

Movie-based Templates for Linear Algebra Problems

Dmitry Vazhenin[†], Nikolay Mirenkov^{††} and Alexander Vazhenin^{†††}

Graduate School Department of Information Systems, University of Aizu, Aizu-Wakamatsu, 965-8580 Japan

Summary

The paper discusses program design approaches supporting effective and convenient programming. The first approach is based on movie-based representation of algorithms and programs. The user has a deal with special multimedia objects, each of which can generate an executable code as well as produce animation frames. These objects build an algorithmic skeleton representing the steps of computation. The other direction is the template metaprogramming technique in which templates are used by a compiler to generate temporary source code. The presented work is in combining both approaches in the software design. The key point is in using an application-oriented movie-based templates library. The important feature of this library is that it is open for addition new templates, and all components can be prepared by means of the movie-based environment. The other peculiarities are the presence of two types of components: functional modules and multimedia macros that can be easily embedded into the user's algorithmic skeleton. In this paper, we show the main features of movie-based programming as well as describe the movie-based template library for linear algebra problems. Examples of the library usage are also presented.

Key words:

Visual Programming, Movie-based Programming, Template Metaprogramming, Linear Algebra Problems.

1. Introduction

The most of modern applications include a big variety of attractive multimedia functions with icons, pictures, animations, sound and other multimedia components providing to the user reliable understanding as well as effective manipulating with the complex objects. This allows programming with visual expressions, direct manipulating visual information as well as supporting visual interaction. The market has already discovered the fascination of visual programming, and various new programming environments have been declared to be "visual". Visual programming languages and tools may be classified according to the type and extent of visual expression used, into icon-based languages, form-based languages and diagram languages. Visual programming environments provide graphical or iconic elements which can be manipulated by the user in an interactive way according to some specific spatial grammar for program [1-5].

Multimedia approach for interactive specifications of applied algorithms and data representations is based upon a collection of computational schemes represented in the "film" format proposed in [6]. In [7], we presented an extension of this approach called the Movie-Based Programming. The programming process is in manipulating with special movie-program objects (MP-objects) generating *automatically* a part of an executable code as well as producing frames, which are *adequate* to the code generated. Both movie and program can *synchronously* be generated and debugged.

The important direction to increase the programmer's productivity is so-called template metaprogramming especially for C++ users [8]. It is a programming technique in which templates are used by a compiler to generate temporary source code, which is merged by the compiler with the rest of the source code and then compiled. The use of templates can be thought of as compile-time execution. This allows using the user-oriented programming language reflecting all specifics of concrete applications. There are some drawbacks to the use of text-based templates. First, many compilers historically have very poor support for templates, so the use of templates can make code somewhat less portable. Second, almost all compilers produce confusing, unhelpful error messages when errors are detected in template code. This can make templates difficult to develop. Third, each use of a template may cause the compiler to generate extra code (an instantiation of the template), so the indiscriminate use of templates can lead to code bloat, resulting in excessively large executables. The extra instantiations generated by templates can also cause debuggers to have difficulty working gracefully with templates. For example, setting a debug breakpoint within a template from a source file may either miss setting the breakpoint in the actual instantiation desired or may set a breakpoint in every place the template is instantiated.

The presented work is in combining both approaches in the software design. The key point is in using an application-oriented movie-based templates library. The important feature of this library is that it is open for addition new templates, and all components can be prepared by means of the movie-based environment. The

other peculiarities are the presence of two types of components: functional modules and multimedia macros that can be easily embedded into the user's algorithmic skeleton.

The methods and tools of linear algebra are often used in many scientific and practical computations. The kernel of these algorithms is matrix data and operations. Traditionally, the linear algebra software consists of well-maintained libraries available commercially or electronically in the public-domain like LINPACK and BLAS packages [9], other libraries distributed with texts or other books, individual subroutines tested, as well as individual or electronic sources which may be hard to use or come without support. As shown in [10], there is a need for tools to help users in picking the best algorithm and implementation for their numerical problems, as well as in getting expert advises on how to tune them. Authors suggested for using special text-based forms called templates. This approach can be considered as a good idea to help in choosing the suitable algorithm. Nevertheless, essential problems with understanding computational schemes presented still exist. This research is devoted to design the new the movie-based linear algebra templates including objects having mentioned-above features.

