size that you want.
- Change button name and functions in properties.



(Name)	Command1
Appearance	1 - 3D
BackColor	&H8000000F
Cancel	False
Caption	Command1
CausesValidation	True
Default	False
DisabledPicture	(None)
DownPicture	(None)
DragIcon	(None)

Caption
Returns/sets the text displayed in an object's title bar or below an object's

Fig. 1.6.: Creating command button in VB

X. CONCLUSION

There has been much research in the area of visual programming tools, because humans think and remember things in terms of pictures. Imagery is an integral part of creative thought. Humans can absorb data much easier from well-defined plots than they can from large codes. Proponents of visual programming therefore argue that the development of visual tools is a natural step in the evolution of programming.
Textual programming languages have been considered as a universal standard for programming. While they can be used for writing arbitrary programs and provide some high level constructs, they are not very practical to use and often fail to simplify basic programming tasks with respect to their text based counterparts.
This paper presented the pros and cons of textual as well as graphical programming. We can conclude that virtual tools, well suited for manipulating high level abstractions should provide a lot of support for frequent tasks, and promote ease of navigation and consistency. Thus it can be said that despite the popularity and extensive use of textual programming languages, programming using virtual tools and instruments is emerging as a fresh alternative.
This being said, a third convention may emerge soon where a choice of textual or graphical programming could be provided in the same IDE, allowing developers to choose the option they feel suitable. Furthermore, this evolved IDE may also arrange for an alternative for converting textual code to graphics and vice versa.

REFERENCES

[1] http://www.ni.com/labview/

[2] Towards Virtual Laboratories: a Survey of LabVIEW-based Teaching/ Learning Tools and Future Trends*

[3] https://channel9.msdn.com/Forums/Coffeehouse/Visual-programming-language-vs-text-based-programming-language

[4] Visual Programming: The Outlook from Academia and Industry- K. N. Whitley and Alan F. Blackwell.