The analysis of visual parallel programming languages

Vladimir Averbukh[1] and Mikhail Bakhterev[2]

[1] Krasovskii Institute of Mathematics and Mechanics, Ural Branch of Russian Academy of Sciences
Yekaterinburg, Sverdloskaya oblast 620990, Russia
averbukh.vl@gmail.com

[2] Krasovskii Institute of Mathematics and Mechanics, Ural Branch of Russian Academy of Sciences
Yekaterinburg, Sverdloskaya oblast 620990, Russia
mike.bakhterev@gmail.com

Abstract

The paper is devoted to the analysis of state of the art in visual parallel programming languages. The brief history of this domain is described. The diagrammatic imagery of visual languages is analyzed. Limitations of the diagrammatic approach are revealed. The additional type of visual parallel programming languages (action language) is described. Some problems of perception of visualization for parallel computing are considered. Some approaches to the evaluation of visual programming languages are suggested.

Keywords: **Visual parallel programming languages; diagrammatic languages; perception of visualization.**

1. Introduction

In the late 70's - early 80's of the last century, Visual Programming was formed as a new independent domain. The researchers and practitioners put high hopes on Visual Programming. They believed that visual way to describe programs might simplify the process of programming. It was noted that pictures may map objects of the real world, whereas text representation may only refer to program objects. In addition, multidimensionality of graphics may increase informativeness in comparison with a one-dimensional text flow by using, for example, shapes, sizes, colors, textures, directions or distances. Therefore, Visual Programming should be more accessible to thinking of novice programmers. Visible and compact techniques of programming had to reduce the abstraction level of algorithm presentations. The usage of visual metaphors basing on natural figurativeness was assumed. For more serious cases of programming such graphic representations as finite automata, data-flow graphs, state transition diagrams, Petri Nets, etc. should be used. However nowadays disappointments take place. The authors of the well known among programmers book "Design Concepts in Programming Languages" say:

'It is also possible to represent programs more pictorially, and visual programming languages are an active area of research. But textual representations enjoy certain advantages over visual ones: they tend to be more compact than visual representations; the technology for processing them and communicating them is well established; and, most important, they can effectively make use of our familiarity with natural language' [1].

Visual programming for parallel computing gives hopes to be more useful and effective. And there are many examples of interesting solutions in this domain. But visual parallel programming languages remain more research projects then real tools. Below we'll discuss our approach to the situation with visual parallel programming languages. Our analysis should help reveal perspectives of the future development of this domain.

2. Visual Programming

The first realizations of Visual Programming environments have been based on the so-called *executable graphics*. In this case algorithms may be described in the visual form and visual programs should execute on computers directly without translation into common (text) programming languages. Visual languages should include graphic representations for all program elements to describe both the control and data structures. It seemed that through visual techniques to design programs (for example *Nassi-Shneiderman diagrams*) programmers may refuse from the stage of coding. (For example there is the idea – "Algorithms without programmers".) It was assumed that the difficulties of novice programmers were successfully handled by means of animation and they can correlate the static text of the program and the process which was generated by this text. Diagrammatic languages were the basis of many visual programming environments developed in the 80's and 90's.

Diagrammatic languages are characterized by well-defined, formalized dictionaries consisting of a relatively small number of elements. Also, as a rule, the spatial syntax is strictly defined. That is the spatial layout of diagram elements and their positions relative to each other are precisely specified. The automatic placements of diagram elements were realized in some programming environments. Work with the environments basing on diagrammatic languages, as a rule, implements according to the common plan. Users employ a system of menu consisting of diagram elements and graphic templates. Thus the diagram step by step is depicted on a screen. Then filling in the appropriate text boxes of graphical templates are occurred. Really programming systems basing on diagrammatic languages use hybrid - text and graphic techniques. There are many examples of using as the main metaphors flow charts and Nassi-Shneiderman diagrams. (These diagrams use the presentation of control flow.) Also there are many examples of visual programming environments basing on presentation of data flows. In this case simple structures of graphs facilitate modular design and allow applying diagrams at all levels of program descriptions. There is no need to use special language features to link individual modules. Examples of visual language based on finite automata, Petri nets, HIPO-charts met less frequently. In the 90-ies, at the next stage of development of visual languages, the systems on the basis of UML diagrams have appeared. Let's note some limitations of a diagrammatic approach to visual programming.