This paper has the following structure. In the section 2, features are shown of movie-based programming technology. In this section, we discuss also main stages of the movie/program design and debugging. Section 3 contains description of movie-based matrix library. Some remarks and examples of the movie-based library usage are shown Section 4. Conclusion and future work are discussed in Section 5.

2. Movie-based Programming Process

2.1 Basic Concepts

The movie-based representation of algorithms and programs is in showing their features as a sequence of animation frames. Accordingly, any animation frame should visualize/animate a corresponding stage of a program/algorithm execution (Fig. 1). We define such a frame as the Movie-Program Frame or MP-frame. Each MP-frame highlights and flashes some elements of a parameterized matrix structures defining operations or formulas. Different operations can be coded by different colours. Special Control Lines (I1, I2, J1, J2) are used to reference these areas of activities. They can change their placement inside matrix during frame transitions.

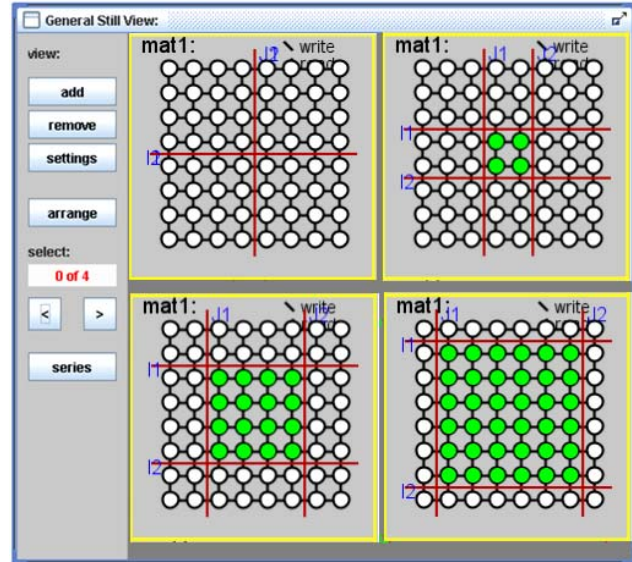


Fig. 1 Algorithmic Movie Example

The Movie-based Programming is in manipulating with special objects generating a part of an executable code as well as producing MP-frames, which are adequate to the code generated.

2.2 MP-skeleton Components

MP-skeleton is a container of MP-components (Fig. 2). In other words, it is a collection of multimedia templates defining how to generate a movie or program. It consists of MP-films. Each MP-film is a set of MP-stills or scenes that are responsible for generating MP-frames as well as corresponding executable code. There exists a simple *MP-still* generating one MP-frame. An *EPISODE-still* produces a series of frames. The main feature of MP-episode is that the same operations should be implemented on all its internal frames.

The *HEAD-still* should be the first still in any film. It contains description of data structures and variables used in a current film. The *END-still* is to finalize a film. *IF-still* is to skip or process selected groups of stills. The user should specify a logical conditional expression as well as mark stills that will be processed for true and false cases correspondingly. The *WHILE-still* is to repeat the processing of stills marked while a condition is true. The *CALL-still* is to pass processing to other MP-films. In this case, the END-still will return control to the parent film.

