New Graphic Polyglot Concept of Programming

Igor Velbitskiy
Glushkov's Fund
Kiev, Ukraine
Email: ivelbit@gmail.com

Gary Ushakov
IBus Datasoft, LLC
Bridgewater, NJ, USA
igor.ushakoff@gmail.com

Abstract

For the first time since 1947 it is proposed to use a simpler and mathematically more rigorous concept of programming with the graphs loaded only through horizontal arcs. Such graph is a polyglot, it has ISO 8631:1989 certificate, and can be used along the entire lifecycle of programming process and working on a computer. The new concept has HUNDREDS(!) times better characteristics from the standpoint of program compactness, memory use, promptness of entering and execution. Processes of designing the algorithms and software, processes of programming, evidence of correctness and self-documenting of software projects are significantly simplified, improved and accelerated. For the first time programming receives a rigorous mathematical basis (culture), wherein natural succession of the best practices in programming (OOP, AOP, WEB, CLOUD etc.) and, above all, continuity of the program libraries are ensured. The larger and the more logically complex is the program project, the greater is the effect of applying the new polyglot- concept. The new concept is so simple that it makes it possible to program for ANYONE, not just for programmers. This article describes the history of development and proving of the new polyglot concept of programming, its description, advantages, implemented graphical programming environment, and perspectives for application.

Keywords – *graph, RR*-schemes, visualization, simplicity, compactness, color, optimizing, quick entry, drawing, polyglot, proof of correctness, network graph, 3D programming, flow-charts, UML, Google Blockly.*

Introduction

Development of programming in graphs started in the 70s' with the development of large real-time control systems [1] and understanding of importance of formal documentation of their development process in order to facilitate corrections and improvements, and also with the works of Dijkstra [2], who was the first one to demonstrate mathematical non-rigorousness and redundancy of the conventional programming concept. Way back in the 1960's I introduced for myself the term "technology" in programming as an intuitive perception of the fact that the main thing is not only the language (at that time it was Algol-60) but HOW to use it and write in that language in a correct and cultural way. Now, having summarized my 15-year experience in developing automated technological complexes for control systems for all major missiles of the former Soviet Union it became obvious that the technology problems (and programming process itself) should be solved only by changing the basic concepts of programming [3,4]. As a result, for the first time after 1947 a new and very simple concept of graphical programming with R-schemes was formed and its comprehensive analysis and pilot implementation (!) was conducted. This concept uses graphs loaded only through the horizontal arcs and no complex profiles (shapes). Unlike many known graphical concepts (graphs loaded through the tops) such as: Flow-chart, UML-diagrams, Workflow Simulink, Google Blokly, etc. the new concept has many unique advantages. For example, it introduces evidence style in programming and possesses better (more simpler) principles of designing, debugging, and documenting. The new concept does not depend on the language of programming; it is a polyglot and complies with strict mathematical principles of information presentation and processing, which approximate to the principles of operation of the brain. As a result, it DOES NOT require a special programming language, and the graphical image (drawing) of the project COINCIDES with the product itself – the program and network graph of its development. It coincides throughout the entire life cycle of all works on the computer both for the developers and for the users (!). This DOES NOT exist either in the conventional programming or even in the industry, where, for instance, a drawing of a vehicle differs from the vehicle itself and the network graph for its development. This new graphical program is HUNDREDS of times smaller and can be entered (developed, run) a lot faster. The principles of the new concept are so simple and natural to a man that for the first time programming will become available to EVERYONE (and not just for programmers) and will be an element of universal literacy and culture of the society. The single disadvantage of the new concept is ONLY the inertia of thinking and habits of the person, e.i. the programmer.