Means of graphical representation of data structures usually are absent in diagrammatic systems based on the concept of control flow. Everything related to the data must be described in plain text. Many systems also require at some stage to detail parts of the program. Therefore it is necessary to do inserts on plain-text of programming languages. That is why all similar systems demand extensive volume of text input. The use of data flow graphs entails similar problems, such as: the lack of high-level control structures, leading to increased complexity and entanglement of diagrams, the only way to represent the semantic information exists. This is using of the names of nodes and arcs. Also usually the scope of the systems based on other diagram metaphors, is strongly limited. To resolve the problems arising from the use of data flow and control flow diagrams, mixed diagrammatic metaphors were used. These metaphors are combined, such as data flow diagrams and Petri Nets. But only a few examples of these decisions took place.

Systems on the basis of iconic languages often use expanded models of data flows. In this case, in nodes of the graphs pictures are placed. The picture as a rule represents previously developed program functions for data processing. There are interesting examples of iconic languages on the basis of natural imagery, quite accurately depict the meaning of a function. Experience of iconic programming languages played a role in graphic (more precisely iconic) interfaces. However, iconic languages did not become a frequent practice in modern programming.

Note, that there are examples of 3D visual languages using abstract imagery but implicitly one may relegate this type of visual languages to a class of iconic languages.

Data flow graphs, as well as control flow graphs, enough easy to animate. However, there are only a few systems with animated executable graphics. There are some examples of realization ideas of executable graphics in visual variants of “normal” programming languages. These examples include functional programming languages. (For functional languages it is very difficult to find adequate techniques of visualization of the main concepts.)

So, in spite of all their limitations, diagrammatic languages are the most popular type of visual programming languages. Visual languages built into some mathematical packages are just diagrammatic (dataflow) languages. Microsoft Visual Programming Language (VPL) is relatively recent example of classic diagrammatic language on the basis of data flows. One may remember that visual meanings based on diagrammatic imagery play an important role in such programming environments as Visual BASIC, Delphi, etc. However, in this case, visual programming means not quite (or not the same) that originally was meant in the 80th years. Now this is not a system executable graphics that seemed as a goal of visual programming environments in 80-ies. In these well-known environments of visual programming a key role plays not a depiction of program control flows or data flows but the depiction of interactive behavior of an application program.

3. Visual Parallel Programming Languages

The first systems based on visual parallel programming languages have appeared almost at once after sequential analogues earlier 80-ies. Depictions of parallelism were very limited in the beginning. Those parts of program graphs (for example data flow graphs), that, according to a programmer, may be parallelized, were marked in some way (usually double or heavy line) [2].

Thus the early environments of this type had no explicit means for a visual support of entities of parallel programming associated with message passing or process creation and management. At the following stage the depictions of parallel program structures were included to visual languages. In some visual languages the simple set of icons were used to represent constructions of parallel programming languages. (For example VISO – visual

realization of parallel programming language Occam [3].)

The important direction in visual parallel languages is the diagrammatic languages which are based on a message-passing paradigm. Such languages were actively developed in the 90th years and were similar to traditional visual languages. Sequential sections of programs were depicted by traditional diagrammatic techniques for control flows. There is the almost full realization of message-passing means in visual language Grapnel (including dynamic process creation and destruction) [4].

Also the tendency to design so-called "concept-based" visual parallel languages was revealed. In these cases researchers and developers suggest the main concepts to describe parallelism and conformably parallel programming.

A typical example of such (early) visual languages is CODE [2]. Visual programs had used data flow graphs contained nodes-processes and arcs to connect ports of these processes. Programmers after depicting of the general scheme have to describe processes and their input/ output ports in text form and to define conditions of node executions.

The interesting example of an early concept-based language is Phred [5]. Phred is a visual parallel programming language in which programs can be statically analyzed for deterministic behavior. The developers of Phred consider that nondeterministic computations are a significant problem in parallel programming. Phred addresses the issue of determinacy by visually indicating regions of a program where nondeterminacy may exist. A Phred program is composed of a control flow graph, a data flow graph, and a set of node interpretations. A Phred support environment allows a software designer to create Phred programs, to statically analyze them for determinacy, and to interpretively execute them.

The interesting concepts of parallelism descriptions are the basis of visual programming language Visper [6]. One of them - *Process communication graph (PCG)* – is used also in some related visual systems. The Process communication graph combines control flow graphs and data flow graphs to the united visual formalism based on the concepts of space-time diagram and concurrency map. These concepts earlier were used as debugging and efficiency tuning tools.