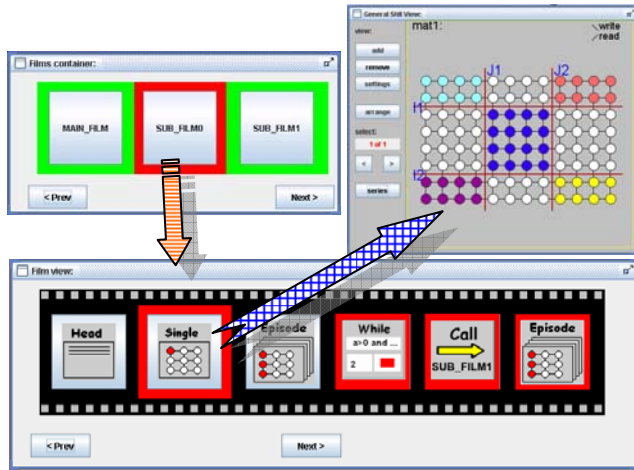


Fig. 2 MP-skeleton components

In most cases, MP-film consists of several stills. Each still contains a set of traversal schemes specifying coloured domains in corresponding structure. Each schema has its own colour and *formula sequence* attached to this colour. The same nesting scheme is always reflected in the corresponding program source code. Each still produces one or several static frames representing skeleton steps of computation and hiding formulas.

Actually, any algorithm represents data structures as well as an order of operations or *formulas* implementing on structure elements or nodes. Therefore, MP-structures are other important components of each MP-film. Each MP-structure includes the following attributes:

- The unique structure name is to identify a structure from other structures.
- Parameters or variables are for defining structure size, for example, number of rows and number of columns for matrix structures. Importantly, these parameters have two values. The first value is used for an animation movie, and the second is used for generating program.
- Structure control components are used to reference activities areas inside a structure (Fig.2). This means that control objects divide a structure into zones each of which can have individual color. Different colors mean that different operations can be implemented on the corresponding nodes. For matrices, we have a deal with vertical (J-lines), horizontal (I-lines) and diagonal lines (D-lines). They can change their placement inside matrix during frame transitions.
- Structure variables can be simple doubles, integers, etc. as well as a composite type like strings, complex

numbers, etc. Each structure can have several variables.

The main order of operations is defined by MP-stills generating computational steps (MP-frames). The Control Flow Formulas or *CF-formulas* are introduced to coordinate operations between frames as well as program the control lines behavior. Involving showed below, the user can specify control lines behavior as well as define a corresponding number of MP-frames or program iterations will be generated.

1. **CF_ID** – Control Line Name
2. Initialization Rule (Start position):

CF_ID = <expression>;

3. Transition Rule (Next position):

if (<condition>)
then CF_ID = <expression 1>;
else CF_ID = <expression 2>;

4. Episode Rule (How to finish episode):

If (<condition>)
then {Generate Next Frame};
else {Finish Episode};

Actually, CF-formulas are to create a shape of computing and define a distribution of computations over MP-nodes. In other words, an algorithmic movie shows data structures and some activities on these structures. To precisely specify their activities, it is necessary to attach arithmetical and/or logical formulas to corresponding nodes. These formulas are called *Computational formulas* or *C-formulas*. We define a C-formula as a subprogram containing a sequence of arithmetical and logical expressions. Each C-formula includes the following components: *MP-expressions*, *Control structures*, *Regular text*. *MP-expressions* are to specify data access and operations on MP-nodes. The C-formula notation is to the conventional mathematical expressions. Example of C-formula is shown below:

$$A[\bullet] = A[\bullet] - A[\text{mat}1_{j1}-1, \bullet] * \frac{A[\bullet, \text{mat}1_{j1}-1]}{A[\text{mat}1_{j1}-1, \text{mat}1_{j1}-1]}$$

We are enhancing C-formulas by using special multimedia attributes like images, symbols and tables in order to improve the formula perception. *Control structures* are

used to point branch conditions. **Regular text** can be comments and/or a custom code, which extends formula capabilities.

2.3 Design Stages

The general scheme of a program movie-based design process includes the following four stages:

- Creating the algorithmic MP-skeleton,
- Attaching Formulas,
- Generating, Executing and Debugging Movie-based Program,
- Exporting algorithmic movie and program to the target machine.

The algorithmic MP-skeleton design process includes specifications of MP-objects like MP-stills, control lines, structures (scalars, vectors and matrices) as well as CF-formulas for each control line. Figure 3 depicts the user's activities in the MP-still creation.

