Movie-based Control Flow Objects and Operations

Alexander Vazhenin[†],

University of Aizu, Graduate School Department, Aizu-Wakamatsu, Japan

Summary
This paper deals with visual programming in the framework of a Movie-based Multimedia Environment, which includes special movie-program objects. These objects can generate frames of an algorithmic movie as well as executable code. The important part of the movie-based programming is in defining activity areas in matrix structures as well as ordering operations according to computational steps. Specific components called control lines and structures support referencing activity areas inside a structure. This means that control objects divide a structure into zones each of which can have individual colour. Paper presents multimedia environment for specifying dynamical behaviour of control lines for matrix structures. We show semantic and syntactical rules of iconic programming language and main features of a GUI-interface.

Key words:
Software Engineering, Visual Programming, Movie-based Programming, Matrix Computing.

1. Introduction

Recently, various programming methods have been developed along with developments in computer architecture. The human-centric design is very often related to a "Program Visualization (PV)" [1] or "Visual Programming Language (VPL)" [2], under which scientists usually understand any programming language that lets users ***specify programs by manipulating program elements graphically rather than by specifying them textually***. In contrast to traditional textual programming languages, where multi-dimensional structures are encoded into one-dimensional strings according to some intricate syntax [3], VPL remove this layer of abstraction, allowing the programmer directly observe and manipulate complex software structures via some space-time metaphors [4].

Tanimoto [5] states that "Data Factory" indicates visual dataflow environment. In this model, users can control icons prepared with mathematical operations in the layout where mathematical methods are connected to others like belt conveyers in the factory. JAVAVIS was developed as a tool to support teaching object-oriented programming concepts with Java [6]. This tool monitors a running Java program and visualizes its behaviour with two types of UML diagrams, which are de-facto standards for describing the dynamic aspects of a program, namely object and sequence diagrams. We can characterize most of the mentioned systems as very special. They are mostly focused on solving specific problems.

The Redwood [7] is a development environment that supports visual representation of program structure and drag-and-drop manipulations of programming constructs called snippets defined as templates in which a solution to a smaller part of a problem is embedded. The XML specification of a snippet consists of the display section (or component visualization section) and the templates section (or code mapping section). After snippet selection, the user should customize its parameters. Unfortunately, the system has no GUI-tools for designing and debugging the user-defined snippets. Moreover, it is unclear how to control the adequacy between both snippet sections. That is why a large collection is needed of pre-built snippets in the environments, in addition to the online snippet library.

The GASPARD (Graphical Array Specification for PARallel and Distributed computing) is a visual programming environment devoted to the development of parallel applications [8]. Task and data parallelism paradigm of parallel computing are mixed in GASPARD to achieve a simple programming interface based on the printed circuit metaphor. The programmer specifies tasks and instantiates by plugging them into a slot (task parallelism). The Elementary Transformation library includes usual signal processing operations like FFT, integration, sum, dot products, etc. Each of these operations is supported by corresponding template that can take input arrays and return output arrays. The user has possibility to parameterize these templates, for example, by the size of arrays. However, it is unclear how to optimize the quality and performance of obtained executable code, which depended on quality of the each hand-made template developed. This means that the user has a small possibility to influence on the control flow sequence.

A multimedia approach for interactive specifications of applied algorithms and data representations is based upon a collection of computational schemes represented in the "film" format. It promotes high-level language constructs is based not only on the introduction of special symbols and images with semantic support, but also on a series of images that can present dynamical features of algorithms. Approach presented in [9,10], supposed to have an enormous number of pre-created algorithmic films with hard-coded templates for each particular domain-specific

Manuscript received September 5, 2012
Manuscript revised September 20, 2012

problem in a library. So, the user with basic programming knowledge can easily find, setup and use them. However, to develop new films which have no pre-created templates in a library becomes a challenging task. So far, the template development should be performed by means of developers at an advanced programming level.

In our approach [11-13], the end users also don't have to create the animation and necessary template programs from scratch. But, to increase the library development efficiency and maintain reusability of the template code, we have proposed a new approach where rather complex template programs can be synthesized automatically by using some set of elementary templates. So, instead of the extraction of the whole film together with hard-coded templates, the user is able to extract parts from a film accompanied by elementary templates, and assemble them into a new complete film. To implement this approach, a special concept of metaframes, domains and sub-domains has been introduced. As a result, the template development is become easier and may be performed even by users of a basic programming level with less specific knowledge of conventional programming languages. The movie-based programming process is manipulated with special movie-program objects (MP-objects) that generate automatically a part of an executable code as well as producing movie frames, which are adequate for the code generated.