A visual, object-oriented parallel programming language Vorlon [7] realizes the parallel object-flow execution model. This model draws upon both object-oriented and dataflow models. As such it is able to manage both parallelism-oriented aspects, like synchronization with dataflow-like constructs, and problem domain complexity through types and type interrelations.

The next visual language for parallel, object-oriented programming is HiPPO (High Performance Parallel Object-oriented) [8]. In HiPPO the data flow model is changed and based on the flow of object references.

There is one again visual parallel programming system – VisualGOP [9]. This system is realized basing on *graph-oriented programming model* which aims at providing high-level abstractions for configuring and programming cooperative parallel processes. With GOP, the programmer can configure the logical structure of a parallel/distributed program by constructing a logical graph to represent the communication and synchronization between the local programs in a distributed processing environment.

Note that new visualization techniques are developed for the new concepts of parallel programming. The task was set to develop all-in one visual programming systems. These systems have to provide all development cycle and include visual means of parallel programming proper, debugging and performance tuning and debugging realized in frameworks of a common mental model. One can even say that forming of this mental model of functioning of parallel programs is the main task of design of these visual parallel programming environments. Quite another question is the task solved and even is this task at hand at modern state of the art. In itself techniques of program development and depiction in these environments are similar, despite of using of various approaches to the parallelism description. A programmer develops the general scheme of a parallel (or distributed) program, and then concretizes it by depiction of concrete details. Majority of systems have diagrammatic imaginary. The set of graphic elements is traditionally limited and, as a whole, is simplified, even if any icons are used. Animation for description of processes dynamics is not used.

We have analyzed a set of visual parallel programming languages. (See also the Taxonomy for visual parallel programming languages in [10].) As a rule, their visual dictionaries base on different types of charts and/or diagrams. These dictionaries are characterized above all a limited set of visual elements. Their semantics are set by strict senses and syntaxes are described by precise rules of element placement on the screen.

Note that visual languages using control flows in general have no any sense, additional to traditional (text) programming languages. In diagrammatic systems based on the concept of control flow, usually there are no graphic means to represent data structures on programs. So these structures often must be described in plain-text form. All these systems need extensive volume of text input. Use the data flow graphs involves a similar problem - the lack of high-level control structures, leading to diagram complexity and complication, the lack of means to depict non-trivial data structures, etc. [11], [12]. As already noted there is the almost only way to describe the semantic information – to use names of nodes and arcs. Systems based on other diagram metaphors have very limited

scope.

Unfortunately, some of the ideas that are implemented in visual programming languages reflect the previous level of development. Limitations of the "diagrammatic" dictionary prevent from solving problems arising in connection with the development of modern programming languages.

Let's consider the example. Means of describing the procedures for access to the data elements are well developed in dynamic languages and compiled languages of last generation. In particular these languages make possible using in operator expressions accesses to dynamic and associative arrays; to lists; to elements of row partition by regular expressions, and in some cases even to entries of data base tables. In most diagrammatic language mechanisms of data addressing don't exist in program structures. Instead of these mechanisms the abstraction of *arrow* (-> <-) is introduced. *Arrows* connect the outputs of one of the operators (or other software design) to the inputs of another. However the Arrow isn't a metaphor of access to elements of data structures because it does not involve the formation of values outside of the connection between the two operators. That is one of the main data properties violates – to exist when there are no operations on them. Thus, in the majority of visual languages the programmer must deal with, though with basic but implicit access to the data values (not even to the own data). In these languages well nigh there are no advanced facilities of describing references to computing results. A little example of such reference is a node in an orgraph representing an operator. A node may be depicted in the form of 3D or (more often) 2D object, for example, a polygon with a line coming out of it, symbolizing a result. And other arcs included to other nodes-operators are connected to this arc. This approach allows independently defining operands for operators which usually are N-ary in these systems. However local changes in the program can demand global editing of communications in its visual representation. For example, when the value received in one group of operators is necessary to process jointly with the value from another, spatially remote. In that case in a language with explicit access to the data (especially the imperative one) one may do a few local patches a program text: to keep the required values in the data structure of the program, adding several expressions into the group of operators separated by the code, and then turn to them, when they need a co-processing. Note that the other situation takes place in data flow iconic languages. In these languages firstly one has to select the group of objects (operands) and then the operations on it. That is, the language is intended on the explicit indication of data over which it is necessary to execute an operation.