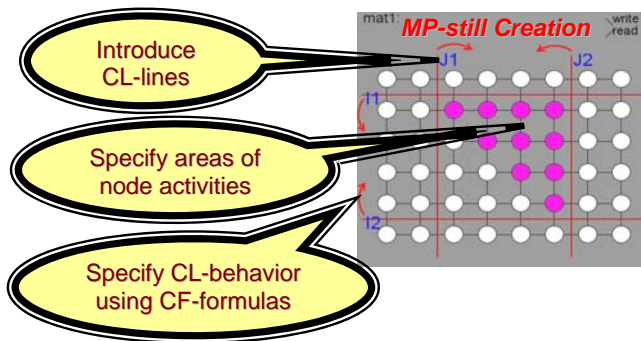


Fig. 3 MP-still creation/editing

The user should also attach the C-formulas for colored MP-nodes using special visual expressions on the Formulas Attachment interface (Fig. 4). After finishing these processes, the user can finally execute movie-based programs and generate executable codes in C++.

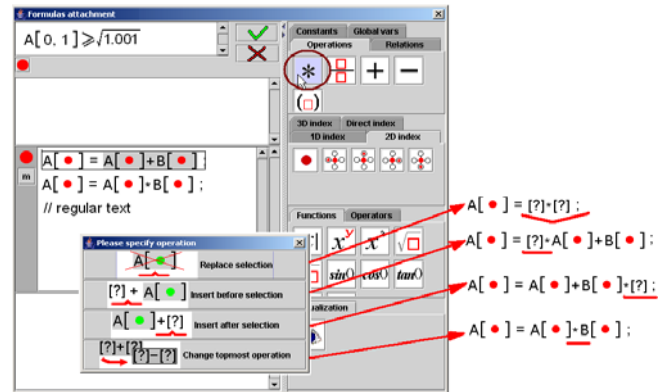


Fig. 4 C-formula attaching

Information stored in MP-templates is used to generate MP-frames as well as an executable code. During code generation, some template components (scanning loops, variables, etc.) will be simply transferred in the final code defined by a target system. C-formulas will be converted to the final code after additional verification. To generate MP-frames, the MP-templates are also used to form images and other graphical information. Calculations using C-formulas attached can also be implemented, and a movie will be generated representing only one possible case of a MP-program execution obtained according to the real data. It is also possible to generate a movie from a MP-skeleton with non-complete formulas and conditions. In this case, MP-frames with images can only be generated. The user may randomize or specify directly branches needed for IF- and WHILE-stills. As was mentioned, the movie and program have different size parameters of MP-structures. This leads that a movie and program will have/reflect different numbers of MP-frames.

The executable code should be generated using this linear textual representation with the substitution rules from semantic tables applied. Thus, the visual terms are represented by their corresponding textual terms. As shown in [7], during formula attachment, a debugging scheme allows visualizing and controlling all references to the structure elements. This allows debugging film structure and formulas activity during design-time. The formula tracing technique is used visualizing nodes referred by a formula on a particular frame (Fig. 5).

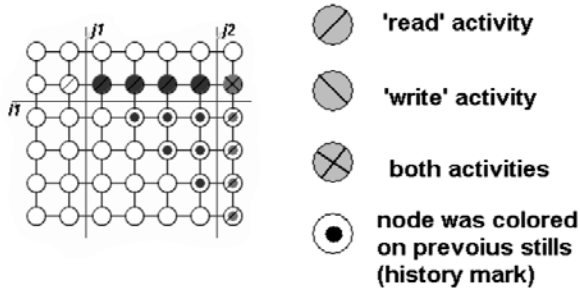


Fig. 5 C-formulas tracing

The Movie-based Program is adequate to the corresponding movie behavior. The Run-Time Debugging makes possible verifying a movie-based program data-flow using special breakpoints. When such a breakpoint achieved, the program stops, and the executor invokes the data visualization tool. Information provided to the user includes a global frame and still numbers and a frame number inside episode. He/she can choose to either continue/terminate execution, or return to design stages

3. Movie-based Template Library

3.1 Library Organization

The efficiency and convenience of the most of programming systems depends on variety of embedded packages and libraries. Those tools can significantly reduce the software design expenses as well as help in understanding of computational methods. That is why we've designed the linear algebra library as a collection of movie-based matrix routines and procedures.

