ALGORITHM VISUALIZATION USING THE VIZALGO PLATFORM

Slavomír ŠIMOŇÁK
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice,
Letná 9, 042 00 Košice, Slovak Republic, tel. +421 55 602 3021, e-mail: slavomir.simonak@tuke.sk

ABSTRACT

Algorithms are central objects of every nontrivial computer application. Every programmer thus should have basic knowledge from the area of algorithm design in order to work more effectively. Simplifying and deepening the understanding of algorithms operation is the main goal of algorithm visualization. There is active research and development in the area of algorithm visualization in the last decades. Solutions of different quality and serving different purposes are available nowadays, but often it is not an easy task to modify or extend the particular solution to meet specific needs exactly. The paper contains concise overview of development in the area of software visualization, as well as a proposal of software visualization platform with the emphasis on its extensibility and portability.

Keywords: *Algorithm visualization, software visualization, visualization systems, plugin-based visualization platform, Java Simple Plugin Framework (JSPF)*

1. INTRODUCTION AND MOTIVATION

Based on the results of research in the visualization area [1,2] as well as on our own experiences we believe that visualization of algorithms can help significantly in simplifying and deepening the understanding of algorithms operation. However, it is not an easy task to develop a purposeful visualization tool, which would support teaching and research processes effectively. It is important to find out what makes the visualization tools effective and helpful. To do so, first we will pay attention to studies that have been already done in the field of software visualization and then, based on the knowledge achieved, a new visualization tool will be proposed.

Except the algorithm visualization, the term *software visualization* is also often used within the papers published in last years. It usually covers both visualization of algorithms and visualization of data structures, but sometimes also another aspects of software are considered, too [3]. Algorithm visualization, as part of software visualization, could be described as ”graphical representation of an algorithm or program that dynamically changes as the algorithm runs” [2]. Data structure animation systems usually are considered to be systems that support repeated graphical display of data structures with changes in content and time to show the viewer how the data are being transformed during execution [39].

More general definition of software visualization can be found in [3], where the author describes it as ”visualization of artifacts related to software and its development process”. Artifacts here can mean program code, requirements and design documentation, and bug reports. Main aspects of software which can be visualized then include visualization of its structure, behavior, and evolution.

An overview of visualization taxonomies [7], together with an analysis of factors increasing the effectiveness of software visualization, is summarized in [4].

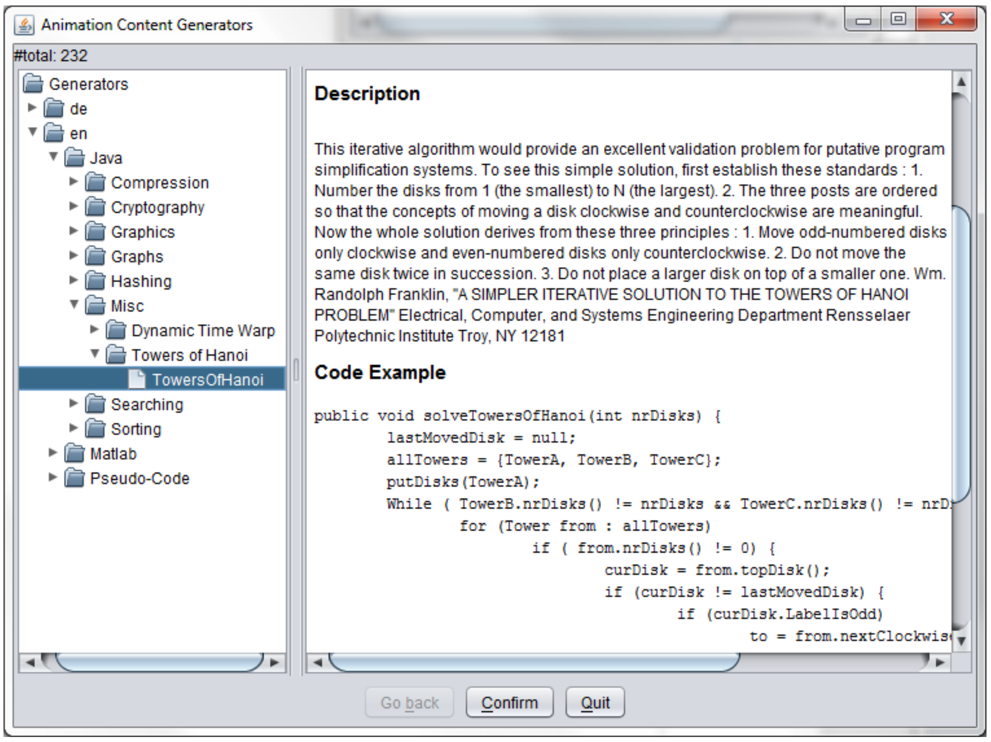
2. OVERVIEW OF EXISTING SOLUTIONS

Even if the beginnings of software visualization date back into the 1940’s [5], the greatest development in the area we could observe within the last 20-30 years. Modern approaches to software visualization were brought in the 1980’s by the introduction of system BALSA (Brown & Sedgewick, Brown University, USA) [6] which represented a framework for interactive animations of algorithms with support of dynamic views on algorithms and data structures. Later it was improved (BALSA-II) and offered interactive visualizations of programs written in Pascal. System itself was written in the C language and according to [7] it was the first one allowing the comparison of operation and performance of algorithms on a single display. A view of source code of the currently executed procedure with the current line highlighted was usually provided, too. For a couple of years the system was in active use within the teaching process as well as a support tool for the design and analysis of algorithms.

In the 1990’s, several new visualization systems started to appear. One of the most famous was the project TANGO (Stasko, Brown University, USA), which brought fluent animations into the area, followed by its improvements called Xtango (using X11 Window System) and Polka. Project Zeus was the first one in which colors and sounds were used [8]. While it is an evolution of system BALSA, Zeus was implemented in multi-threaded and multi-processor environment and utilized MIDI synthesizer. 3D graphics in projects Polka-3D and Zeus-3D was used for the first time in the area of software visualization.

A list of interesting tools of the 1990’s includes also ANIM (Bentley & Kernighan, AT&T Bell Laboratories) [9] - a system for creating animation and static figures from the scripts generated during the program runtime (so the original program was adapted by adding output commands in its interesting places), UWPI (University of Washington Program Illustrator, Henry & Whaley & Forstall, University of Washington, Seattle, USA) [10] creating visualizations of high-level abstract data structures, TPM (The Transparent Prolog Machine, Eisenstadt & Brayshaw, Open University, UK) - graphical tracer and debugger for the declarative language Prolog, or Pavane (Roman, et al., Washington University, Saint Louis, USA) [12] - dedicated for three-dimensional visualization of parallel programs. Illustrative figures or animations, in some cases capturing the operation of tools from the 1980's and the 1990's, can be found in [7].