Of course, many factors affect the popularity and breadth of using of one or another programming tools. But the presence in the programming language sophisticated facilities for connecting between an operator part of an expression and data does this language more useful. These tools may play an important role in simplifying of parallel programming.

In the following (from the second half of the 90s) an effort was made to break "diagrammatic" deadlock by means of entering dynamics into the process of programming. There are projects of visual parallel programming systems where there are attempts to visualize as well as parallelism, and dynamics of processes. Conceptually in these systems one may directly depict the modeling objects as it is usual for the application. It provides a direct mapping of visual specifications into the program. Visual images should represent higher-level mathematical objects. (See, for example, VIM Language [13] or to some extent "CYBER-FILM" [14].)

Visual interface that specifies initial values for the applied computing system also may be considered as a specialized visual language for parallel programming. For example, the project ASSY [15] aims to develop a metasystem that supports the development of problem-oriented programming systems. In this system the means to solve the problem of the interaction of the flow of rarefied plasma are realized. In such environments, nothing but the main entities are visualized. In case of ASSY, these entities are high-level concepts of a certain computational method.

Also there are serious problems of visual representation, human perception and interpretation in connection with visualization of parallel programs. These problems are common for visual programming, visual debugging and performance tuning. Visualization of real parallel programs and data leads to cumbersome and often not interpretable pictures.

No matter how big the screen is, but the volume of visual data required to represent a serious parallel and/or distributed programs will exceed its capabilities. Practice shows that even small complication of the program structure leads to maze patterns, similar to puzzles on complexity of interpretation. A possible decision of the arising problems is connected with using 3D graphics and particularly virtual reality and augmented reality environments. Just these means should provide the most effective use of tridimentionality and dynamics. Nevertheless the serious problem there is an adequate interpretation of extremely large volumes of visual data with very complex structure.

4. Conclusion

Experience in the development and use of visual programming languages shows that their successes are associated with specialization. (See for example well-

known LabView [16].) Universal visual programming languages could not overcome the level of academics studies. The same situation was found in visual parallel programming languages. The interest in visual programming parallel language wanes in gradual mode. New developments (as of 2010-th) are appeared more and more seldom, although there is no question of the complete cessation of activity in this field of research. Let us try to understand the reasons for such situation.

From the very beginning of the development it is considered that the main goal of Visual Programming is to reduce mental efforts of programmers [17]. But is drawing of detailed program diagrams easier than a detailed textual programming? Note that detailed program depicting may be considered even as a sophisticated pictographic script.

Visual languages and visualization in general, are used to depict objects and their attributes. The basic communicative potentialities of concrete visualization are demonstrated on representation of qualitative and quantitative properties of objects, depiction of relationships between objects and processes associated with these objects [18]. Therefore, as a goal of visual programming one may consider the possibility of presentation of data structures and data elements, supporting of representation of program dynamics and the possibility of program generalization.

One may consider another goal of visualization in parallel programming – to support analysis and exact interpretation of programs during the process of their development. Then the evaluation of visualization should be linked to the possibility of interpreting images, and interpretability will be an important measure of the quality of visual languages.

Visualization either maps pre-existing mental models of users (programmers in this case) or forms them again. (Sometimes a combination of both processes takes place almost simultaneously - that is on the basis of pre-existing mental picture a new one is built.) This yields one again quality criterion of visualization in visual languages - the correspondence of visual languages and the existing programmer mental models. On our opinion attempts to create the new models of parallelism made in so-called conceptual languages, are not mapped logics of development of programming languages. On our opinion at this stage the auxiliary means of a parallelizing support may be more useful.

The very interesting and productive idea, to depict all aspects of message-passing interaction, realized in Grapnel [4] have no the further development. As it seems there is no large need in depiction of parallel programs. Probably this is due to the fact that in many case the effective parallelization is reached by means of modern compilers or other tools for automatic parallelization. And more complex problems are solved through new languages, for which the visual representation of abstract concepts not found yet.

Thus, the development of a visual programming for parallel computing has faced a number of challenges related to both fundamental issues of visual description of modern programming languages entities, and the perception of large amounts of visual information. The visual languages for parallel programming have not become the real tools for professional programming. One of the reason is connects with the limitations of diagrammatic techniques. We will continue our research and development. The next step will be a visual language to support the new paradigm of parallel programming. Problem solving should be sought through searching of fundamentally new methods of parallel programming, including the ability to use metaphors and visualization design that may support adequate mental models.