As was pointed above, each MP-still includes a set of templates used for generating corresponding animation frames and parts of executable code. Moreover, each library component can be used as a movie-based template (MP-template) during MP-skeleton preparation. There are two types of templates: functional modules and multimedia macros that can be easily embedded into the user's algorithmic skeleton. Templates of the first type are MP-films. The user can insert them into MP-skeleton and connect to other film using *CALL-still*. Templates of the second type are MP-stills that can be imported into the user's MP-film. Importantly, MP-template can be fully or partially prepared according to the presence or absence of CF-formulas. The movie-based library is open for extension. This means that the can add new templates or edit existing components. Moreover, all components can be prepared by means of the movie-based environment.

3.2 MP-library Sections

All library components are divided into sections according to their functional specialization. This allows covering practically all stages of information processing including inputting/generating initial data, processing as well as visualization of result. In this section, we show some examples of library components related to the linear algebra problems.

Matrix Generators and Service Functions are a special set of standardized MP-stills including movie-based algorithms for obtaining matrices with given types and features. These matrices are necessary in developing and evaluating many matrix algorithms. Table 1 contains examples of MP-templates that can be used as generator of input data. Some other templates are to print or visualize results during implementing of MP-program.

Table 1: Matrix Generators and Service Functions

<i>Band Matrix defined by five Control Diagonal Lines</i>	<i>Five Diagonal Band Matrix defined by band templates</i>

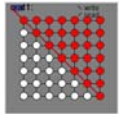
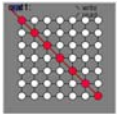
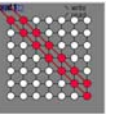
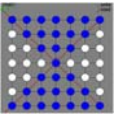
Matrix operators include a set of standard matrix operation like scalar and vector products, matrix-vector and matrix-matrix multiplications, transposition, etc. Table 2 shows examples of different schemes of matrix multiplication.

Table 2: Matrix operations

<i>Row-column Matrix Multiplication</i>	<i>Row-row Matrix Multiplication</i>	Matrix Transposition


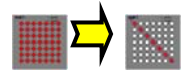

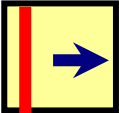
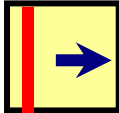
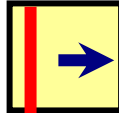
Usually, direct methods of solving systems of linear algebraic equations (SLAE) are based on transformation of the initial matrix to a standard matrix with a standard shape. This section includes a set of *Direct SLAE solvers with standard matrix shapes* (Table 3).

Table 3: SLAE solvers with standard matrices

Triangular matrix	Diagonal matrix	Bi-diagonal matrix	Bi-triangular matrix
			

Direct SLAE solvers are classified according to the type elimination procedure like triangulation, diagonal matrix, bi-diagonal, etc. The other classification is according to the way of elimination of unknown including by-element, by-row, by-column eliminations, etc. (Table 4). In other word, it reflects a way how zones of computation can be reduced. The computation reduction is because of number of zeroes element is increased during eliminating unknowns. Operations on these elements can be avoided by changing MP-nodes during frames transitions. Methods are also distinguished according to the CF-formulas, for example, Gauss-Jordan of division-free elimination.

Table 4: Direct SLAE solvers

Gaussian Elimination	Gauss-Jordan Elimination	Division-free Elimination
		
		

There are a great variety of iterative methods for 2-D problems. However, as a rule, they are also reduced to a rather limited number of implementation schemes. These schemes are directly related to how scanning operations on corresponding matrix elements are performed. Library contains a set of MP-stills realizing *Standard Iterative Schemes* like Jacoby and Gauss-Seidel iterations.

