ISSN:2320-0790

An International Journal of Advanced Computer Technology

A Survey on Graphical Programming Systems

Gurudatt Kulkarni[1], Sathyaraj. R[2]

[1,2]School of Computing Science and Engineering
VIT University
Vellore, Tamil Nadu

Abstract: Recently there has been an increasing interest in the use of graphics to help programming and understanding of computer systems. The Graphical Programming and Program Simulations are exciting areas of active computer science research that show the signs for improving the programming process. An array of different design methodologies have arisen from research efforts and many graphical programming systems have been developed to address both general programming tasks and specific application areas such as physical simulation and user interface design. This paper presents a survey of the field of graphical programming languages starting with a historical overview of some of pioneering efforts in the field. In addition this paper also presents different classifications of graphical programming languages.

Keywords: Graphical programming, Visual programming, Data-flow programming, End-user programming

I. INTRODUCTION

In today's industrial systems software have become complex and development require high qualities in terms of reusability and maintainability. The process of software development is a tedious process which requires highly specialized skills in system programming. Conventional programming languages are difficult to learn and use, requiring skills that many people cannot acquire. It has a steep learning curve. Alternately, there are many advantages in providing programming features in the graphical form. As the adoption of computing systems such as computers, tablets, smartphones grows, the major population of users do not know how to program. For the user to learn to program the systems, graphical approach is the most suitable method to make them familiar with programming. Some Graphical programming systems have successfully presented that non-programmers can create fairly difficult programs with little training. There are other classes of systems known as Program Simulation systems which are usually used for debugging or for training purposes.

II. RELATED WORK

The field of graphical programming has grown from the combination of work in programming, human-computer interaction and computer graphics. Sketchpad [2] was the first major work developed by Ivan Sutherland which allowed users to work with a lightpen to create two dimensional graphics by creating primitives and then using features such as copy and constraints on the geometry on the shapes. Its graphical layout and support for user-specifiable properties are the main features which made sketchpad stand out. It had lot of contribution to graphical programming languages. Behave [3] is another graphical programming environment for specifying behavior. It was developed by Michel Travers at MIT Media Labs. The main aim of this language is to describe the idea that a program or a rule is a complex object created from basic objects according to a specific grammar. This was developed to control the actions of a robot fish in a virtual fish tank. It takes the concepts from programming language theory and converts them into graphical representation. For example, the types are represented using colours .LabVIEW [1] stands for (Laboratory Virtual Instrument Engineering Workbench). It was originally released in 1986, by National Instruments. It uses a graphical programming language known as G. It is a dataflow programming language. Execution of a program is determined based on the structure of a graphical block diagram which is connected by wires. Wires are used for flow of variables and any block can execute as soon as input data is available. LabVIEW has an inbuilt compiler that produces native code for the CPU platform. The graphical diagram is converted into executable machine code by parsing the syntax and by compilation. William Sutherland developed a graphical programming language on TX-2 to develop simple graphical data flow programming language. The system made possible users to create, debug, and execute dataflow programs in a unified graphical environment. The next major work in graphical

programming language was done by David Canfield Smith in his dissertation named "Pygmalion: A Creative Programming Environment" [4]. This was a pioneering work which marked the beginning of number of branches of research in this field which exist to this day. For instance, it consisted of icon based programming architecture in which a user created, modified, and connected together small graphical objects, called icons, with fixed properties to perform computations. Google developed App Inventor which used visual programming for developing Android Apps easily it is now maintained by MIT [25]. "Blockly" is another pioneering work done by Google. It contains a visual programming editor. It uses a drag and drop model to build an application. It has many languages and platform integrated with it such as Google App engine. It also has many features like unit testing which was previously unheard of in graphical programming environment [22].

III. GRAPHICAL PROGRAMMING LANGUAGES

As the field of graphical programming languages has evolved, more and more interest has been focused on creating a standardized, robust classification for work in the area. Such a classification helps in finding similar work but also provides a foundation to compare and evaluate different systems. The following are different types of graphical programming languages that are present today.