In the new concept, **ALL** the conventional machine-oriented statements (such as **if-then**, **else**, **for**, **while**,

0730-3157/16 $31.00 © 2016 IEEE
DOI 10.1109/COMPSAC.2016.160

IEEE computer society

goto, **labels** and brackets like **begin-end, {-},** etc.) are *excluded* from programming. **They are outdated**. There are too many of them for a man, they are complicated, empirical, have low capacity, and provide a primitive handicraft technology of programming. People are spending too many efforts *to compensate for* such drawbacks by means of creating many special languages, methods and programming environments that "supposedly make things easier", but in fact, they alienate specialists and make the programming extremely complex, non-evolutional and inaccessible for **everyone**. For two small programs of 24 (452) lines in C++ cited in this article the equivalent graphical program of the new concept occupies 4 (10) lines. The graphical program makes unnecessary more than half of the characters =142-63%(=4364-73%) in the C++ program, which are keywords, block brackets, characters line feed, indents, etc. Entering of one statement in C++ takes an average of 20 (29) keyboard clicks, and of a graphical statement (arc) - 0.8 (0.21), which is 25 (138) times less (faster), etc. The current article (635 lines) describes 9 examples, 140 (22%) of R-scheme lines, which in the conventional concept occupy a total of about 43,000 lines in C++ and Delphi, but the proposed concept is just 0,3% of that and it is in 307 times smaller.

Essence of Programming in Graphs

The concept of programming in graphs (ISO 8631:1989 [5]) proposes using only ***one*** horizontal arc, which is the R-scheme (see Fig. 1) throughout the entire life cycle of the program instead of the conventional type statements such as **if**, **for**, **goto,** etc. (**total** about 10). This R-scheme has the *Condition* always written at the top, and at the bottom - the *Actions* to be executed if the Condition is "true". Any language, including English, Russian, Chinese etc., mathematical and any programming languages can be used for writing the condition and actions in one or several lines. This is a polyglot principle.

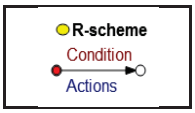

Figure 1. R-scheme: one horizontal arc to the right or to left.

Any number of alternative arcs may radiate to the right and/or to the left from one node, see Fig. 2-12. These arcs are viewed (analyzed) sequentially from top downward until the true Condition appears. Then the appropriate Actions are executed and transition on the arc arrow into the new program (graph) state is performed. If the Condition is "false" and the last arc is reached, the transition (without performing the Actions) is done on the arrow to the next state of the graph.

The example of defining a new statement is given in Fig. 2. Its R-scheme of is given in Fig. 2a, its R*

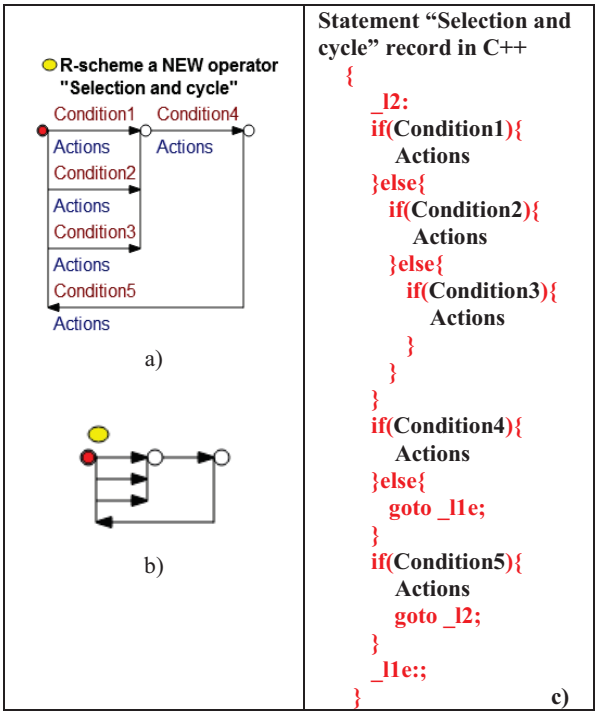

Figure 2. Recording of a new statement of "Selection and Cycle" in RR*-schemes and in C++. Red symbols in C++ are superfluous for R-schemes