From about the year 2000 until the present time, many interesting systems dedicated to software visualization have been created. The project ANIMAL [13, 14], (Technical University of Darmstadt, Germany) brought a system for creating algorithm animations completely developed using the Java language. An interesting feature of the system is the possibility to generate algorithm visualizations (Fig. 1) from the different areas of computer science.



Description

This iterative algorithm would provide an excellent validation problem for putative program simplification systems. To see this simple solution, first establish these standards : 1. Number the disks from 1 (the smallest) to N (the largest). 2. The three posts are ordered so that the concepts of moving a disk clockwise and counterclockwise are meaningful. Now the whole solution derives from these three principles : 1. Move odd-numbered disks only clockwise and even-numbered disks only counterclockwise. 2. Do not move the same disk twice in succession. 3. Do not place a larger disk on top of a smaller one. Wm. Randolph Franklin, "A SIMPLER ITERATIVE SOLUTION TO THE TOWERS OF HANOI PROBLEM" Electrical, Computer, and Systems Engineering Department Rensselaer Polytechnic Institute Troy, NY 12181

Code Example

```
public void solveTowersOfHanoi(int nrDisks) {
        lastMovedDisk = null;
        allTowers = {TowerA, TowerB, TowerC};
        putDisks(TowerA);
        While ( TowerB.nrDisks() != nrDisks && TowerC.nrDisks() != nrD
                for (Tower from : allTowers)
                        if ( from.nrDisks() != 0) {
                                curDisk = from.topDisk();
                                if (curDisk != lastMovedDisk) {
                                        if (curDisk.LabelIsOdd)
                                                to = from.nextClockwis
```

Fig. 1 The ANIMAL system

The current version of the system is available at its home page [14], together with corresponding documentation and sample animations.

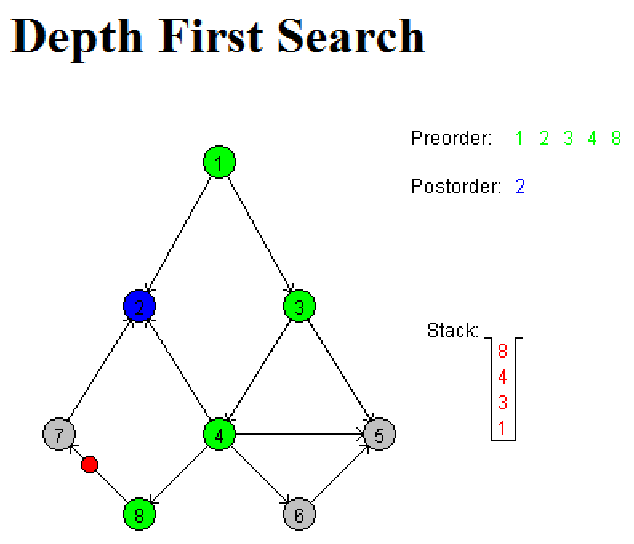


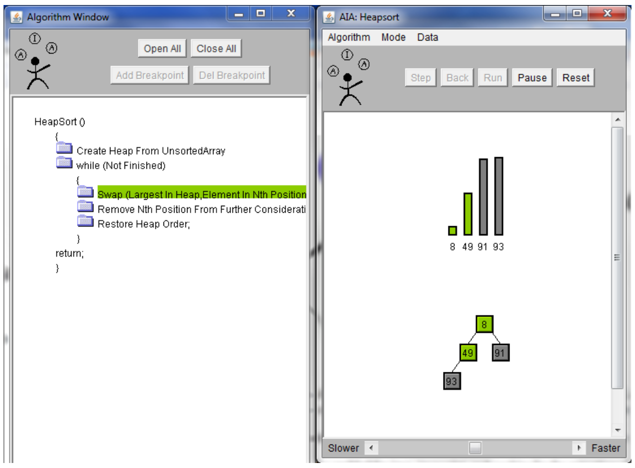
Fig. 2 The JAWAA visualization

The JAWAA system (Duke University, Durham, NC, USA) is developed in Java and provides the scripting language and development environment for simple creation of algorithm animations with the possibility of their subsequent displaying using the web browser.

JAWAA scripts can also be generated by the program written in an arbitrary programming language and so creating required animations (e.g. changes in data structures). Language commands include primitive objects (circle, line, text, rectangle, ...), generic actions (changeParam, moveRelative, delay, ...), data structures (array, queue, stack, tree, ...) and actions over the data structures (push, pop, enqueue, ...), which can be used for visualization design. The project homepage [15], except the basic system-related information and papers published, contains some demonstrating visualizations, too (Fig. 2).

System Algorithms In Action [16] (University of Melbourne, Australia) was developed as a supporting tool for teaching algorithms. Application's interface consists of several windows displaying algorithm pseudocode, explanations and algorithm visualization itself (Fig. 3). Algorithm visualizations implemented currently are subdivided into four groups: searching algorithms (Binary search tree, 2-3-4 tree, RB tree, Skip list, ...), sorting algorithms (Shellsort, Quicksort, Heapsort, ...), string processing (KMP, Boyer-Moore), and graph algorithms (BFS, DFS, Minimal spanning tree, ...). System is developed using Java and Java script programming languages.

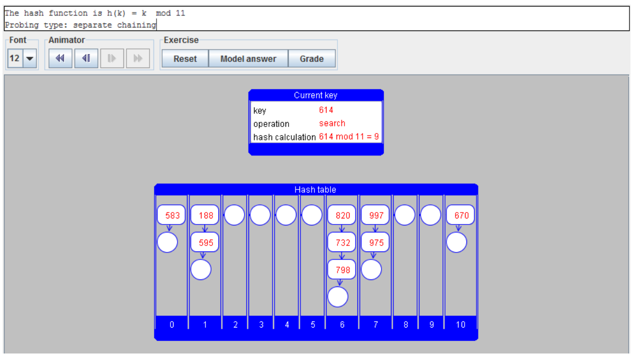
TRAKLA2 (Helsinki University of Technology, Finland) is an environment for teaching of data structures and algorithms as well as for visual assessment of knowledge acquired from the given area. An overview of main features of the system is nicely captured in a demonstration video [18].



```
HeapSort ()
{
  Create Heap From UnsortedArray
  while (Not Finished)
  {
    Swap (Largest In Heap,Element In Nth Position
    Remove Nth Position From Further Considerati
    Restore Heap Order;
  }
  return;
}
```

Fig. 3 Algorithms In Action

Within the project homepage [17], except the basic information about the system, the list of publications, download section, also the TRAKLA2 exercise package is available. This package contains a lot of prepared algorithm visualizations with the self-testing feature implemented (Fig. 4) to get the immediate feedback using the automatic assessment system. A web browser with the Java support is required.



```
The hash function is h(k) = k  mod 11
Probing type: separate chaining
```

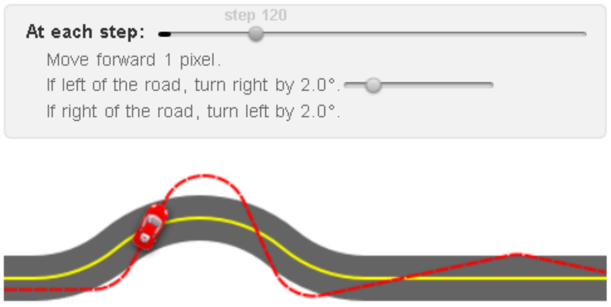
key	614
operation	search
hash calculation	614 mod 11 = 9

Fig. 4 TRAKLA2 exercise package

Very unusual, but interesting form of algorithm visualization can also be found on video sharing site YouTube, like Quick-sort with Hungarian folk dance [19] or Radix Sort on the Playground [20].