4. Export/Import of Library Components

Any movie-based library component can easily be imported or inserted into the user's application skeleton. Importantly, the library is open for changing. This means that the user add/remove components in library.

4.1 Operations on MP-film library

The first library part is a collection of MP-films each of which represents a complete method and consists of a MP-still series. Figure 6 shows how the user can work with the MP-films database. As shown in Figure, there are four steps to import MP-film in his/her MP-skeleton:

1. Select a suitable MP-film from the corresponding library section and insert it in the user's MP-skeleton,
2. Create the Call MP-still in the corresponding parent MP-film,
3. In the Call MP-still, specify MP-components like Control Lines, structures and variables used in both parent and child films.
4. Tune the child MP-film defining values of MP-components used locally.

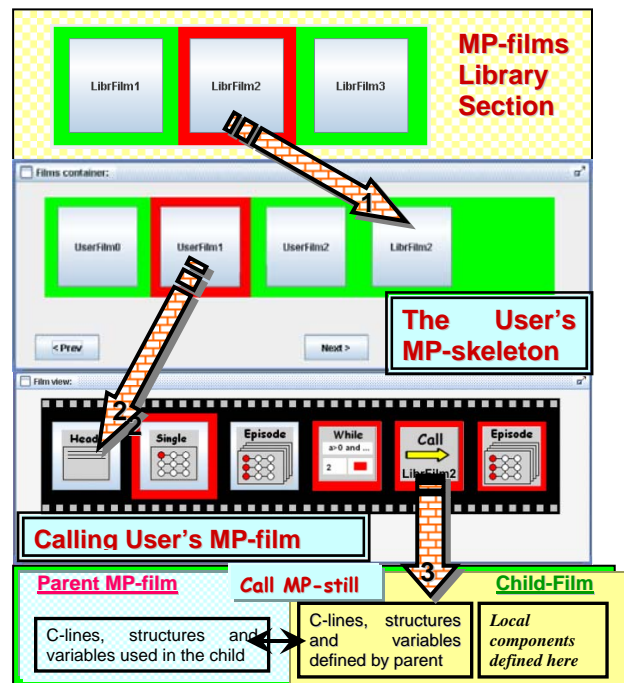


Fig. 6 Import a library MP-film

Importantly, it is possible to have different variant of calling the MP-films allowing the flexible use of the library components.

4.2 Operations with MP-still template database

Import operations on stills and episodes have some specifics because of necessity in redefining variables and structures names, sizes, etc (Figure 7). The user should specify a mapping of imported entities and real variables and structures defined in the user's MP-film. All formulas will be transformed according to the user's film notations. If an inserted MP-still or episode has an extended set of variables and structures, all unspecified components will be added to the user's film component set.

According to this technique, MP-skeleton generating the Gaussian Elimination Movie or program can have two episodes and two MP-stills. The first episode is the Gaussian Elimination Procedure implementing matrix triangulation. Back Substitution Episode represents obtaining the final solutions.. The first MP-still is the matrix generator. The final MP-still can be used, for example, for printing results