•Programming-by-example systems
•Constraint-oriented systems.
•Hybrid text and graphical systems.
•Purely graphical languages.
•Form-based systems.

The above mentioned classification is not mutually exclusive. Many languages can be placed in more than one category.

The purely visual languages can be called as one of the most important category. Languages that fall into this category are heavily dependent on graphical techniques throughout the programming process. In these systems almost all operations are performed by manipulating graphical objects on a workspace. Further, the program is debugged and executed in the same environment. The program is directly compiled into low level representation directly without any intermediate stages. One of the examples of such a system is LabVIEW [1].

One important sub category of graphical programming system attempts to combine graphical and textual objects. In this type of systems programs are created graphically and then converted into a high-level textual representation of graphical elements. One of the examples of such system is Pentagruel presented by [5] which explain the system of home automation. In this system, authors present a two-step process; one is the description of the functionalities and the properties of the environment entities. Second is the development of an application for home automation. It is driven by taxonomy of objects and consists of adjusting those using high-level constructs. To help these two steps, the language consists of a textual layer and graphical layer. The above two form two major classification of graphical programming language. In addition to these two there are other small classifications. There are many graphical programming languages which fall into programming by example category such as [6] [7]. In this category system allows the user to create and modify graphical objects with the motive of recording the actions so that the system can perform those actions automatically in the future. The other sub classification is a constraint-oriented system; in this type of system graphical programs are constrained to real world scenarios. [8][9][10] are example of such systems in which real world constraint is modelled in the programs. The application of constraint-oriented systems is also used in development of graphical user interfaces. This is clearly visible in the works of [11]. There are some other classes of graphical programming languages which use graphical and programming components from spreadsheets. These languages are known as form based languages. They depict programming as modifying a group of internetworked cells over time and often allow the programmer to view the execution of a program as a series of different cell states which progress through time. The form based systems are presented in [11] [12]. These systems used form based interface to create and manipulate objects in the system.

IV. CONCEPTUAL VIEW OF GRAPHICAL PROGRAMMING LANGUAGE

This part of the paper presents the advances in literature of graphical programming languages. These advances are mostly taken from the works by [13]. There are some of standard definitions in literature which are as follows.

A. Icon

A block with a dual structure consisting of physical part and logical part.

B. Iconic system

A systematic set of related icons.

C. Iconic Sentence

A graphical arrangement of icons from iconic system.

D. Visual language

A collection of iconic sentences created with given syntax and semantics.

E. Syntactic Analysis

Analysis of an iconic sentence to find the core structure.

F. Semantic analysis

Analysis of an iconic sentence to find the intrinsic meaning.

V. PRESCRIBED SPECIFICATION OF GRAPHICAL PROGRAMMING LANGUAGES

A graphical arrangement of icons that are part of a graphical sentence is a two-dimensional equivalent of a one dimensional arrangement of tokens in textual programming languages. In textual languages a program is written as a string in which individual tokens are concatenated to form a sentence whose arrangement and meaning are discovered by syntactic and semantic analysis, respectively. In opposition to textual languages, graphical languages are differentiated into three creation rules that are used to arrange icons: horizontal concatenation, vertical concatenation and spatial overlay.

In formalizing graphical programming languages, it is usual to differentiate object icons from process icons. The later express computations; the latter can be further categorized into simple object icons and composite object icons. The simple object icons depict primitive objects in the language, whereas the composite object icons depict complex arrangements of simple object icons.

A graphical programming language is defined by a triple (I, G, B), where I is the icon set, G is a grammar and B is a domain-specific knowledge. The icon set is a set of generalized icons each of which is described as a pair (Xm, Xi), where Xm is a logical part and Xi is a graphical part. The grammar G specifies how complex programs may be created from simple objects by placing them logically on workspace. The domain specific information needed for creating a meaning for a given program is represented as B. It contains information related to the icon set, relation between those icons, logical meaning of icons and real world meaning of icon set.