Nowadays, visualizations increasingly often are used not only to help understanding of particular predefined algorithms operation, but also in more general cases connected with programming and system design. The rest of the section is devoted to couple of approaches of this kind.

A powerful way to gain insight into a system operation can lead via moving between levels of abstraction. A systematic approach to interactive visualization is explored deeply in [41]. The approach is illustrated by example of designing the control system for a simple car simulation (Fig. 5).



At each step:
Move forward 1 pixel.
If left of the road, turn right by 2.0°.
If right of the road, turn left by 2.0°.

Fig. 5 Control system for a simple car simulation

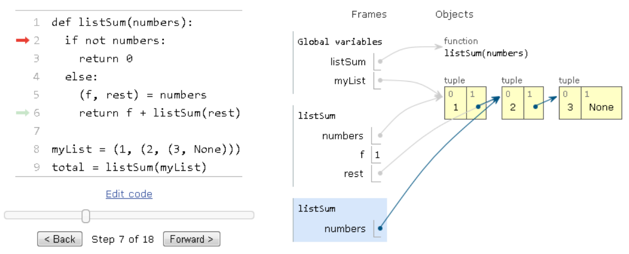
In essay entitled Learnable Programming [42] the following goals for designing a programming system for understanding programs are formulated:

1. to support and encourage powerful ways of thinking
2. to enable programmers to see and understand the execution of their programs

A number of useful hints to fulfill the goals formulated is given within the essay.

An inspiring talk of the author of essays [41] and [42], mentioned above, given at CUSEC 2012 conference is also available [40]. Within the talk, the tools enabling people to understand and create in a visual way are presented.

Another successful example is the Online Python Tutor, which is a free educational tool for learning programming by visualizing code execution. Visualization executions from different areas are prepared, e.g. Basic Examples, Math-Related, User Input, Object-Oriented Programming, Linked Lists (Fig. 6), etc. According to the information available on the project's homepage [43], instructors in over a dozen universities have used it for teaching introductory computer science and programming courses.



```
def listSum(numbers):
  if not numbers:
    return 0
  else:
    (f, rest) = numbers
    return f + listSum(rest)

myList = (1, (2, (3, None)))
total = listSum(myList)
```

Fig. 6 Python Tutor

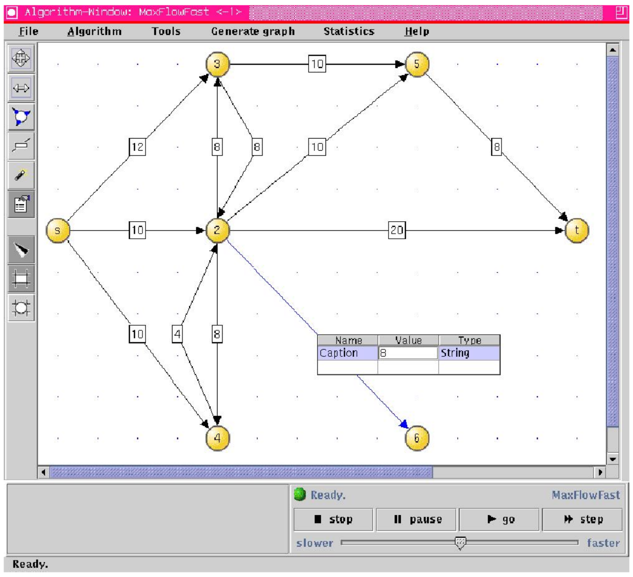
Light Table is an interactive IDE, providing a possibility of running program modifications, embedding different kinds of objects and other features to help understanding how programs really work [44].

3. DOMAIN SPECIFIC VISUALIZATIONS

Searching for adequate abstractions of the specific properties of a particular domain requires the specific knowledge from a given domain (dominant types of objects and operations on them). Sometimes general-purpose algorithm animation systems can be used for producing domain-specific visualizations. However, the effort to do that could be significantly higher, compared to approach when the narrowly-focused systems are used [23].

3.1. Computational Geometry

Finding abstractions of processed data can be quite simple if the data contain position-related information and so could be displayed without extensive transformations. A general tool for interactive visualization of geometric algorithms is GeoWin [26]. System is implemented as a C++ class and thus can be interfaced with algorithmic software libraries such as LEDA [27].



Name	Value	Type
Caption	8	String

Fig. 7 EVEGA visualization environment [28]

Java-based system EVEGA [28] is a visualization environment for graph algorithms (Fig. 7). It provides means to design and edit graphs, display visualizations, and perform algorithm comparisons.

3.2. Concurrent Programs

Within the visualization of concurrent programs a number of specific problems (regarding data collection, data display, program execution, resource allocation, etc.) must be addressed. An overview of systems for parallel program visualization can be found e.g. in [29].

PARADE [30] is an event-based environment, which supports the design and implementation of concurrent and distributed program visualizations. The events can be received via program calls, pipes, or read from a file. A specific component of the system collects events for every single process and provides the user with possibility to manipulate order of the events (chronologically, logically, ...). Animation system POLKA is used within the PARADE for creation of graphical views.