mathematical R-scheme without implementation details (without entries on the arcs) is given in Fig. 2b, and Fig. 2c demonstrates the equivalent conventional entry in C++, where the *superfluous* symbols for the R-scheme in Fig. 2a are shown in red color. There are 79 of those superfluous symbols in Fig. 2c. Their number including the indents and line feed is **142 (63%)**. The new statement recording in Fig. 2a is **2** times and statement recording in Fig. 2b is **6** times smaller compared to their recording in C++ given in Fig. 2c. Entering of one statement in C++ (Fig. 2c) on average requires 20(=142/7) button clicks of the keyboard buttons, while entering of an R-scheme graphical statement or arc (Fig. 2a) requires 0.8(=4/5) clicks, which is **25**(=20/0.8) times less and faster.Entering of an R-scheme (Fig. 9) is **138**(=29/0,21) times faster, where 29(=**4364**/150) is the number of symbols for entering one equivalent statement

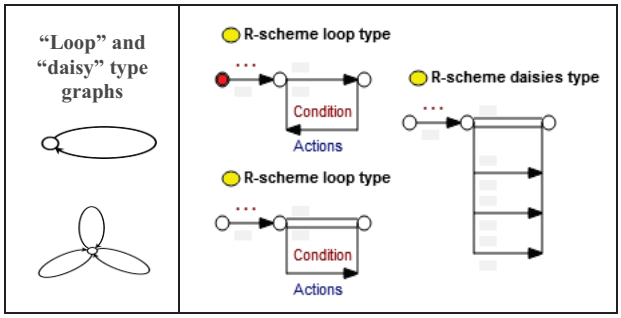

Figure 3. "Loop" and "Daisy" R-schemes

in C++, and 0.21(=21/100) is the number of clicks for entering one graphical statement (arc) in Fig. 9. R-schemes of the "loop" and "daisy" types are given in Fig. 3. The "equal mark" arc between the nodes combines them into one appropriate graph node.

Keywords such as **real**, **procedure**, **function**, **form,** etc., which are always true and set a new understanding and usage of all the R-scheme elements can be used on the top of the arc. For example, they set a single (Fig. 4a), i.e. a mandatory *parallel* execution of **all** the arcs radiating from the node in conformity with the keyword on the arc. The semantics of this problem are displayed in standard notations in Fig. 4c. Many descriptions of variables and appropriate keywords can be excluded from the new concept because they are easily deduced (understood) from the record layout of these variables on the R-scheme arcs like in mathematics. It even more improves the visualization and compactness of RR*-scheme recording.

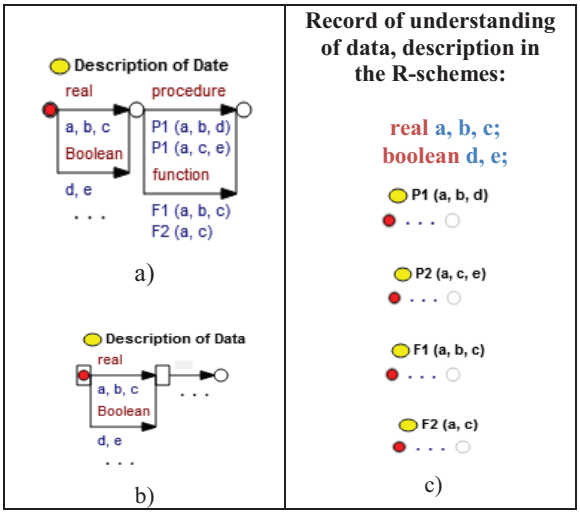

Figure 4. Definition of Data, Procedures and Functions

The graph nodes are not assigned names, but they may have different *configuration* and *color* for implementation of signal lights in the program, definition of &-arcs, 3D programming, etc. For example, the red-colored node can always define the project start; rectangles for recording of &-arcs executed in parallel are used in Fig. 4b, 6, and 7; and the principle of 3D and