An important aspect of a programming language is a control flow problem that refers to the order in which the individual statements, instructions or function calls are executed or evaluated [14]. According to the movie-based programming, it is in defining activity areas in matrix structures as well as an ordering of operations according to computational steps. Specific components called control lines and structures support referencing activity areas inside a structure. This means that control objects divide a structure into zones each of which can have individual colour. Paper presents multimedia environment for specifying dynamical behaviours of control lines for matrix structures. We show semantic and syntactical rules of iconic programming language and main features of a GUI-interface.

The rest of the paper is as follows. In section 2, principles of the movie-based programming are described focusing on dualistic features of metaframes. Section 3 presents semantic and syntactical rules of iconic programming language including examples of typical and complex control lines movement. Features of a GUI allowing graphical specification/editing/debugging of Control Flow Formulas are shown in Section 4. The last section contains the conclusion and future research topics.

2. Principles of Movie-based Programming

2.1 Common Remarks and Definitions

The movie-based programming is a creation of computational process that is depicted as a movie by correlating animation frames with solution steps. A frame is an image representing dynamical features of an algorithm at a particular time step. So, sequences of frames are observable as an animation. It can be composed and visually debugged. As shown in Figure 1, a computational step is visualized as a combination of visual symbols within a frame. A set of frames joined into an animated sequence (i.e. a visualization of a computation on a structure according to the traversal schemes and the computational formulas for the frame) is called an ***algorithmic film*** or just a ***film***. It is a combination of

- *S,* a collection of spatial structures;
- *D,* a set of variable declarations;
- *M,* a collection of ***metaframes***.

A ***metaframe*** is a special object representing a set of rules and parameters which are meant to specify how frames should be produced (visualized) in a film, and, how they should be implemented in an executable program. As shown in Figure 1, the two main types of metaframes, ***single*** and ***episode***, can be used to specify single or multiple algorithmic steps respectively. In other words, a metaframe is a combination of:

- *T*, a set of ***traversal schemes*** for node activation on structures in *S*;
- *F,* a set of ***computational formulas*** or ***C-formulas*** to be performed on nodes affected by schemes in *T*;
- *C,* a set of **control-flow formulas** exploited in the frame-generation process.

2.2 Structures and Control Lines

A structure is a medium representing an algorithm as computational process, and the scheme is a computational plan, which describes this process as a flow of activities on the structure. The activities are represented by a so-called “flashing” of nodes on which the attached operations are distinguished by a type of flashing. There can be a number of flashing types: full, contour, colour, shape, etc. In the presented work, we distinguish these operations by so-called flashing colours of nodes; different colours are representing different operations. Figure 2 shows a film on the matrix structure and some matrix algorithmic scheme

where the computational "wave" is moving upside down. This film consists of an episode metaframe, which produces three animation frames. Figure 2 also shows that the number of active nodes and placement of activities areas are changing from step-to-step that is reflected in a corresponding frame.

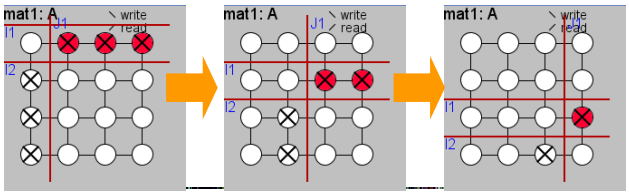


Fig. 2 Animation frames on an episode.

Rather often, the data structures in applications can be regular ordered sets of elements presented as 1D, 2D or 3D structures where structure nodes contain operable objects. So, sets of elements involved in computations can be related to the sub-domain nodes of these structures. The data structures of matrix and vector types have the following attributes (Figure 3):

- **Name** to identify structure (*mat3* in Figure 3),
- **Dimension** to distinguish matrices and vectors,
- **Parameters** to define structure sizes (*N, M* for rows and columns respectively),
- A set of **structure variables** (*A* in Figure 3) declared to store instances of data in nodes,
- A set of **control lines** which are used to refer to spatial placements and domains (for matrices, there are vertical, horizontal, major and minor diagonal lines).