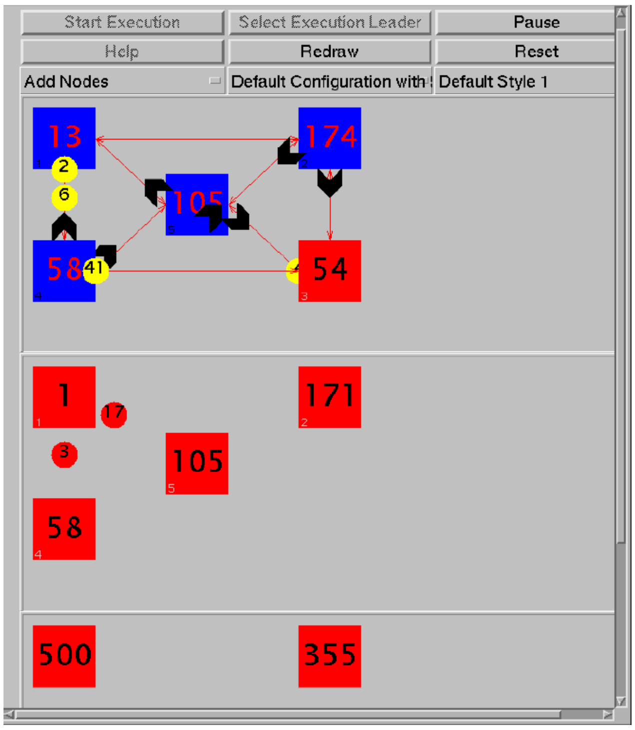
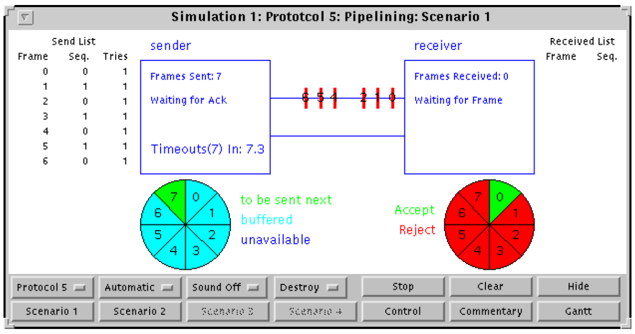


Fig. 8 Visualization of concurrent programs (VADE [31])

Another example of a system from the area of concurrent program visualization is VADE [31]. It is a client-server system for visualization of distributed algorithms (Fig. 8). The system provides libraries for automatic visualization generation, synchronization methods for maintaining consistency as well as support for building web presentations of animations created.

3.3. Real-time Visualizations

In the case of some specific domains (e.g. network protocols) it is strongly required to represent exact timing relations of a given program. An extension of system POLKA with the ability to animate actions with exact timing is system POLKA-RC [32]. It also provides flexible multiprocessor mapping between the program and the visualization, i.e. the program and its animation run as separate processes communicating by sockets.



Send List

Frame	Seq.	Tries
0	0	1
1	1	1
2	0	1
3	1	1
4	0	1
5	1	1
6	0	1

Received List

Frame	Seq.

Fig. 9 JOTSA - simulation of network protocol [33]

Another interesting system here is JOTSA (Java On Time Synchronous Animation) [33], developed using Java and intended for design of interactive algorithm animations, mainly of network protocols (Fig. 9). It supports exact timing within animations, multiple independent synchronized views, display zooming and linking of collections of objects. In addition it provides support for animation of user-defined simulations of network protocols. The project homepage [34] contains a couple of links to documents about JOTSA and applications utilizing the system. Animation examples together with source codes and the application itself with source code are available, too.

3.4. Computational Models

Another area of interest for algorithm animators is the visualization of computational models of formal languages. These models are typically used for mathematical reasoning rather than for programming of real applications. A tool for design and simulation of several kinds of automata (finite state automata, pushdown automata, and Turing machines) and for mutual conversions of language representations is JFLAP (Duke University, Durham, NC, USA) [35]. JFLAP is written in Java and has been used for teaching as well as research purposes.

An interesting tool for teaching subjects from the area of theoretical informatics or design and construction of compilers can be GANIFA [36]. It is the Java applet for visualizations of algorithms from the field of finite automata theory.

3.5. Proof Animations

According to [23], visualization of proofs within the scope of teaching theoretical computer science is relatively unexplored area. As an example in this case can serve the proof animation system SCAPA [37]. The system generates both an HTML document and a Java file from a proof written in LaTeX. The task of visualizer is to extend and modify these files. The proof animation is then created by using an extended version of the LAMBADA tool.

4. ANALYSIS AND DESIGN OF VISUALIZATION SYSTEMS

In a process of development of software system, analysis plays an important role. Specific issues of analytical phase in design of algorithm visualization systems are treated in [11]:

- *User analysis* should provide information concerning the target group of users of the system under design (students, teachers, researchers, developers). By using this information, designer can decide on right content, its organization, depth, and methods of presentation and interaction.
- *Needs analysis* - do we really need the presentation of an algorithm by its visualization, or there exist other effective ways to do so?
- *Task analysis* should give answers to what intended users will do with the tool (creating new animations, interacting with existing visualizations, debugging programs, etc.).
- *Information analysis* - what kind of information will be presented? Do we want to put emphasis on the data structures rather than on the algorithms, display a profile of resource utilization, compare two algorithms?
- *Domain analysis* gives the scope of the algorithm visualization. The scope ranges from general-purpose algorithm visualization systems (able to visualize almost any algorithm, like BALSA, TANGO, Zeus) to specialized algorithm visualization systems. While specialized systems are focused on a certain field of computer science, they simplify the process of creating new visualizations.
- *Resources analysis* includes estimation of development time, the size and skills of development team, and the needs of specialized computer resources.

Within the design phase, special attention is required when choosing the right *specification method* and *techniques for visual representation of information*. In the case of specification method, three main options are available:

- *Predefinition* (hand coded visualization) is mainly used in application-specific visualizations. The method is characterized by fixed (or highly restricted) mapping of the steps of the algorithm to the graphical views. Main advantages include execution speed, simple distribution and resulting visualizations can be very interactive and visually attractive.
- *Annotation* - important steps of an algorithm are annotated with events calling graphical operations from the library (change of position, color, ...). The main advantage of the approach is the ability to define events at any suitable level. Main disadvantage, on the other hand, is the need to access and modify the code of the program.
- *Declaration* - defined are mappings between states of the program and graphical objects. Changes in program state induce changes in its graphical representation (Pavane).

Brief description of some interesting techniques for visual representation of information follows:

- *Default visualizations* - design of appropriate complexity of visual presentations for intended purposes is required.
- *Screen design and multiple views* - for less experienced users, a way for reduction of the information overload is abstraction (complicated parts of the scene can be transformed into simpler items, some phases of an algorithm operation can be omitted, using step-wise refinement, where basic idea is presented first, deeper details later, at user request). Recommended is a screen subdivision into functional areas, containing related type of information. When

multiple views are used, each view provides a particular aspect of an algorithm and these views are conceptually simpler and easier to implement than monolithic views.

- *Color and Sound.* Using of colors calls attention to specific data or information, helps to identify elements of structures, and capture the changes in time. Color generally allows for denser presentation of information and according to [11] it increases appeal, believability, memorability, and comprehensibility. However, some drawbacks or disadvantages are connected with using of colors, too (more complicated design, color perception is subjective, so introducing general rules for color use is very difficult), so it is recommended to be rather conservative when using colors. Sound can help to reduce the visual clutter of current graphic interfaces, but there are also some problems connected with using sound (more than one computer using audio in the same room, more than one view uses audio output, etc.). Similarly as in the case of color, it is recommended to be rather conservative in using sound, too.