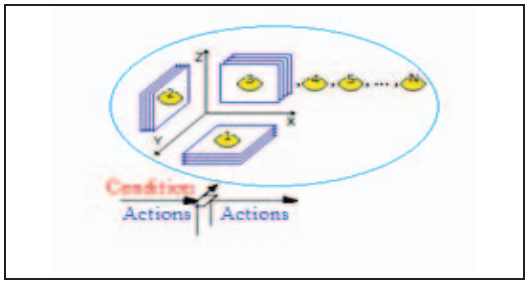

Figure 5. Organization of 3D programming and motivation of decisions

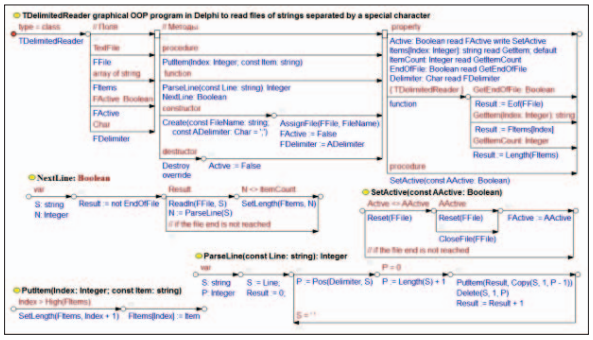

Figure 6. R-scheme of the OOD program is 7 times **smaller** compared to the conventional recording in Delphi [9].

documenting of motivation of the decisions is depicted as a parallelogram type node in Fig. 5.

Such graph is assigned a name written on the top near the yellow ellipse, see Fig. 1-4, 6, 8, 10-12. The name can be with or without parameters, see Fig. 6. It is possible to program with any number of the graphs interrelated by name, see Fig. 6. Unlike the conventional program recording such R graph (Fig. 6) has a record R* (Fig. 7) of the mathematical abstraction of the program without entries on the arcs (without

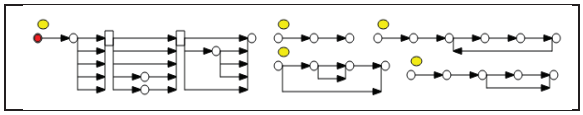

Figure 7. R*-scheme of the program recording without implementation details is 35.7 times more compact compared with Delphi

implementation details). This R* record enables setting the program models and their mathematical studies as well as identical transformations and optimization by various parameters, such as time, memory, and also classification for the new type of graphical software archives, etc. It is more compact and significantly

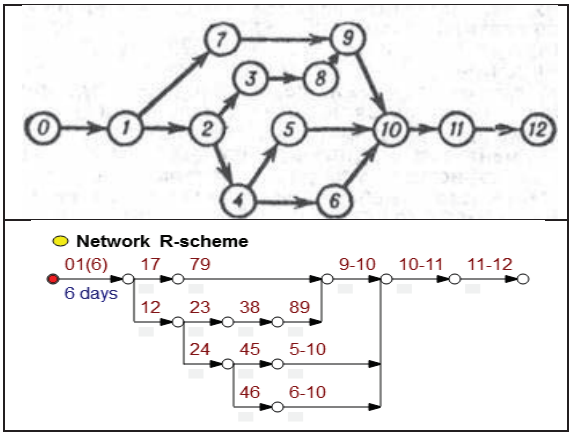

Figure 8. Recording of the network diagram in R-schemes

simplifies program designing. For example, for Fig. 6 the compactness is R=7, and for Fig. 7 the compactness is R*=**35.7** times better compared to recording of this program in the conventional text in Delphi [9], also see Fig. 2abc and 8. As of today, the maximum compactness of *graphic shells* for real programs is R = **19**, and R* = **130**. The larger and more logically complex is the program, the better is its compactness,see Fig. 2a-2b and Fig. 6-7.