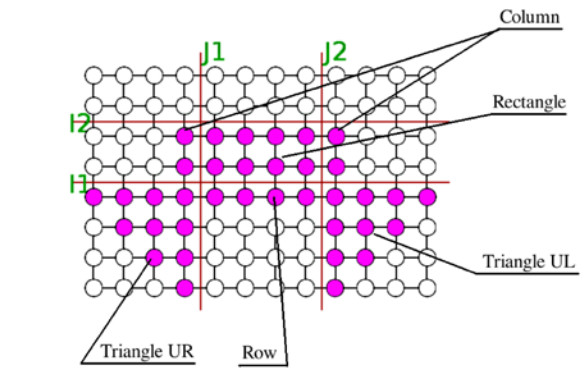


Fig. 3 Structure Attributes and Shape-based Domain Decomposition.

2.3 Shape-based Specification of Computations

To specify a computational activity on a structure from *S,* it is necessary to implement some appropriate traversal schema or a combination of schemes from *T*. In movie-based programming the schema is represented by a ***domain*** (Δ). Domain is a pattern describing a set of nodes to specify particular bounded areas of activity in a structure. Domain is aligned to control lines and depends on their placement, so it changes its configuration from frame to frame according to the control lines updating during frames transitions. CF-Formulas are used to specify a transitional behavior of control lines. Each domain corresponds to some partial scanning loop in executable code and is composed by sub-domains. Formally, sub-domain is a set of nodes which coordinates are satisfied to the system of constraints *Ω*. For matrix structures, general forms of *Ω* are defined in (1), where *(i,j)* - coordinates of nodes, *H1,H2,V1,V2* - positions of appropriate control lines or structure bounds (they are used to specify a sub-domain activity area) and predicates *Pk(i,j)* - special conditions (they are used to specify the sub-domain shape).

$$\Omega = \{(i,j) | H_1 \le i < H_2 \wedge V_1 \le j < V_2 \wedge P_1(i,j) \wedge P_2(i,j) \wedge \ldots \wedge P_n(i,j)\} \quad (1)$$

In general, domain $\Delta \in T$ is defined as a composition of sub-domains $\Omega i,\ i = 1,2,\ldots,n$.

(2)

Domain has a colour attribute to be visually distinguishable from other domains. Also, constraints *Hi,Vi* (1) are visually represented by vertical or horizontal lines which may lay over boundary of a structure. This approach allows the user to operate with parametric relation-based specification of computations rather than index-based to make it more compact and perceptive.

Domains can be both regular and non-regular ones. We consider a regular domain as a domain, which can be visited by partial scanning of nested loops. An irregular domain is a domain, which contains nodes that are impossible or difficult to visit by a partially ordered scanning of nested loops. Usually, the global irregularities of computations appear rarely because of possible performance degradation. However, the problem can be very important if there is necessity to place and store complex data in a computer memory. Our approach is based on decomposing such irregular areas to a number of regular sub-domains having ***typical shapes*** (row, column, rectangle, triangle, etc.). An example of complex domain

decomposed to typical shape-based sub-domains is also shown on Figure 3.

In order to implement a computational schema on a visually presented domain, it is necessary to have a visitor program, which can visit all nodes within this domain. In fact, it is necessary to have a special code generator, which can implement the visitor program, which depends on the domain shape. In fact, to produce a visitor program for a particular domain shape we use ***templates*** [11]. Template is a special data object related to the domain implementing an activation traversal scheme on a domain within the frame. Any template contains a high-level description used to visually reflect the scheme behaviour and a snippet of source code in a conventional programming language used to generate some fragment of executable code. Templates are playing the key role in the film development process since they are used as "***bricks"*** to construct an algorithmic movie.

3. Control Flow Semantics and Icon Language

3.1 MP-metaframe Semantic Rules and Micro Icons

An order of operations is defined by MP-metaframes generating a code according to computational steps each of which corresponds to an animation frame. As was pointed above, we distinguish two types of metaframes: a ***single metaframe*** and ***episode metaframe or MP-episode.*** A single metaframe represents the one algorithmic step and generates one corresponding animation frame. An ***MP-episode*** produces a series of frames. Importantly, the same operations should be implemented on all its internal frames. The Control Flow Formulas or ***CF-formulas*** are introduced to coordinate operations between frames as well as program the control lines behaviour. By using three semantic rules for CF-formulas showed in Fig. 3, the user can specify control lines movement as well as define a corresponding number of MP-frames or program iterations that will be generated.