- *3D* displays are usually used to represent the 3-dimensional data some algorithms manipulate, or provide an extra dimension to convey more information about two-dimensional data. Some techniques that may be used with 3D (rotation, transparency, navigation, exploration of objects) can be interesting in the area of algorithm visualization.

- *Interaction* makes the visualizations more effective by allowing the user to step through the execution or set its speed, design input data for the algorithm, or control the presentation and amount of information displayed. Allowing the user to simultaneously run several algorithms for comparison purposes can be an interesting feature of a designed system, too.

According to [11], over one hundred of different algorithm visualization tools have been built in the last twenty years, but very few of them were evaluated systematically. Difficulties connected with formal evaluation include: evaluation takes time, which visualization to use for testing, how to measure program understanding, which criteria to use, etc. So far, we have rather informal evidence available that applications of algorithm visualizations are useful.

5. VIZALGO: PLUGIN-BASED ALGORITHM VISUALIZATION PLATFORM

We start with analysis and present the resulting decisions in design of the application later in this section. Descriptions of the structure of prototype implementation, user interface and the method of implementing plugin modules are provided within the section, too.

The decision to work on our own solution, while a lot of existing solutions is available, is driven mainly by the fact, that the application is intended to be used as a support tool within the subject named Data Structures and Algorithms, taught in a bachelor study program at the author's home institution. The selection of topics within the scope of the subject is quite wide and it could probably be changed over the time. To cover the scope of the subject, probably more tools would be used, or quite big interventions to selected tool would be required. Taking also possible changes to the subject's structure into account, we believe it is better to start designing and developing our own solution with emphasis on its extensibility.

5.1. Requirements Analysis

There are some specific issues of analysis and design of algorithm visualization systems, as it was described in section 4 of this paper. At this place we try to give answers at least to most important questions formulated there.

User analysis - the target group of users of the system is mainly formed by students of bachelor study program. According to this fact, methods of presentation and interaction should be rather simple, than too detailed and complicated.

Needs analysis - visualizations here are intended to be an additional form of algorithm presentation, used together with more traditional methods like pseudocodes, static figures, debugging outputs of running program for different inputs, etc.

Task analysis - it is not supposed, that new animations will be created by the users of the system (students). Majority of users will only interact with existing visualizations of predefined algorithms in order to better understand their operation.

Information analysis - within an initial version, emphasis is put on displaying the steps of given algorithm together with corresponding changes in data structures used by the algorithm. More advanced features, like profile of resource utilization, or comparing two algorithms solving the same problem could be attractive future enhancement.

Domain analysis - the system under design could be considered general-purpose algorithm visualization system, rather than specialized one. The reason is quite wide selection of topics within the scope of Data Structures and Algorithms subject, where it is intended to be used as a support tool.

Resources analysis - JSPF (Java Simple Plugin Framework) helps to simplify communication of plugin modules with the main module, so the reduction in development time by using the JSPF is supposed. However, the estimation of total development time has not been performed, because the development of additional plugin modules is considered in a longer period of time in the future.

The most suitable specification method to use within the system seems to be *annotation*. Within the method, important steps of an algorithm (implemented by the plugin module) are annotated with calls of corresponding graphical operations provided by the main module.

From visual representation of information point of view, it is intended to keep the interface of the application rather simple, as it is designed mainly for use by less experienced users (students), as it was mentioned above. Colors are intended to be used in a conservative way and sound output will not be used at all. While the interaction possibility makes visualizations more effective, features like stepping through the algorithm execution, setting its speed, and supplying input data by the user are those to be included in our solution.

5.2. Design Decisions

Having in mind the goals mentioned at the beginning of the paper, including extensibility and portability of the application, and the supposed way of using it, we had to make several decisions. The need for extensibility was explained above, at the beginning of the section 5 of the paper. Application is intended to be used mainly by students, within university laboratories equipped with desktop computers, possibly exploiting different software and hardware platforms.

While Java is a mature platform with properties suitable for project of this type (huge set of available libraries, tools and frameworks, safety, productivity, maintainability, performance, thread management, etc. [45]), the JVM (Java Virtual Machine) is available for many software and hardware platforms and usually it is preinstalled on computers in laboratories, Java seems to be a good candidate for this project.

Another technology to consider in our case, wold be the HTML5 in combination with JavaScript, which is increasingly popular choice for many applications nowadays. It also offers a lot of useful features, like browser UI (user interface) uniformity, speed of development, simplicity, scripts usable without compilation, etc. [45].

If our intention was the web-based application with the support for different types of client devices, the technology of our choice probably would be the HTML5. Instead of this, stand-alone application, possibly preinstalled on computers in laboratories, without the necessity of maintaining dedicated server, in an environment with (even rare) connectivity failures, would be more suitable for us. So at the moment, we believe the better choice for the project is Java, which proved itself a suitable technology for other projects of this type [15–17]. Another important decision to made was the selection of software framework to decrease the programming effort by support the cooperation with plugin modules. After the analysis of available solutions [21] the JSPF was chosen [22]. JSPF was designed to reduce the time of development of plugin-based applications. The framework hides details of component implementation from the programmer, which can concentrate better on the module functionality. Inter-module communication is performed only by using the predefined interface.

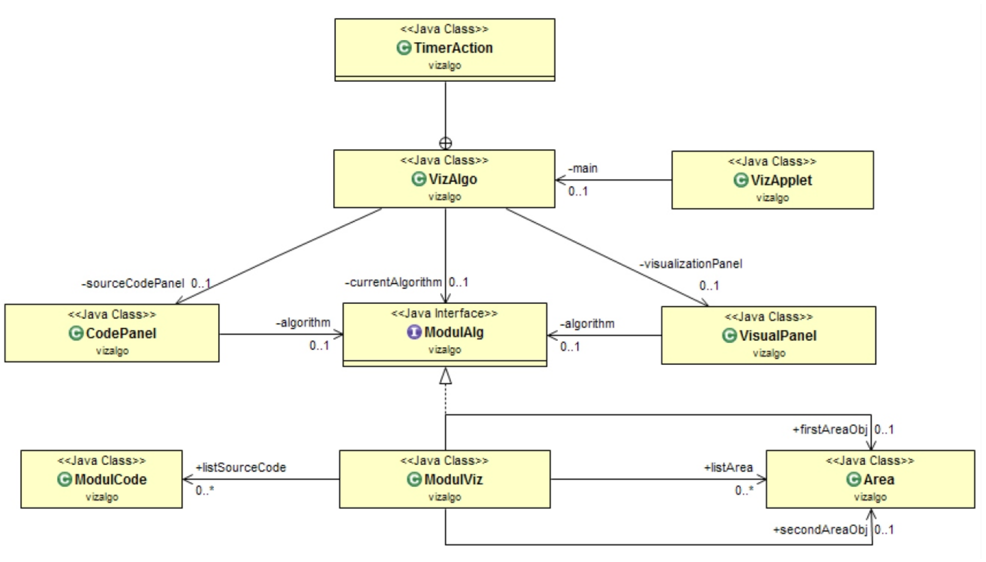


Fig. 10 VizAlgo: The structure of the main module [21]