The draft program drawing coincides with the product, its documentation, and network development schedule throughout its entire lifecycle, see Fig. 6-8. For the first time the supervisor of works has the possibility to write the executor's name and estimated time for the work completion in special brackets on the R-scheme arc next to each piece of work and **control** the quality of the performed work.

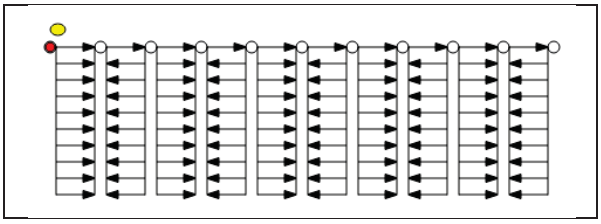

Figure 9. Entry of an R-scheme (program) of 10x10 =100 arcs in **21** clicks (**138** times faster), and the compactness is R=**15**, R*=**45** compared to C++.

Each horizontal arc of the graph is entered by one mouse or keyboard click or by finger on the touch screen. The vertical arcs and graph nodes are not entered into the computer but the graphics editor draws them automatically. Several new horizontal arcs can be entered with a single click. For example, in order to program the R-scheme in Fig. 2b, it is necessary to make 4 (for 5 arcs) clicks of the left mouse button or the keyboard, in Fig. 3a

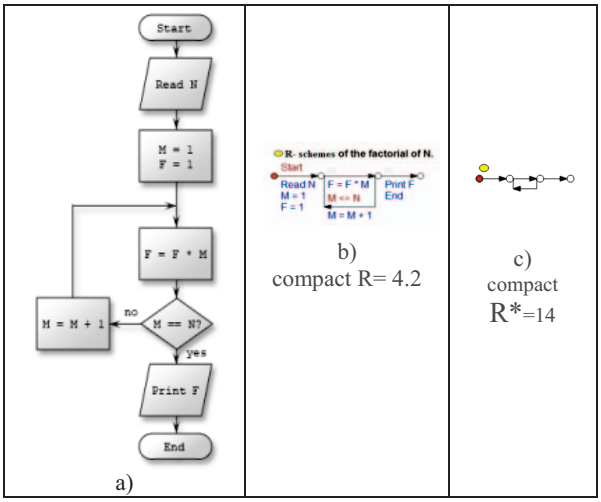

Figure 10. Example of recording the computation algorithm for the factorial of N in flow-charts and RR*-schemes (b, c)

– 3(4), in Fig. 6 – 25 (39), and in Fig. 10c – 2(4). As a result, in average less than one click (up to e=0.01, 100 times less) of the mouse or the keyboard is required to set one arc of the graph, for example, Fig. 9 = 21 (=10x10=100, e=21/100=**0.21**). The R-scheme (program) of 100x100 arcs has e=**0.0302**, R=**133,** and R*=**400** and contains **578** times less symbols; therefore its entry into the computer is **578** times faster compared to the equivalent program in C++, etc. No language is required for setting the arcs, so the new concept is a polyglot.

A man is always trying to find a graphical image for any of his actions. A huge amount of graphical methods for program recording are available, including Flow-charts, UML, Workflow, Simulink, SDL, Dragon, Google Blokly [10], etc. As a rule, these are graphs loaded through the nodes, Fig. 10a. R-schemes are graphs loaded through the arcs, Fig.10bc, and Fig. 11,12, which have considerable quantative and qualitative advantages compared to the conventional ones, Fig. 10a and 10bc, Fig. 11,12 (top) and (bottom). Unlike the conventional Flow-charts, UML, etc., R-schemes are *executable* on the computer throughout the lifecycle of any works; they are more visual, compact and simple.

Thus, the single graphical shell is proposed for all the languages, including the natural, mathematical and programming ones. Recording, translation and interpretation of programs and projects become easier in that shell. It is for the first time when a program (R- scheme) becomes an object of mathematics for a human being(!), and not only for the computer.