In particular, CF-formulas define the control lines expression defining how to transfer data from the current metaframe to the next metaframe. A control line can be depended from other lines. To make editing process more convenient, we developed the icon language to specify CF-formulas. These icons can be used as macros in order assign behaviour of control lines for typical and complex dependences. The Figure 4 depicts an icon structure for vertical control lines.

CF_ID – **Control Line Name**

- **Initialization Rule (Start position):**
  CF_ID = <expression>;
- **Transition Rule (Next position):**
  if (<condition>)
  then CF_ID = <expression 1>;
  else CF_ID = <expression 2>;
- **Episode Rule (How to finish):**
  if (<condition>)
  then {Generate Next Frame};
  else {Finish Episode};

Fig. 3 Semantic Rules of a Control Line Movement.

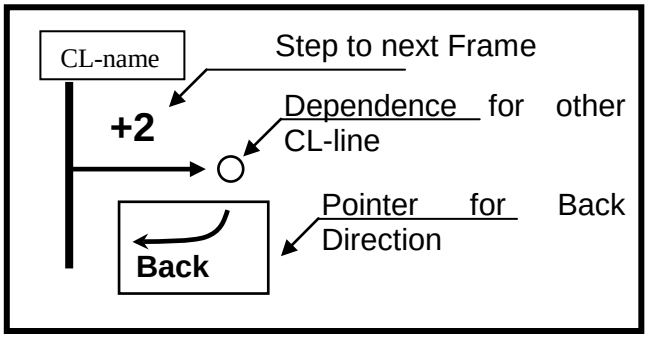
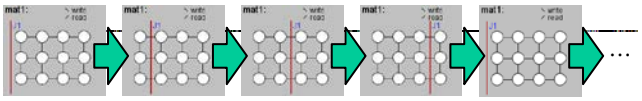


Figure 4. CL-micro Icon Components

The CL-icon components show visually directions of the Control Line movement and step to get next position. If step is omitted then a control line will take adjacent positions. The circle specifies how the current control line may depend on behaviour of another control line. As was pointed above, CF-formulas allow specifying different directions of control lines movement, for example, forward and back movement, etc. To visualize this possibility the CL-micro icon has a special pointer. The icon can also show the name of control line.

3.2 Simple Control Lines Movement

In this subsection, we show examples of icons for specification of simple CL-movement, under which we understand the absence of any dependency between control lines. Figure 5 demonstrates left-right movement of a vertical control line inside a matrix as well as corresponding icon including corresponding specification of a CF-formula. This right-movement is implementing until reaching the last matrix column. After this, the Control Line will jump to the initial position.



	CF-formula mat1_J1
if	mat1_J1<mat1_M-1
then	mat1_J1=mat1_J1+1
else	mat1_J1=0

Micro-icon

Fig. 5. Examples of the Simple CL-movement

The icon can also describe the simultaneous movement of several control lines, for example, as shown in Figure 6.

	CF-formula mat1_J1
if	mat1_J1<mat1_M-1
then	mat1_J1=mat1_J1+1
else	mat1_J1=mat1_J1
	CF-formula mat1_I1
if	mat1_I1<mat1_M-1
then	mat1_I1=mat1_J1+1
else	mat1_J1=mat1_J1

Micro-icon

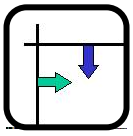
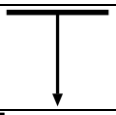
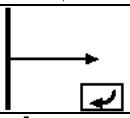
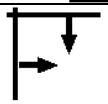
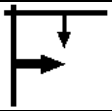
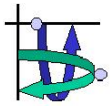
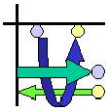


Fig. 6 Concurrent CL-movement.

