Malaysian Journal of Computer Science, Vol. 12 No. 1, June 1999, pp. 76-81

VISUAL LANGUAGES: A NEW WAY OF PROGRAMMING

Rodina Ahmad
Software Engineering Department
Faculty of Computer Science & Information Technology
University of Malaya
50603 Kuala Lumpur
email: rodina@fsktm.um.edu.my

ABSTRACT

Visual programming languages(VPLs) represent quite the biggest departure from traditional programming approaches. However the last ten years have seen quite remarkable progress in this field. There is a number of visual programming languages (VPLs) already in the market and others are still in the research prototype stage and it is difficult to predict the suitability of its usage for the real world applications. This paper highlights some of the basic concepts of visual programming, its classification, current research trends and the benefits gained from using it.

Keywords: ***Visual programming language, visual object-oriented language, form-based language***

1.0 HISTORICAL BACKGROUND OF PROGRAMMING LANGUAGES AND ENVIRONMENTS: TEXTUAL PROGRAMMING

The textual programming languages and environments have been developed ever since the computer was invented and therefore have been greatly influenced by the computer hardware organization. As a result, most of the textual languages developed are oriented towards using character-based input and output and the program structure is usually sequential. Beside hardware, programming languages are also influenced by natural language and mathematical formalisms such as algebra, predicate calculus and lambda calculus. This gives way to creation of variables to symbolize addresses, keywords such as if-then-else to describe conditional branches and unconditional branches. However, as computer progresses, it was found that some higher level languages do not favor natural language structure and choose mathematical formalisms like lambda calculus and predicate calculus to develop languages like LISP and PROLOG. These languages have been widely used especially in the area of artificial intelligence applications until today.

Textual programming languages which were developed as the result of the influences mentioned have several drawbacks. These languages provide facilities to express algorithms according to how computers operate but not according to how the human mind works. Since the medium of expression is text, which is one dimensional, algorithms are expected to be sequential. This requirement has restrained the programmer's thinking and forced him to consider organizing every program to be linear which may not be the requirements for its algorithm. In addition, textual programming usually has a complex syntax which is inherited from its natural language ancestry. Usually, the programmer is forced to follow strict rules to ensure that the program executes. This prohibits the programmer from being more creative and becomes more focus on the language syntax.

There are several other loopholes of textual programming such as their use of variables which have various conflicting roles, their weakness in expressing the concepts of object orientation and in describing data structures such as lists, graphs and trees which require many levels of abstraction from the intended semantics. However, text also has its worth which cannot be dismissed easily. Text is superior in expressing compact concepts such as algebraic formulae, comments and program element identification. As a result, it is always seen as part of the program.

2.0 THE ADVENT OF VISUAL PROGRAMMING

Visualization is long associated with computers and programming. Most of the analogue machines which were constructed prior to the development of digital computers were programmed in a pictorial fashion. Besides, flowcharting which uses picture like diagram, has been used as a heuristic aid in designing algorithms in programming our present digital computers. Visual programming field has matured from the unification of progress in computer graphics, programming language and human-computer interaction. One of its pioneer contributions came from the work of Ivan Sutherlands who designed Sketchpad on the TX-2 computer at MIT in 1963. His work has been recognized as the first computer graphics application which allowed users to work with a lightpen to create 2D graphics by creating simple primitives and applying operations on the geometric

shapes which the users created. Sketchpad has contributed mostly in its graphical interface and support for user-specifiable constraints.

The second major contribution in VPLs was made by David Canfield Smith in his Ph.D. dissertation entitled 'Pygmalion: A Creative Programming Environment'. Smith introduced the idea of icon-based programming paradigm which allows the user to create, modify and link together small pictorial objects (icons) with defined properties to perform computations. Pygmalion also introduced the concept of programming-by-example in which the user shows the system how to perform a task in a specific case and the system generates a program to perform generalized tasks. Smith's work has initiated many other visual programming researches till today.

3.0 VISUAL PROGRAMMING AND ITS DEFINITION

Visual programming differs from textual programming in terms of expressing the syntax and semantics of programming languages. Visual programming uses visual syntax which means that some terminals are able to display pictures or forms or other illustrations. A visual syntax may incorporate spatial information and visual attributes such as color, depth and location. According to Shu [25], a visual programming language uses "some visual representations (in addition to or in place of words and numbers) to accomplish what would otherwise have to be written in a traditional one-dimensional programming language." Shu also emphasizes that "to be considered a visual programming language, the language itself must employ some meaningful… visual expressions as a means of programming".