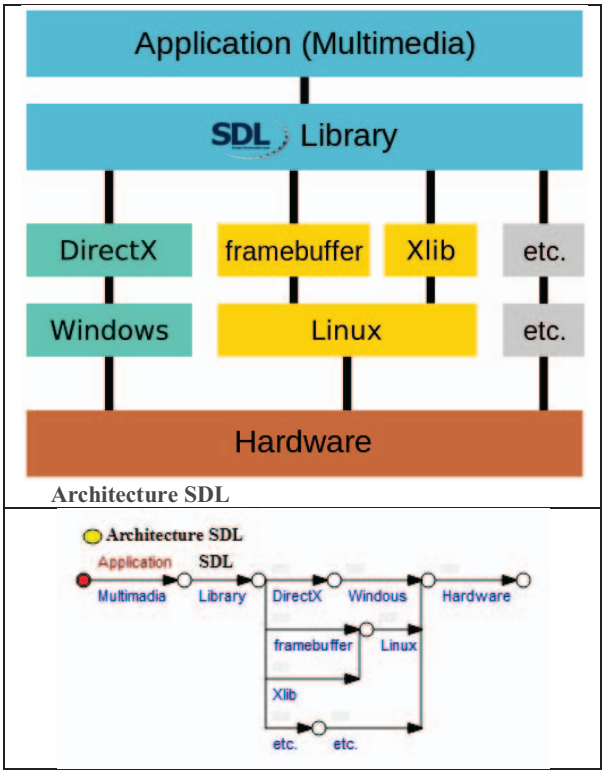

Figure 11. Workflow of SDL architecture

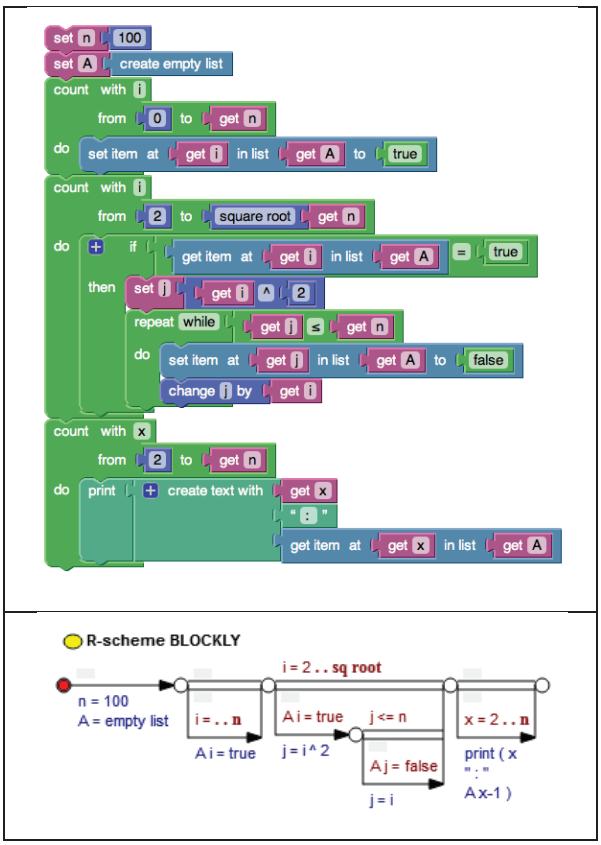

Figure 12. Example of program recording in visual programming language Google Blockly [10] and in the R-scheme. Compactness R=3, R*=14

RR*-schemes are ideal for *reverse* translation of programs (which is not found in the conventional programming) in any language into a new single graphical shell as standard. As a result the program being a "thing in itself", which is understood only by its author (who is not always available), becomes available for the development and improvement by others and the whole programming process becomes evolutionary with preservation and accumulation of experience.

The form of graphical recording has substantial reserves for expansion and development in the future. Different colors can be used for nodes, arcs, and records. The arc's nodes and arrows can have different configuration, the arcs can be dual, wavy, dashed, etc.