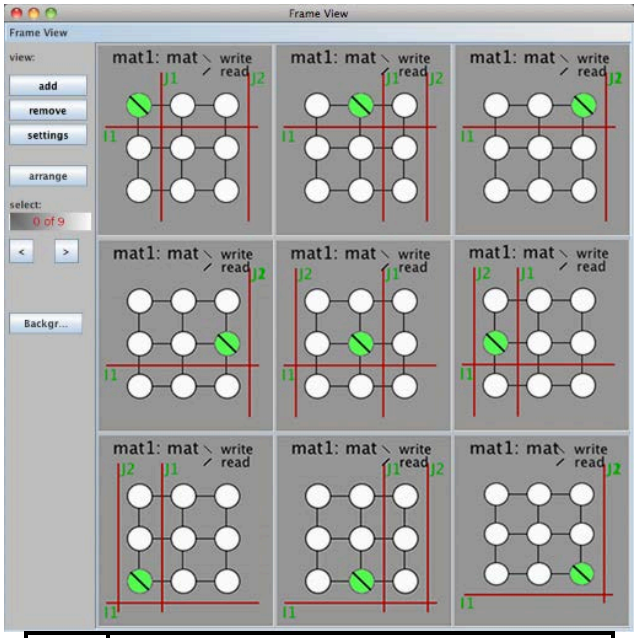
Table 1 contains some examples of CF-icons. The CL-dependences can be represented as to start/stop/reverse movement according some condition is obtained.

Table 1: Examples of CF-icons

NN	Icon	Explanation
1		A horizontal control line moves down (simple behavior).
2		A vertical control line moves right. After reaching a limit, it jumps to its initial position.
3		A vertical control line and a horizontal control line are moving together and synchronously.
4		Control lines are moving in rotation, starting by the vertical control line (fat arrow).
5		A vertical control line turns N times.
6		A vertical control line starts movement. After reaching some point, it reverses its movement, and a horizontal control line starts a movement that also will be reversed after reaching some point.
7		The same situation like in Case 6 the vertical control line stay during the horizontal control line stop to move.

3.2 Complex Scanning Sequence

The proposed approach allows to program visually different types of scanning in 2D-structures. Figure 7 depicts a sequence of frames illustrating row-wised zigzag scanning. To realize it, three control lines were used. The vertical line J2 is to control the increment/decrement movement of the line J1.



	CF-formula: mat1_I1
if	mat1_I1%2==1&&mat1_I1==mat1_M|| mat1_I1%2==0&& mat1_I1==1
then	mat1_I1=mat1_I1+1
else	mat1_I1=mat1_I1
	CF-formula: mat1_J1
if	mat1_I1==mat_J2&&mat1_I1<mat1_M|+1| mat1_J1==mat_J2+1&& mat1_J1>1
then	mat1_I1=mat1_J1+1
else	mat1_J1=mat1_J1
	CF-formula: mat1_J2

if	mat1_I1%2==1
then	mat1_J2=mat1_M
else	mat_J2=0

Fig. 7 Row-wised Zigzag Scanning Sequence.

Figure 7 shows also the CF-formulas for each control line. Even they are rather complex expressions, the designed GUI showed in the next section allows programming and debugging them efficiently.

4. Editing and Debugging

As was discussed above, the one of key point in designing movie-based algorithms is in defining control lines activities in order to update control lines positions during frame transitions. This process mostly related to the specifying CF-formulas for Control Lines attached to a particular structure during processing concrete MP-episode. Therefore, each control line can have several CF-formulas formulas inside different metaframes. CF-formulas can also be used for specification of conditions to finish MP-episodes. These conditions can be, for example, an equality of the control line positions, reaching structure boundaries, etc.

Figure 9 shows the basic GUI to specify control lines. Operations related to the CF-formulas debugging can be implemented simultaneously with editing operations. In any particular metaframe the system allows to animate control lines movement as a result of the CF-formulas evaluation. It is also possible to browse episodes frame-to-frame. The user interface for CF-formulas input includes not only text-based expressions but also icon-oriented subset to simplify some typical control line movements. The interface consists of the following parts:

1. Metaframe view panel with selected control lines,
2. Control line editor panel,
3. CF-formula definition area,
4. Control line formula assistant editor,
5. A special set of icons for typical operations on control line like increment/decrement of position to help construct CF-formula quickly

To introduce new control line, the user should select a corresponding structure; choose the type of a line (horizontal, vertical or diagonal); input a CF-formula. The special GUI was developed on order to manipulate with the control lines icons. It includes the icon synthesis tools using independent and/or cooperative behaviors of vertical, horizontal and diagonal control lines. The basic CF-formula interface has special tools in order to make visual synthesizing and verifying of complex CF-formulas more comfortable. This interface is based on the proposed icon CF-formula language (Figure 10). From the user's point of view, the programming process consists of the following actions:

1. Choose the control lines participating in actions,
2. Select the corresponding icon,
3. Correct CF-formulas by defining their boundaries.

Importantly, the user can also edit and debug CF-formulas directly using basic text-editing tools. After finishing this process, a new formula can be stored in the library, and an icon will be generated or assigned to it.