In our current scenarios, a VPL is usually embedded inside a visual environment. The environment is said to be a visual environment when the tools are graphical or adopted graphical techniques for manipulating pictorial elements and displaying program structure. This is where the programmer works to create, modify and design his programs. The environment may also consist of a set of tools and a user interface for accessing the tools.

The objective of having visual programming and visual environments is to enable programmers to express their logic or ideas of solving a problem in a simpler way, and to enable them to clearly understand how the program works visually. In order to achieve this objective, VPL usually uses the following four characteristics.

1. Conceptual simplicity: VPL represents the underlying concepts as natural as possible and simplifies abstract concepts. It only emphasizes logic which is directly pertinent to the application and not the programming mechanics such as event loops, storage allocation and scope rule for objects.

2. Concreteness: VPL uses concreteness to facilitate creation of a program. Concreteness can be used to provide feedback or to provide specification to the program that needs to be created, for example, using the point and click user interface technique to enable programmers to specify what they want for their program.

3. Explicitness: VPL usually shows relationships among objects or modules in a very explicit way such as through connections or diagrams.

4. Immediate visual feedback: Liveness level refers to the degree in which VPL may provide immediate feedback. VPL is fully live as it displays the effects of changes made by the programmer automatically, that is, the programmer does not need to press a special button or do something in order to see the effects.

4.0 CLASSIFICATION OF VISUAL PROGRAMMING LANGUAGES (VPLS)

As the field of VPL grows, many people have tried to categorize VPLs. Basically, they can be categorized as follows:

A. Purely visual language
B. Hybrid text and visual systems
C. Others, such as constraint-oriented systems and form-based systems

Purely visual language semantics have been derived entirely or predominantly from graphical rules. These languages are usually characterized by heavy reliance on visual techniques throughout the programming process. The program created in purely visual language is compiled directly from its visual representation and not translated into any text-based language. Examples of these languages include Pictorial Janus, VIPR, Prograph and PICT. There are other suggestion to subdivide this category into more specific language paradigm such as object-oriented, functional, imperative and logic. Burnett and Baker [5] have developed a classification scheme for classifying VPL research papers. Their aim is to help other researchers to easily locate relevant works in this field.

Another subset of VPLs favors the idea of integration of visual languages with well-established textual programming languages. They perceive the integration might be more likely to meet the actual requirements of practical software development than the highly ambitious goal of creating purely visual languages. These hybrid systems may include both programs created visually and

then translated into high-level textual language, or programs which involve the use of graphical elements in a textual language. The work of Andrew, Erwig and Meyer [13] are examples of programs created graphically and the system generates textual program from it. The latter includes works to extend existing textual language such as C++ and Basic. The current commercial system in this category includes Visual Basic and Visual C++.

Beside the two major categories discussed, there exists many VPLs which fall into smaller classifications similar to Pygmalion or Sketchpad. They are termed constraint-based languages which are especially popular for simulation design. An example of this is ThingLab II [16], which is an object-oriented and constrainted programming system. A few other VPLs follow the metaphor of spreadsheets and these languages are classified as form-based VPLs. Programming using form-based VPLs involves altering a group of interconnected cells and allowing the programmer to visualize the execution of a program as a sequence of different cell states which progress through time. Examples of form-based VPLs are Forms/3, ASP (Analytic Spreadsheet Package), NoPumpG, NoPumpII and Penguims. Besides that, there are VPLs such as Vampire [17], ChemTrains and BITPICT which combine visual object-oriented programming language with a rule-base approach. Table 1 summarizes classification of most of the visual system or VPLs frequently quoted according to the three categories highlighted above.

The Prograph environment has three main components which are the editor, interpreter and application builder. The editor is used for program design and construction, the interpreter executes a program and provides debugging tools and the application builder helps the task of constructing a graphical user interface for a program.

Another example of VPL is VIPR or Visual Imperative Programming. VIPR is developed by Citrin et al [8] at the University of Colorado in 1993. It is a general-purpose visual programming languages. It uses nested series of concentric rings to visualize or represent a program. Each computation step is represented by the merging of two such nested rings in the presence of a state object which is connected to the outermost ring. Fig. 2 illustrates three sequential statements in action. Fig. 3 illustrates an example of if-then-else statement in VIPR. In this case, if two or more rings are nested inside a ring with an attached state object, the conditions of all these rings are evaluated and one of the rings will be evaluated.

Table 1: Classification of Visual Systems or Visual Programming Languages

Purely Visual Languages Systems	**Hybrid text & Visual Programming Systems**	**Others**
VIPR, Prograph, Pictorial Janus, LabVIEW, Visual Engineering Environment(VEE), HI-VISUAL, Vista	Visual Basic, VisualWorks, Visual J++, Visual C++, OpenStep, ObjectWorld, Rehearsal World	Pygmalion, ThingLab, Forms/3, Pursuit, Vampire, ChemTrains, BITPICT, ASP, NoPumpF, NoPumpII, Penguims.