VI. ANALYSIS OF GRAPHICAL PROGRAMMING LANGUAGES

As stated in above section a graphical program is created form simple icons using various operations. The syntactic analysis of graphical program is also known as parsing is based on many different approaches. This paper presents a few of such approaches.

A. S-System Petri Net Generator

Petri net is a mathematical model in which there are nodes and places. The nodes are represented as bars and places are represented by circles. There are directed arcs which describe pre and post conditions for different places. In this system petri nets have been used for the functionalities in parsing graphical program. It does syntax checking for programs. Petri nets are also responsible for converting graphical program to textual form. It is also responsible for assembly code generation [15].

B. Graph Grammar

Diagrammatic manipulations of multi-dimensional data in a graphical program can be represented using graph grammar. It is used for syntactic pattern recognition. Grammatical modifications in graphical language can be conveniently processed using this technique. In this technique an algorithm iteratively finds common sub structures from the given graph and converts it into a production rule [16] [17].

C. Operator Precedence grammars

This grammar can be used for mathematical expression analysis. These types of grammars are useful for analyzing graphical programs containing graphical operators and graphical blocks. A tree is created based on comparison of precedence of operators in a pattern and differentiating patterns into one or more subcategories.

D. Context free grammars and Context dependent grammars

A context free grammar is a set of rules used to generate string patterns. It contains a set of terminal symbols, nonterminal symbols and a set of production rules for replacing non terminal symbols. In graphical programming, graphical sentences are parsed using this technique. In this blocks are considered as terminal symbols and composite diagram created with these simple blocks are considered as sentence.

VII. PROBLEMS IN GRAPHICAL PROGRAMMING LANGUAGES

This section describes some of the prominent issues faced by graphical programming languages.

A. Domain Specific Language

Graphical programming languages are predominately domain specific. This makes it restricted to a particular domain. To create a graphical language, domain knowledge must be thoroughly researched. As the domain evolves the language must also be simultaneously updated. Many general-purpose graphical languages have been researched but have not been as efficient as domain specific languages [20] [21].

B. Graphical Representation problems

In graphical languages programming constructs are represented as simple blocks. The representations must be simple enough that it can be easily understood by a novice programmer. Lot of research has gone into icon representation and interaction design. A major concern is to develop a system that is usable; this generally means designing systems that are easy to learn, providing an immersive user interface and effective to use [22].

C. Code Reusability

This is one of the major challenges faced by graphical programming languages. Since graphical languages are domain specific code cannot be reused in other modules. Most commonly performed functionalities are already designed in the system. Since, this is an end-user programing system, users are generally not aware of concepts of software engineering.

D. Program Abstraction

Graphical programming languages are not truly considered as complete programming languages since they provide a high-level abstraction of a program. These languages contain an underlying textual representation that is actually considered as a program. Graphical languages have a textual definition for all the components of the language. Whenever, a graphical program is created, a textual representation is generated. It is this textual representation that is further used for future program execution stages. The graphical representation adds an extra overhead which needs to be processed. If a language is graphics intensive then this will need systems with high configuration.

VIII. CONCLUSION

The field of graphical programming language has many examples which have proved to be very useful and helpful for people with no programming experience to learn and continue programming. Even though many programming systems discussed above defer in details and domain of operation they all share a common goal of easing the programming process. Recent developments in this field have reaffirmed its position and made has made the platform mature. Even though there has been lot of research in past thirty years many systems such as Scratch and Pygmalion are still popular. The survey shows that even though graphical programming systems are easy to understand and learn textual programming must not be avoided. There are cases where textual programming can perform the task easily and in efficient way in such cases graphical programming must be avoided. With the improvement in hardware and 3D technology graphical programming languages can take advantages of these technologies and create even more immersive programming experience.

IX. REFERENCES

[1] "LabVIEW", *http://www.ni.com/labview*, January 23, 2014.

[2] Ivan Edward Sutherland, "Sketchpad: A man-machine graphical communication system", *University of Cambridge*, September 2003.

[3] "Behave", *http://xenia.media.mit.edu/~mt/behave/behave.html*, January 23, 2014.

[4] David Canfield Smith, "A Creative Programming Environment", MIT Media Labs, 1975.