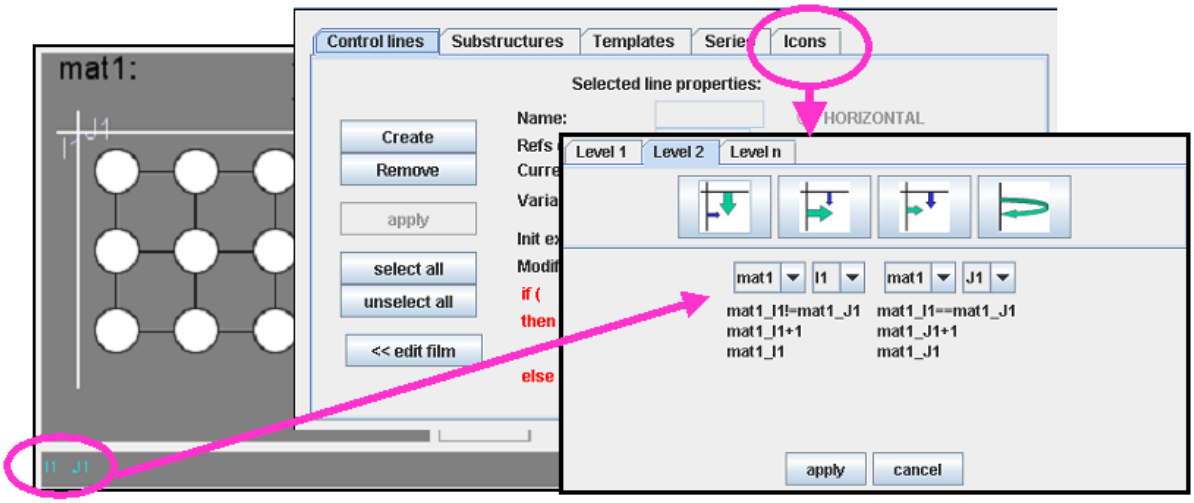


Figure 10. Icon-Oriented GUI

The important feature of the Control-flow GUI is in possibility trace all CL-positions during episode processing. The user can watch animation frames visualizing control line movement during frames transitions or obtain a compact representation of these behaviors in graphical form. Using the first approach, the user can watch the animation movie of the CL-movement. It is also possible to scroll one-by-one all frames in the corresponding MP-episode. The second approach is in graphical representations of the control line positions during frames transitions. Fig. 11 depicts an example of a graph showing positions of two cooperating control lines. It is possible to distinguish and analyze dependences between control lines as well as control lines and episode frame numbers. This allows convenient verifying and debugging of CF-formulas.

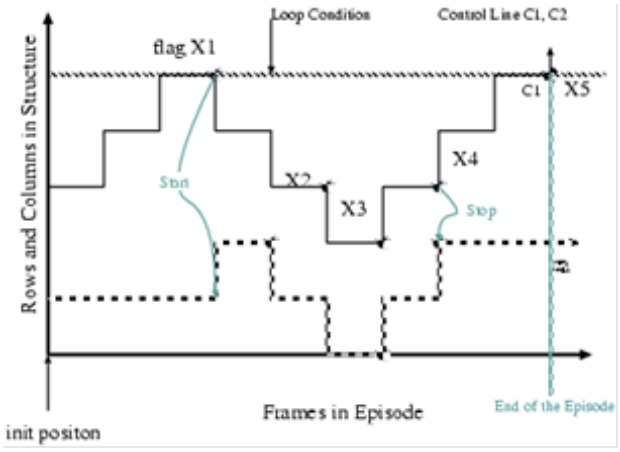


Figure 11. Graphical monitoring of CL-behavior

5. Conclusion

The presented technique is oriented to visualize and simplify programming dynamical behaviors of control lines for matrix structures. The presented semantic rules and proposed icon language allow defining a corresponding number of MP-frames or program iterations that will be generated automatically. The user can watch the animations of the CL-movement or one-by-one scrolling of all frames in the corresponding MP-episode. Moreover, this movement can be represented as a graph that can reflect behaviors of several control lines. This combination improves efficiency and time of debugging matrix algorithms. The results of testing confirm that the presented system can be used not only as an algorithm demonstration tool but also as a programming tool. The system presented is realized on Java. It generates C/C++ programs including OpenMP platform and can export movies in the Macromedia Flash Animation format.