Acknowledgments

This work was supported the Program “Algorithms and Software for Supercomputer Systems” of the Presidium of Russian Academy of Sciences (project No.12-P-1-1034 of UB RAS).

References

[1] F. Turbak, D. Gifford (with Sheldon M.A.) “Design Concepts in Programming Languages”. Cambridge (Massachusetts). The MIT Press, 2008.

[2] P. A Newton “Graphical Retargetable Parallel Programming Environment and Its Efficient Implementation”. Technical Report TR93-28, Dept. of Computer Sciences, Univ. of Texas at Austin, 1993.

[3] M.S. Al-Mulhem “Concurrent programming in VISO” Concurrency: Pract. Exper. 2000; 12. Pp. 281-288.

[4] P. Kacsuk, G. Dozsa, T. Fadgyas “Designing parallel programs by the graphical language GRAPNEL”. Microprocess . And Microprogramm. 1996, V. 41, 8-9. Pp. 625 - 643.

[5] A.L. Beguelin, G.J. Nutt “Visual parallel programming and determinacy: A language specification, an analysis technique, and a programming tool”. Journal of Parallel and Distributed Computing, 22(2), August 1994. Pp. 235-250.

[6] N. Stankovic, Kang Zhang “A distributed parallel programming framework”. IEEE Transactions on Software Engineering. Vol. 28. No 5. May 2002. Pp. 478-493.

[7] J. Webber, P.A. Lee “Visual, Object-Oriented Development of Parallel Applications”. Journal of Visual Languages & Computing Vol. 12, Issue 2, Pp. 145-161.

[8] P.A. Lee, C. Phillips, P. Watson “Final Report: High Performance (Parallel) Object-Oriented Software Systems (HiPPO)” http://www.parallelism.cs.ncl.ac.uk/projects/hippo/FinalReport.pdf

[9] Fan Chan, Jiannong Cao, Alvin T. S. Chan1 and Kang Zhang “Visual programming support for graph-oriented parallel/distributed processing”. Softw. Pract. Exper. 2005;

35. Pp. 1409–1439.
[10] P. A. Lee, J. Webber “Taxonomy for Visual Parallel Programming Languages”. Technical report series. University of Newcastle upon Tyne, Computing Science, 2003. http://www.cs.ncl.ac.uk/publications/trs/papers/793.pdf
[11] Philip Cox, Simon Gauvin, Andrew Rau-Chaplin “Adding Parallelism to Visual Data Flow Programs”. SoftVis '05 Proceedings of the 2005 ACM symposium on Software visualization. Pp. 135-144.
[12] Philip Cox, Simon Gauvin “Dataflow Visual Programming Languages”. VINCI '11 Proceedings of the 2011 Visual Information Communication - International Symposium. Article No. 9.
[13] N. Mirenkov “VIM Language Paradigm”. Proceeding of CONPAR-94 - VAPP VI International Conference on Parallel and Vector Processing. J. Kepler University of Linz. Austria. September 6-8 1994. (Lecture Notes in Computer Science) Springer-Verlag. Berlin. 1994. Pp. 569-580.
[14] R. R. Roxas, N. Mirenkov “Cyber-Film”: A Visual Approach That Facilitates Program Comprehension”. Int. J. Soft. Eng. Knowl. Eng. 2005. V. 15. N6. Pp. 941-973.
[15] V.A. Vshivkov, M.A. Kraeva, V.E. Malyshkin “Parallel Implementation of the Particlein-Cell Method”. Programming and Computer Software, 1997, V. 23, N.2. Pp. 87-97.
[16] http://www.ni.com/labview/
[17] S-.K. Chang “Visual Languages: A Tutorial and Survey”. Visualization in Programming. (Lecture Notes in Computer Science 282). Berlin. Springer-Verlag. 1987. Pp. 1-23.
[18] William J. Bowman “Graphic Communication”. New York. Wiley, 1968.

Vladimir L. Averbukh is the head of the Computer Visualization researcher's section in Krasovskii Institute of Mathematics and Mechanics, Ural Branch of the Russian Academy of Sciences. He’s also the associate professor in Ural Federal University. His primary research area is computer visualization, focusing on theoretical problems of interactive visualization. Averbukh has a PhD in computer science.

Mikhail O. Bakhterev is the researcher of System Programming Department in Krasovskii Institute of Mathematics and Mechanics, Ural Branch of the Russian Academy of Sciences. He’s also the assistant professor in Ural Federal University. His primary research area is Software Engineering for parallel and distributed computer systems.