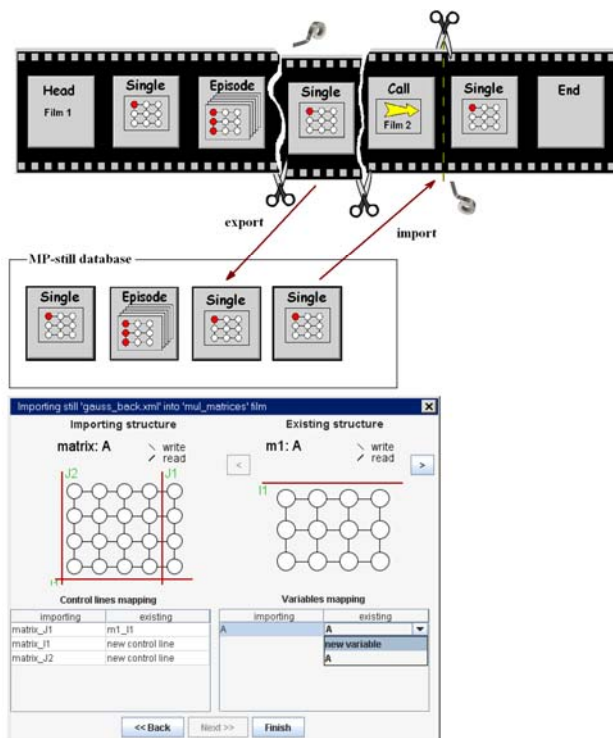


Fig. 7 Export/Import a library MP-still

5. Conclusion

The proposed movie-based library can help in designing and debugging matrix algorithms because of combining an executable code generation with the visual representation of algorithms and programs. The library is open and allows adding/designing new matrix algorithms. The programmer can easier understand relations between a real application and algorithm used for it. During design of a formula sequence, the user should operate only with visual objects specifying variable names and index expressions as multimedia symbols. 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 platform. It generates C/C++ programs and can export movies in the Macromedia Flash Animation format. Our further investigations are related in extending the set of components as well as designing

References

- [1] J. Stasko, J. Dominique, M. Brown, and B. Price, Software Visualization: Programming As a Multimedia Experience, The MIT Press, 1998.
- [2] Limnor Tutorial. ©2003 Longflow Enterprises Ltd. <http://www.limnor.com/>
- [3] S. Tanimoto, "Programming in a Data Factory," Proc. of Human Centric Computing Languages and Environments, Auckland, pp. 100-107, 2003.
- [4] R. Oechsle, and T. Schmitt, "JAVAVIS: Automatic Program Visualization with Object and Sequence Diagrams Using the Java Debug Interface (JDI)," LNCS, Springer-Verlag, Vol. 2269, pp. 1-15, 2002.
- [5] M. Boshernitsan, M. Downes Visual Programming Languages: A Survey, Report No. UCB/CSD-04-1368, University of California, 2004.
- [6] N. Mirenkov, A. Vazhenin, R. Yoshioka, Ts. Ebihara, et al., "Self-Explanatory Components: A New Programming Paradigm," Int. Jour. of Soft. Eng. and Knowledge Eng., vol. 11, no. 1, pp. 5-36, 2001.
- [7] D. Vazhenin, A. Vazhenin, and N. Mirenkov, "Movie-based Multimedia Environment for Programming and Algorithms Design," LNCS, Springer-Verlag, Vol. 3333, Part III, pp. 533-541.
- [8] D. Abrahams, A. Gurtovoy, "C++ Template Metaprogramming", Addison-Wesley, 2004.
- [9] <http://www.netlib.org>
- [10] Z. Bai, D. Day, J. Demmel, J. Dongarra, M. Gu, A. Ruhe, and H. Vorst, "Templates for Linear Algebra Problems," LNCS, Springer-Verlag, Vol. 1000, Springer-Verlag, 1996.



Dmitri Vazhenin received his B.S and M.S. degrees in Applied Mathematics and Informatics from Novosibirsk State technical University (Russia) in 2001 and 2003, respectively. He published about 20 referred papers. His research/educational interests include parallel algorithms, visual programming tools, multimedia and

Internet technologies. He now is a PhD student at the University of Aizu, Japan.



Nikolay Mirenkov received his M.S degree in 1967, PhD in 1971, and D.Sc in 1983 in Computer Science from Institute of Mathematics (Russia). He authored and edited 12 books and published more then 150 refereed papers. His research/educational interests include parallel programming systems, self-explanatory software components, human-computer

interface and new formats for data/knowledge representation. He has been organizer and program committee member of many international conferences. He is currently professor and head of Distributed Processing Lab at the University of Aizu.



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 more then 70 refereed 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 associate professor at the University of Aizu, Japan.