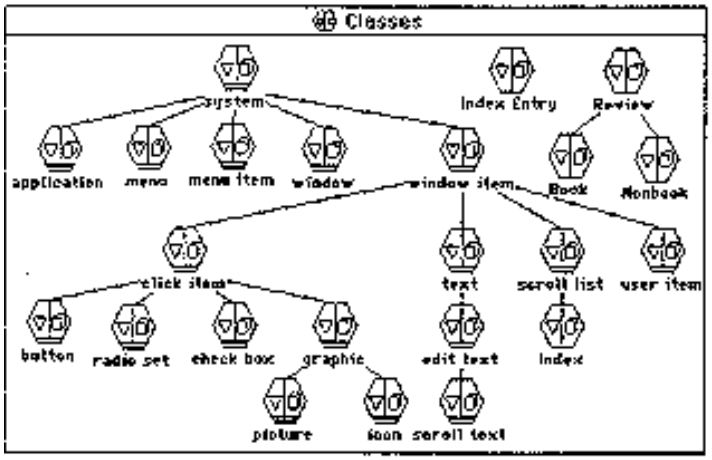

Fig. 1: Example of Class Window in Prograph

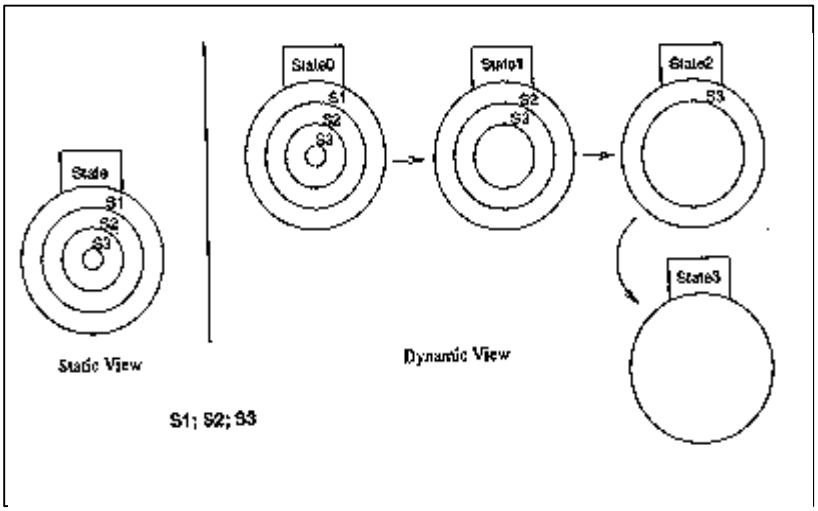

Fig. 2: Sequential statements S1,S2 and S3 are represented statically and dynamically in VIPR

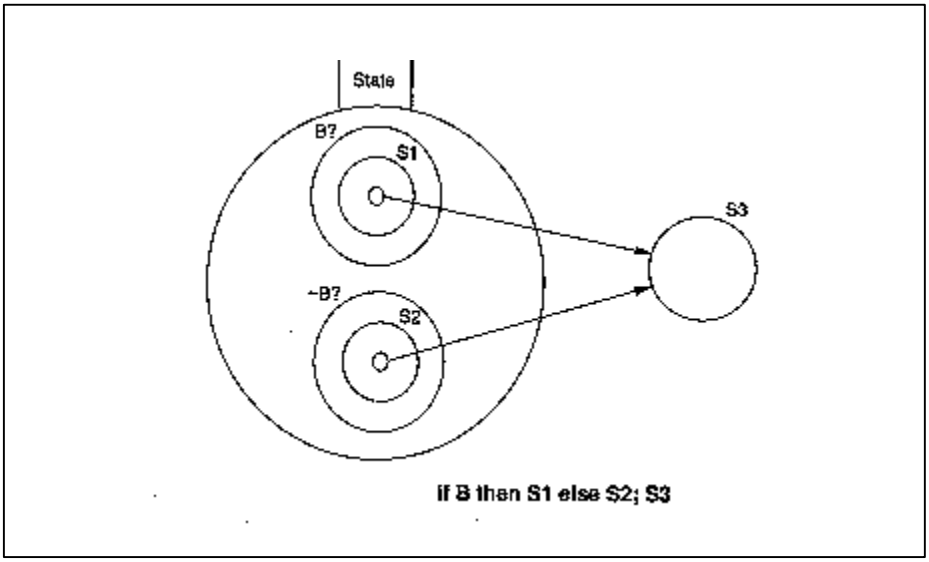

Fig. 3: An if statement in static VIPR representation

5.0 THE BENEFITS OF VISUAL PROGRAMMING

Visual programming addresses two important issues in software engineering. Firstly, it enables users to master the complexity of programming by visualizing it. The raising of the abstraction to the visual level reveals semantic relationships among program entities and makes it more understandable. Another one is increasing productivity which is gained either through the ease of using the systems or the increase in communication among the developers and the users of the systems, because the users can usually understand the beginning process of designing the systems.