5.3. The Structure of the Application

The VizAlgo application consists of two cooperating parts - the main module and a set of independent plugin modules. The main module functionality includes support for displaying and controlling the algorithm execution. Independent plugin modules form the second part of the platform. A plugin module thus should contain the algorithm code to be visualized and can utilize the services provided by the main module. A structure of the main module consists of several classes and interfaces as depicted in Fig. 10.

- *VizApplet* class - provides visualization-related services for different components and selection of interface language.

- *VizAlgo* class - provides execution logic, algorithm settings, and animation control. Drives a cooperation of the main module with plugin modules. Creates a list of available algorithms from which the user can select one for visualization.

- the rest of classes (*Area*, *CodePanel*, *ModulCode*, *ModulViz*, *VisualPanel*) and the interface *ModulAlg* provide supporting methods for the application functionality and their detailed description can be found in [21].

5.4. Plugin Modules

Plugin modules within the platform are responsible for implementation and visualization of particular algorithms. The process of plugin module development will be illustrated by example of a simple sorting algorithm BubbleSort.

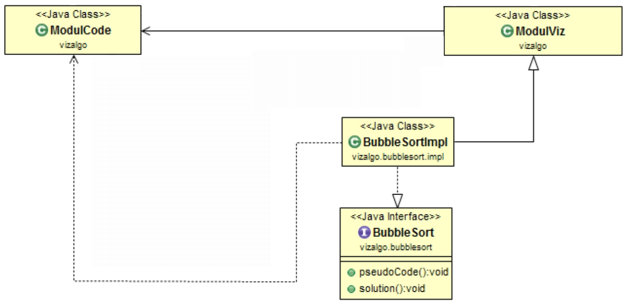


Fig. 11 VizAlgo: The structure of a plugin module [21]

We start by creation of the *BubbleSort* interface and the *BubbleSortImpl* class. The interface created will contain two abstract methods *pseudoCode()* and *solution()*, and these will be overwritten in the *BubbleSortImpl* class later. The class extends the *ModulViz* class and implements interfaces *BubbleSort* and *Runnable* (Fig. 11).

```
public BubbleSortImpl() {
    sort = new Thread(this);
    sort.start();
    pseudoCode();
    }
```

Fig. 12 VizAlgo: The *BubbleSortImpl* method

Within the constructor of the *BubbleSortImpl* class (Fig. 12), creation and execution of a new thread is followed by loading the algorithm pseudocode by calling the *pseudoCode()* method. For storing the pseudocode of algorithm, the method utilizes a list of *ModulCode* type (Fig. 13).

```
public void pseudoCode() {
listSourceCode.add(new ModulCode("for (int pass=1; pass < n; pass++) {"));
listSourceCode.add(new ModulCode("   for (int i=0; i < n-pass; i++) {"));
listSourceCode.add(new ModulCode("       if (x[i] > x[i+1]) {"));
listSourceCode.add(new ModulCode("          int temp = x[i];    "));
listSourceCode.add(new ModulCode("          x[i] = x[i+1];"));
listSourceCode.add(new ModulCode("          x[i+1] = temp;"));
listSourceCode.add(new ModulCode("       }"));
listSourceCode.add(new ModulCode("    }"));
listSourceCode.add(new ModulCode("}"));
}
```

Fig. 13 VizAlgo: The *pseudoCode* method

The *solution()* method contains a code for algorithm execution, in conjunction with managing the highlighting of proper line of pseudo code and visual representation of algorithm operation. A core part of the method for a simple sorting algorithm visualization (Bubble sort) is given in Fig. 14.

```
public void solution() {
    ...
try {
    areaRestore();
    rectangleClear(10, 10, 320, 320);
    loadArea(60, 120);

    for (int pass = 1; pass < listArea.size(); pass++) {
        colorCodeMarker(0);
        colorCodeMarker(1);
        for (int i = 0; i < n - pass; i++) {
            pause();
            colorMarker(i, i + 1, 2, 55, 93);
            if (table[i] > table[(i + 1)]) {
                colorCodeMarker(2);
                colorMarker(i, i + 1, 1, 55, 93);
                colorCodeMarker(3);
                int temp = table[i];
                colorCodeMarker(4);
                table[i] = table[(i + 1)];
                colorCodeMarker(5);
                stringTable[i] = stringTable[(i + 1)];
                table[(i + 1)] = temp;
                stringTable[(i + 1)] = Integer.toString(temp);
                colorCodeMarker(1);
                linePrint(60, 120);
            }
        }
    }

    drawText(algorithmEnd, 40, 320, 2);
    vizalgo.VizAlgo.end = true;

} catch (Exception e) { return; }
}
```

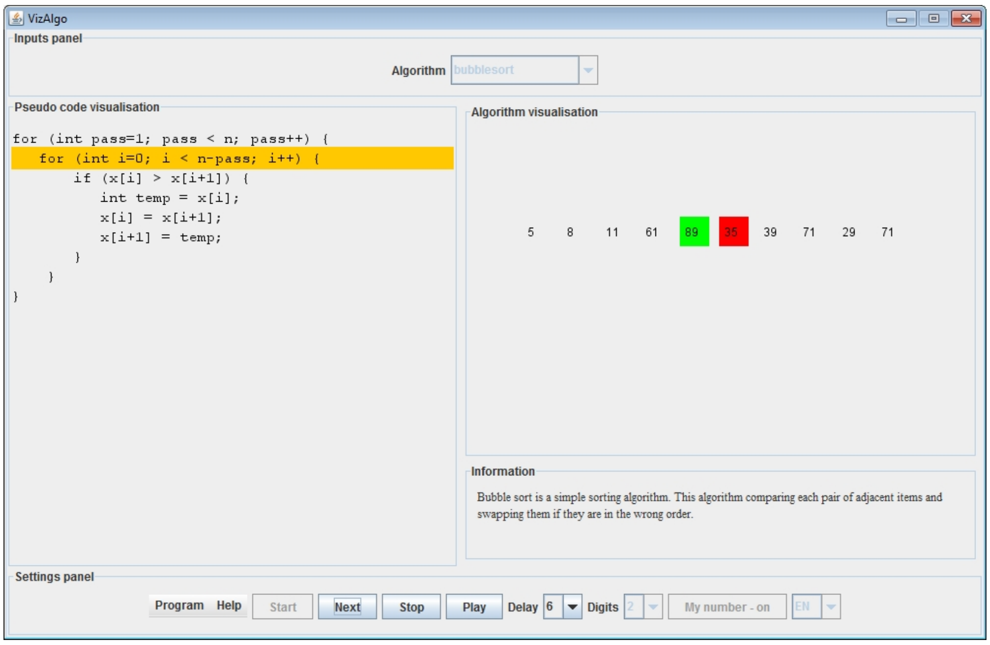
Fig. 14 VizAlgo: The *solution* method

By using methods of the *ModulViz* class within the algorithm execution code, it is possible to perform following visualization-related operations:

- *rectangleClear()* - clearing the content of given rectangular area,
- *loadArea()* - loading the elements with given drawing position,
- *linePrint()* - printing the list of elements,
- *colorCodeMarker()* - highlighting the particular line of source code,
- *areaRestore()* - clearing all the highlightings,
- *colorMarker()* - coloring of particular elements of array,
- *drawText()* - drawing a text at the position specified.

5.5. User Interface

Main window of the application, representing the user interface, can be subdivided into five logical areas: Inputs panel, Pseudo code visualization, Algorithm visualization, Information and Settings panel (Fig. 15).



```
for (int pass=1; pass < n; pass++) {
   for (int i=0; i < n-pass; i++) {
      if (x[i] > x[i+1]) {
         int temp = x[i];
         x[i] = x[i+1];
         x[i+1] = temp;
      }
   }
}
```

Bubble sort is a simple sorting algorithm. This algorithm comparing each pair of adjacent items and swapping them if they are in the wrong order.

Fig. 15 VizAlgo: User Interface

The inputs panel is located at the top of the application's window and it includes a scrolling menu for selection of an algorithm. Pseudo code of the algorithm selected is displayed in pseudo code visualization area at the left side of the window. While the visualization is running, the part of the algorithm actually executed is highlighted here. Algorithm visualization area is located at the right side of the window and it displays the visual representation of the algorithm operation.

Under the algorithm visualization area, the information panel is located, containing the short description of a given algorithm supplied by the author of the plugin module. At the bottom of the window, the settings panel is located. In direction from the left to the right side it contains two menus (*Program* and *Help*) for terminating the program and displaying information about the author of the program and about the program itself respectively.

Next four buttons, in direction from left to right, are responsible for the basic algorithm execution control: *Start* - loading and executing the algorithm selected, *Next* - step by step execution and visualization, *Stop* - terminating the algorithm execution, and *Play* - activating the automatic visualization mode. Next to the right, the speed of the animation, the number of digits of sorted numbers, and the language (Slovak or English) can be set using the corresponding controls. The *My number* button enables the user to enter its own sequence of numbers to sort instead of leaving the program to generate the sequence randomly.

5.6. VizAlgo: the Current State

At the present time, three sorting algorithms are implemented in a form of plugin modules. Two of them, Bubble sort and Selection sort are simple comparison-based sorting algorithms. The third one, Radix sort, takes rather different approach to sort integers with the same number of digits, without direct comparison of sorted elements. Elements here are distributed into buckets (ADT Queue) according to the digit at a particular position, starting from the rightmost one. When the distribution phase by the digit at a given position is finished, contents of buckets are concatenated back to the main bucket and thus forming sequence sorted according to the given digit and digits on the right from it. The algorithm continues by distribution into buckets again, according to higher-order digits, with subsequent concatenation into the main bucket. After processing the sequence of elements according to the highest-order digit, the sequence is sorted.

In addition to the three sorting algorithms mentioned above, visualization of the Stack ADT is implemented and supplied as the next plugin module.

6. CONCLUSIONS

Within the paper we provided a concise overview of development in the area of software visualization and summarized the specific issues of analysis and design of visualization systems. According to our intention to design and develop a plugin-based platform for algorithm visualization with the emphasis on its extensibility and portability, we proposed the core of the platform, together with a couple of plugin modules for examination of the design. The extensibility is ensured by the possibility of independent development of plugin modules, extending the functionality of the application without intervening into the main module. By using the JSPF framework, communication of the plugin modules with the main module was simplified significantly and so the development effort could be reduced, too. Portability was ensured by using the Java programming language, so the application can be used on many software and hardware platforms, where the Java environment is available.

As the main direction of future extensions we recognize the implementation of new plugin modules. By increasing the number of plugin modules available, the usability of the application can be improved, too. We would like to implement not only the next sorting algorithms, for visualization of which the application has shown its capabilities, but also algorithms from other areas. As examples here can serve the visualizations of algorithms on graphs, trees, hash tables, or other data structures. As a next benefit from implementing such extensions the feedback could be obtained, showing us the way in which the main module could be extended or modified to give even better support for visualization of different kinds of algorithms. As a possibility in this direction we can mention the ability to step back or to adjust some of parameters while the visualization is running. Another interesting extension to implement would be a testing module providing automatic knowledge assessment functionality within the system.

ACKNOWLEDGEMENT

This work is the result of the project implementation: Development of the Center of Information and Communication Technologies for Knowledge Systems (ITMS project code: 26220120030) supported by the Research & Development Operational Program funded by the ERDF.

REFERENCES

[1] STASKO, J.: Algorithm Visualization Reflections and Future Directions. The Third International Computing Education Research Workshop ICER'07, Georgia Institute of Technology, Atlanta, GA, USA, 2007.

[2] TUDOREANU, M. E. – WU, R. – HAMILTON-TAYLOR, A. – KRAEMER, E.: Empirical Evidence that Algorithm Animation Promotes Understanding of Distributed Algorithms, IEEE 2002 Symposium on Human Centric Computing Languages and Environments, 2002.

[3] DIEHL, S.: Software visualization: Visualizing the Structure, Behaviour, and Evolution of Software. Springer, New York, 2007, 187 p.

[4] ŠUCHOVÁ, Ž.: Visualization of algorithms and data structures, DCI FEEI TU of Košice, Bachelor thesis, 2010 (in Slovak).

[5] DIEHL, S. (Ed.): Software Visualization, Lecture Notes in Computer Science, Vol.2269, 2002.

[6] BROWN, M. H. – SEDGEWICK, R.: A system for algorithm animation, Proceedings of the 11th annual conference on Computer graphics and interactive techniques, SIGGRAPH'84, 1984.

[7] PRICE, B. A. – BAECKER, R. M. – SMALL, I. S.: A Principled Taxonomy of Software Visualization, Journal of Visual Languages and Computing, Volume 4, Issue 3, pp. 211-266, 1993. Available: http://mcs.open.ac.uk/bp5/papers/1993-JVLC/

[8] BROWN, M. H. – HERSHBERGER, J.: Color and sound in algorithm animation, Computer, 25(12), 1992.