[5] Zoe Drey,Charles Consel, "Taxonomy-driven prototyping of home automation applications: A novice-programmer visual language and its evaluation", *Journal of Visual Languages and Computing, Elsevier*, August 2012.

[6] Mehdi Manshadi, Daniel Gildea, James Allen, "Integrating Programming by Example and Natural Language Programming", Department of Computer Science, University of Rochester, Rochester, NY 14627.

[7] J W Carlson, " A Visual Language for Data Mapping", Workshop on Domain-Specific Visual Languages, Tampa Bay, FL, U.S Department of Energy, August 2001.

[8] Eugenio J. Marchiori, Angel del Blanco, Javier Torrente, Ivan Martinez-Ortiz, Baltasar Fernandez Manjon, "A visual language for the creation of narrative educational games", *Journal of Visual Languages and Computing, Elsevier*, September 2011.

[9] Gary Rommel, "Work in Progress - Using A Graphical Programming Language Teach to Microprocessor Interfacing", ASEE/IEEE Frontiers in Education Conference,October 2005.

[10] Ye Weijun, Ying Shi, Zhao Kai, Ni Youcong, "Design and Implementation of Semantic Programming Language Graphical Edit Tool", *International Conference On Computer Design And Appliations, IEEE*, 2010.

[11] Yubin Liu, Li Wu2 and Xinfa Dong, "Research on Controls-Based Visual Programming", *Second International Conference on MultiMedia and Information Technology, IEEE, 2010*.

[12] Gilbert Tekli, Richard Chbeir, Jacques Fayolle, "A visual programming language for XML manipulation", *Journal of Visual Languages and Computing, Elsevier*, February 2013.

[13] Chang, S.-K., *"Principles of Visual Programming Systems"*, Prentice Hall, New York.

[14] Teboul, O., Kokkinos, I., Simon, L., Koutsourakis, P., & Paragios, N. (2011, June). Shape grammar parsing via reinforcement learning. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on* (pp. 2273-2280). IEEE.

[15] Ng, K. Mun, and Z. Alam Haron. "Visual microcontroller programming using extended S-system Petri nets." WSEAS Transactions on Computers 9.6 (2010): 573-582.

[16] Fahmy, Hoda, and Dorothea Blostein. "A survey of graph grammars: Theory and applications." Pattern Recognition, 1992. Vol. II. Conference B: Pattern Recognition Methodology and Systems,

Proceedings. 11th IAPR International Conference on. IEEE, 1992.

[17] Boshernitsan, Marat, and Michael Sean Downes. Visual programming languages: A survey. Computer Science Division, University of California, 2004.

[18] Zhao, Chunying, Jun Kong, and Kang Zhang. "Program behavior discovery and verification: A graph grammar approach." Software Engineering, IEEE Transactions on 36.3 (2010): 431-448.

[19] Dobesova, Zdena. "Visual programming language in geographic information systems." Proceedings of the 2nd international conference on applied informatics and computing theory. World Scientific and Engineering Academy and Society (WSEAS), 2011.

[20] Kosar, Tomaž, et al. "Comparing general-purpose and domain-specific languages: An empirical study." Computer Science and Information Systems 7.2 (2010): 247-264.

[21] Kosar, Tomaž, Marjan Mernik, and Jeffrey C. Carver. "Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments." Empirical software engineering 17.3 (2012): 276-304.

[22] Rogers, Yvonne, Helen Sharp, and Jenny Preece. Interaction design: beyond human-computer interaction. John Wiley & Sons, 2011.

[23] Ashrov, Adiel, et al. "A use-case for behavioral programming: architecture in JavaScript and Blockly for interactive applications with cross-cutting scenarios." Science of Computer Programming (2014).

[24] Fowler, Allan, and Brian Cusack. "Kodu game lab: improving the motivation for learning programming concepts." Proceedings of the 6th International Conference on Foundations of Digital Games. ACM, 2011.

[25] "App Inventor", *http://appinventor.mit.edu/explore/*, March 29, 2014