Our investigations now are oriented to extend a set of icons in order to cover the wider range of matrix algorithms.

References

[1] J. Stasko, J. Dominique, M. Brown, and B. Price, *Software Visualization: Programming As a Multimedia Experience*, The MIT Press, 1998.

[2] M. Boshernitsan and M. Downes. *Visual Programming Languages: A Survey*, Report No. UCB/CSD-04-1368. University of California, California, USA, 2004.

[3] J. Edwards. Subtext: Uncovering the Simplicity of Programming. *In Proc. of the 20th Annual ACM SIGPLAN conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA'05)*, San Diego, California, USA, 2005, pp. 505-518.

[4] A. Blackwell and T. R. G. Green. Does Metaphor Increase Visual Language Usability? *Proc. of the 1999 IEEE Symposium on Visual Languages VL'99*. Tokyo, Japan, 1999, pp. 246-253.

[5] S. Tanimoto, Programming in a Data Factory, *Proc. of Human Centric Computing Languages and Environments (HCC'03)*, Auckland, 2003, pp. 100-107.

[6] R. Oechsle, and T. Schmitt, JAVAVIS: Automatic Program Visualization with Object and Sequence Diagrams Using the Java Debug Interface (JDI), *LNCS*, Vol. 2269, Springer-Verlag, 2002, pp. 1-15.

[7] B. T. Westphal, F. C. Harris, and S. Dascalu, Snippets: Support for Drag-and-Drop Programming in the Redwood Environment, *The Journal of Universal Computer Science*, Vol. 10, No. 7, 2004, pp. 859–871.

[8] Fl. Devin, P. Boulet, J.-L. Dekeyser, and Ph. Marquet. GASPARD - a Visual Parallel Programming Environment. *Proc. of the Int. Conference on Parallel Computing in Electrical Engineering (PARELEC'02)*, Warsaw, Poland, 2002, pp. 145-150.

[9] N. Mirenkov, A. Vazhenin, R. Yoshioka, Ts. Ebihara, at al., Self-Explanatory Components: A New Programming Paradigm, *Int. Jour. of Soft. Eng. and Knowledge Eng.*, vol. 11, No. 1, 2001, pp. 5-36.

[10] Yu. Watanobe, R. Yoshioka, and N. Mirenkov. Programming in Pictures: a Way Toward Reliable Software, *Frontiers in Artificial Intelligence and Applications*, IOS Press, Vol. 231, pp. 183-197, 2011.

[11] D. Vazhenin and A. Vazhenin. MP-templates Operating Toolkit in Movie-based programming. *Proc. of the IEEE Japan-China Joint Workshop on Frontier of Computer Science and Technology*, Nagasaki, Japan, 2008, pp. 67-74.

[12] D. Vazhenin and A. Vazhenin, On-line Debugging Methods and Tools in Movie-based Programming, *Proceedings of the 10th WSEAS International Conference on Applied Computer Science (ACS'10)*, Iwate, Japan, 2010, pp. 418-425.

[13] D. Vazhenin and A. Vazhenin, Implementation of Movie-based Matrix Algorithms on OpenMP Platform, *Proc. of*

the Federated Conference on Computer Science and Information systems (FedCSIS 2011), Szczecin, Poland, 2011, pp. 491 – 494.

[14] L. Grunske, K. Winter, and N. Yatapanage. Defining the Abstract Syntax of Visual Languages with Advanced Graph Grammars - a Case Study Based on Behavior Trees, *J. Vis. Lang. Comput*. Vol. 19, No 3, 2008, pp. 343-379.

[15] Eddy Z. Zhang, Y. Jiang, Z. Guo, K. Tian, and X. Shen. On-the-fly Elimination of Dynamic Irregularities for GPU Computing. *Proc. of the Sixteenth Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS '11)*, NY, USA, 2011. ACM, pp.369-380.

Alexander Vazhenin received his M.S in Computer Engineering from the Novosibirsk State Technical University (Russia) in 1978. He received his PhD in Computer Science from the Institute of Informatics Systems of the Siberian Division of the Russian Academy of Sciences in 1993. He published about 100 refereed academic papers. His research and educational interests include parallel architectures, algorithms and programming tools, self-explanatory software high-accuracy computations, visual, and multimedia and Internet technology. He has been program and organizing committee member of many international conferences. He is currently senior associate professor at the University of Aizu, Japan.