Some studies have been carried out to compare visual programming with other types of programming. Green[14] compared the readability of textual and graphical programming and concluded that graphical programs took longer to understand than textual ones. Moher [19]also compared petri-net representations with textual program representation and found areas where the petri-net representation was more suitable. Besides, Pandey and Burnett [20] compared time, ease, and errors in constructing code using visual and textual languages. They found that matrix and vector manipulation programs constructed using visual programming had fewer errors. Another study by Cunniff and Taylor on the comprehension of a static visual flowchart language versus the textual language Pascal, was conducted on 23 novice programmers. The finding is based on reaction time and on the number of correct responses from the programmers. The study reported that the flowchart language was easier to comprehend.

In the real world application, Baroth and Hartsough [2] reported on the use of LabView in their workplace. They had two group of developers to create a telemetry analyzer. One group was asked to use LabView and another group was asked to use C programming. After about eight weeks of work, they found that the visual programming group had exceeded the original requirements while the C programming group had not completed the original requirements. They cited several advantages of visual programming such as its flexibility in the design process, improvement of communication between users and programmers, and shorter period to train programmer to master the language. Besides, they have observed increased productivity through a reduction in software development time as communication between the customer, the developer and the computer is facilitated by the visual programming tools used.

Other attractive features of visual programming tools are that they are easy to use and understand and make rapid prototyping possible. As a result, early testing and demonstration, and allows corrections to be made early in the development life cycle. The prototype can later be refined to produce the final application. Visual programming also makes coding easier to follow and the codes become more reusable, easier to debug and easier to document. Maintenance is also easier because the visual programs are usually run-time reconfigurable. As a result, the pace of work is faster since changes made visually are immediately in effect and testable without the recompilation stage. On the whole, it shortens up the development process. Another benefit cited is that visual programs are much easier to adapt to run on parallel processing systems. This is due to the fact that a visual program preserves information about the "dependencies" in an algorithm and this information is needed by the computer in order to run in parallel. A textual program usually lacks this explicit dependency information and the compiler usually must first try to infer from the existing structure before attempting any parallel execution.

6.0 SUMMARY

This paper looks at the definition and classification of visual programming. VPLs are characterized by their immediate visual feedback, concreteness, explicitness and conceptual simplicity to enable programmers to better understand programs. Visual programming has simplified the programming process and made programming projects more manageable. It also enables the users to be involved more directly in the process of programming. Future research should focus more on its use in the development of applications.

REFERENCES

[1] J. W. Atwood, Jr., M. M. Burnett, R. A. Walpole, E. M. Wilcox and S. Yang, "Steering Programs Via Time Travel" in *IEEE Symposium on Visual Languages*, Boulder, Colorado, Sept. 1996.

[2] E. Baroth and C. Hartsough, "Visual Programming in the Real World", in M. M. Burnett, A. Goldberg, and T. Lewis, eds., *Visual Object-Oriented Programming: Concepts and Environments*, Englewood Cliffs, Prentice-Hall, 1994.

[3] T. Budd, *An Introduction to Object-Oriented Programming*, Reading, Addison-Wesley, 1990.

[4] M. M. Burnett, "Seven Programming Language Issues" in M. M. Burnett, A. Goldberg, and T. Lewis, eds., *Visual Object-Oriented Programming: Concepts and Environments*, Englewood Cliffs, Prentice-Hall, 1994.

[5] M. M. Burnett and M. J. Baker, "A Classification System For Visual Programming Languages", Technical Report 93-60-14, Department of Computer Science Oregon State University, Corvallis, OR.

[6] S. K. Chang, "Elements of a Visual Language", http://www.cs.pitt.edu/~chang/365/elements.html.

[7] S. K. Chang," Principles of Visual Programming Systems", http://www.cs.pitt.edu/~chang/365/sk1. html.