[9] BENTLEY, J. L. – KERNIGHAN, B. W.: A System for Algorithm Animation. Computing Systems, 4(1): 5-30, 1991.

[10] HENRY, R. R. – WHALEY, K. M. – FORSTALL, B.: The University of Washington Illustrating Compiler. In Proceedings of The ACM SIGPLAN'90 Conference on Programming Language Design and Implementation, (pp. 223-233). New York, 1990.

[11] KHURI, S.: Designing Effective Algorithm Visualizations, First International Program Visualization Workshop, ITiCSE, Finland, 2000. Available: http://www.cs.sjsu.edu/~khuri/invited.html

[12] ROMAN, G. C. – COX, K. C. – WILCOX, C. D. – PLUN, J. Y.: Pavane: a System for Declarative Visualization of Concurrent Computations. Journal of Visual Languages and Computing, Vol. 3, pp.161-193, 1992.

[13] RÖßLING, G. – FREISLEBEN, B.: ANIMAL: A system for supporting multiple roles in algorithm animation, Journal of Visual Languages and Computing, 2002.

[14] ANIMAL Home Page, Available: http://www.algoanim.info/AnimalAV/

[15] The JAWAA Homepage, Available: https://www.cs.duke.edu/csed/jawaa2/

[16] Algorithms in Action, Available: http://ww2.cs.mu.oz.au/aia/

[17] TRAKLA2 Software Project, Available: http://www.cse.hut.fi/en/research/SVG/TRAKLA2/

[18] TRAKLA2 video, Available: http://www.cse.hut.fi/en/research/SVG/TRAKLA2/video/

[19] Quick-sort with Hungarian (Küküllomenti legényes) folk dance, YouTube, Available: http://www.youtube.com/watch?v=kDgvnbUIqT4&feature=related

[20] Radix Sort on the Playground, YouTube, Available: http://www.youtube.com/watch?v=ibtN8rY7V5k&feature=related

[21] SAJKO, A.: Algorithm Visualization, DCI FEEI TU of Košice, Bachelor thesis, 2012 (in Slovak).

[22] JSPF: Java Simple Plugin Framework. Available: http://code.google.com/p/jspf/

[23] KERREN, A. – STASKO, J.: Algorithm Animation - Introduction, Software Visualization State of the Art Survey, Springer, LNCS 2269, 2002, Chapter 1, pp. 1-15.

[24] BERGHAMMER, R.: KIEL: A Program for Visualizations of the Evaluation of Functional Programs (in German). In S. Diehl and A. Kerren, editors: Proceedings of the GI-Workshop "Software Visualization" SV2000.

[25] EISENSTADT, M. – BRAYSHAW, M.: The Transparent Prolog Machine (TPM): An Execution Model and Graphical Debugger for Logic Programming. Journal of Logic Programming, 5(4):1-66, 1988.

[26] BÄSKEN, M. – NÄHER, S: GeoWin A Generic Tool for Interactive Visualization of Geometric Algorithms. In Proceedings of Dagstuhl Seminar on Software Visualization, 2001.

[27] LEDA home, Available: http://www.algorithmic-solutions.com/index.htm

[28] KHURI, S. – HOLZAPFEL, K.: EVEGA: An Educational Visualization Environment for Graph Algorithms. In Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE 2001. ACM Press, 2001.

[29] KRAEMER, E.: Visualizing Concurrent Programs. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price, editors, Software Visualization: Programming as a Multimedia Experience, chapter 17, pages 237-256. MIT Press, Cambridge, MA, 1998.

[30] STASKO, J. T.: The PARADE Environment for Visualizing Parallel Program Executions: A Progress Report. Technical Report GIT-GVU-95-03, 1995.

[31] MOSES, Y. – POLUNSKY, Z. – TAL, A. – ULITSKY, L.: Algorithm Visualization For Distributed Environments. In Proceedings of the IEEE Symposium on Information Visualization 1998, pages 71-78, 1998.

[32] STASKO, J. T. – McCRICKARD, D. S.: Real Clock Time Animation Support for Developing Software Visualisations. Australian Computer Journal, 27(4):118-128, 1995.

[33] ROBBINS, S.: The JOTSA Animation Environment. In Proceedings of the 31st Hawaii Int. Conference on Systems Science, pages 655-664, 1998.

[34] JOTSA Home Page, Available: http://vip.cs.utsa.edu/java/jotsahome/

[35] GRAMOND, E. – RODGER, S. H.: Using JFLAP to Interact with Theorems in Automata Theory. SIGCSE Bulletin (ACM Special Interest Group on Computer Science Education), 31, 1999.

[36] DIEHL, S. – KERREN, A. – WELLER, T.: Visual Exploration of Generation Algorithms for Finite Automata. In Implementation and Application of Automata, LNCS 2088, pages 327-328, 2001.

[37] PAPE, C.: Animation of Structured Proofs in Education at University Level (in German). PhD thesis, University of Karlsruhe, Germany, 1999.

[38] Sami Khuri's Visualization and Animation Packages, Available: http://www.cs.sjsu.edu/faculty/khuri/animation.html

[39] STASKO, J. T. – PATTERSON, C.: Understanding and Characterising Program Visualization Systems, Technical Report GIT-GVU-91-17, 1993.

[40] VICTOR, B.: Inventing on Principle. Canadian University Software Engineering Conference, CUSEC 2012. Available: http://vimeo.com/36579366

[41] VICTOR, B.: Up and Down the Ladder of Abstraction. October, 2011. Available: http://worrydream.com/LadderOfAbstraction/

[42] VICTOR, B.: Learnable Programming. September, 2012. Available: http://worrydream.com/LearnableProgramming/

[43] GUO, P.: Online Python Tutor. 2010-2013. Available: http://www.pythontutor.com/

[44] GRANGER, C.: Light Table, 2013. Available: http://www.chris-granger.com/lighttable/

[45] TRAVERSAT, B.: HTML5/JavaScript and Java: The Facts and the Myths, JavaOne annual conference, Oracle, 2011. Available: http://www.oracle.com/javaone/lad-en/session-presentations/clientside/24821-enok-1439095.pdf

Received March 18, 2013, accepted June 17, 2013

BIOGRAPHY

Slavomír Šimoňák was born in 1974. In 1998 he graduated from the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University of Košice and defended his PhD thesis titled 'Formal Method Integration Based on Transformation of Petri Nets and Process Algebras' in 2003. Currently he works as an assistant professor at the Department of Computers and Informatics. His research interests include formal methods integration and application, algorithms and data structures, machine-oriented languages, and computer emulation.