[8] W. Citrin, M. Doherty and B. Zorn, "The Design of a Completely Visual OOP Language", in M. M. Burnett, A. Goldberg, and T. Lewis, eds., *Visual Object-Oriented Programming: Concepts and Environments*, Englewood Cliffs, Prentice-Hall, 1994.

[9] P. T. Cox, F.R. Giles, and T. Pietrzykowski, "Prograph: A step towards liberating programming from textual conditioning", in *IEEE Workshop on Visual Languages, Rome, October 4-6, 1989*, pp. 150-156.

[10] P. T. Cox, H. Glaser and S. Maclean, "A Visual Development for Parallel Applications", *Proceedings of IEEE Symposium on Visual Languages, Halifax, 1998*.

[11] R. W. Djang and M. M. Burnett, "Similarity Inheritance: A New Model of Inheritance for Spreadsheet VPLs", *IEEE Proceedings of Visual Language '98, Nova Scotia, Canada, September 1998.*

[12] R. Dye, "Visual Object-Oriented Programming", *Dr. Dobb's Macintosh Journal*, Fall 1989.

[13] M. Erwig, and B. Meyer, "Heterogeneous Visual Languages-Integrating Visual and Textual Programming" in *11th IEEE Symp. on Visual Languages, Darmstadt, 1995*, pp. 318-325.

[14] T. R. G. Green, M. Petre, and R. K. E. Bellamy, "Comprehensibility of Visual and Textual Programs: A Test of Superlativism Against the Match-Mismatch Conjecture", in *Fourth Workshop on Empirical Studies of Programmers, New Brunswick, December 1991*, pp. 121-146.

[15] J. Grundy, J. Hosking, S. Fenwick, and W. Mugridge, "Connecting the Pieces" in M. M. Burnett, A. Goldberg, and T. Lewis, eds., *Visual Object-Oriented Programming: Concepts and Environments*, Englewood Cliffs, Prentice-Hall, 1994.

[16] J. H. Maloney, A. Borning and B. N. Freeman-Benson, "Constraint Technology for User-Interface Construction in ThingLab II", in *Proceedings OOPSLA '89*, ACM SIGPLAN Notices, Vol. 24, No. 10, 1989, pp. 381-388.

[17] D. W. Mcintyre, "Design and Implementation with Vampire", in M. M. Burnett, A. Goldberg, and T. Lewis, eds., *Visual Object-Oriented Programming: Concepts and Environments*, Englewood Cliffs, Prentice-Hall, 1994.

[18] F. Modugno, "Interface Issues in Visual Shell Programming", in , M. M. Burnett, A. Goldberg, and T. Lewis, eds., *Visual Object-Oriented Programming: Concepts and Environments*Prentice Hall, Englewood Cliffs, 1994.

[19] T. G. Moher, D. C. Mak, B. Blumenthal, and L. M. Leventahal, "Comparing the Comprehensibility of Textual and Graphical Programs: The Case of Petri Nets", in *Fifth Workshop on Empirical Studies of Programmers, Palo Alto, December,* 1993.

[20] R. K. Pandey, M. M. Burnett, "Is it Easier to Write Matrix Manipulation Programs Visually or Textually? An Empirical Study", in *IEEE Symposium on Visual Languages, Bergen, Norway, Aug. 1993,* pp. 344-351.

[21] M. Ratcliff, C. Wang, R. J. Gautier, and B. R. Whittel "Dora- a Structure Oriented Environment Generator", *Software Engineering Journal*, Vol.7, No. 3, 1992, pp. 184-190.

[22] S. P. Reiss, "Interacting with the FIELD Environment", *Software-Practice and Experience*, Vol. 20, No. S1, 1990, pp. S1/89-S1/115.

[23] S. Schiffer and J. H. Frohlich, "Visual Programming and Software Engineering with Vista", in M. M. Burnett, A. Goldberg, and T. Lewis, eds., *Visual Object-Oriented Programming: Concepts and Environments*, Englewood Cliffs, Prentice Hall, 1994.

[24] A. Scrivener, "The Impact of Visual Programming in Medical Research", in *Conference of Medicine Meets Virtual Reality II-Interactive Technology and Healthcare: Visionary Applications for Simulation, Visualization and Robotics, January 27-30, 1994.*

[25] R. Sebesta (1996) *Concepts of Programming Languages.* Menlo Park, California, Addison-Wesley.

[26] N. C. Shu, *Visual Programming*, New York, N. Y. Van Nostrand Reinhold, 1988.

BIOGRAPHY

Rodina Ahmad obtained her Master of Computer Science from RPI (USA) in 1991. Currently, she is a lecturer in the Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya. Her research interests include software measurement, object-oriented programming and computer aided instructions.