Technology of Designing and Proof of Program Correctness

The conventional design concept of an algorithm (program) is based on construction of a computing scheme for solution of the initial problem using approximately ten fixed machine-oriented statements. To this end the original problem is converted for recording using these statements. It is a well-known complex and multi-step process. As a result of this process, the initial problem is transformed "beyond recognition", the connection and motivation of decisions taken are lost in the course of transformation of the initial problem, and understanding of "how all that works" in the whole is lost. Numerous methods have been invented to simplify this process, which make programming even more complicated and confusing.

The new design concept is based on construction of the logical scheme of the initial problem in the terms and words of the initial program. The "step by step from logic" technique is used for recording of the initial statement of the problem in the single graphical shell of R-schemes and in the same language as the initial problem description. This process identifies and clarifies all the basic inaccuracies in the formulation of the problem. As a result, the logical R-scheme of the problem becomes equally clear to both the Client and the Contractor and is approved by both of them. Subsequently, elaboration of the algorithm and the program is carried out with the formal methods of mathematical derivation, which eventually will serve both as a proof of the program correctness and its development process. The resulting documentation is different from the conventional one, as it retains the motivation of decisions, and it is more compact and self-documenting. The graphical environment monitors and directs a person in such a way that his formal transformations exclude all the ambiguities from the original logical scheme and transform it into a language natural to a man (as a rule, into the graphical language of simple mathematics) and subsequently, automatically, into a computer language. The language natural to a man is being continuously improved (evolves along with a man) and approaches the professional language of the relevant team of program developers. It is important that the language of R-schemes of such description is **uniform (!)** for the computer, the Client, and for all the performers of the project and its users throughout its life cycle.

Implementation of Graphical Programming Environment

At present, the graphical environment of program development *has been implemented* (lab version, programmers: **Anton Khodakovskiy, A. Gubov**), which includes the entry system for R-schemes and any texts on the arcs in any language; creation and memorization of the project tree; graphics editor; converter of R-schemes into R* and reverse; compiler of RR* in C++, etc. This graphical environment has been developed as a REditor plug-in for *Qt Creator*. It consists of 5 areas, which divide and form the display screen. About 90% of the central part of the screen is used for 1) The work field, which is used for development of all the project's R-schemes. Three lines at the top are used for arrangement of 2) Menu for establishment of the environment architecture, 3) Toolbar (with 14 graphical

icons), and 4) Panel of open (unlimited number) names of work fields of the R-schemes. 5) The last area occupies a narrow strip (5%) of the screen on the left for the R-schemes *Tree*. Therefore, the implemented graphical environment has summarized the available experience of programming in graphs, and it is considered complete (80% according to our estimates) for development of the commercial version on its basis.

One of the *independent* users said the following about this new graphical environment**:** "After two weeks of drawing the schemes in the editor I resolved the problem, which I would have never been able to solve without R-schemes**…** I see R-schemes as a useful and wonderful tool not only for programming, but also elsewhere if there is a need for understanding the logic of interaction between the parts of any problem or work. They demonstrate obvious advantages compared to the well-known flow-charts, UML, etc. R-schemes are executable on a computer along the entire lifecycle of any work, and they are more visual, compact, and not overloaded with superfluous details (figures). They are suitable for visualization of nonlinear algorithms (e.g. C++ classes) or data structures. Therefore, I absolutely refuse to understand why this tool does not enjoy a ***tremendous*** popularity", V. Bushkevich, 04.05.2015.

Conclusions

A new mathematical concept (culture) of programming is proposed, which is interesting due to its advantages, simplicity, humanity and possibility of adopting by **everyone,** not only by the computer programmers. This culture allows introducing a proof style into the professional programming as well as the long-developed mathematical methods of analysis and synthesis, and the new technology of industrial development of large programs. Taking into account an extraordinary (1-2 orders of magnitude greater) compactness and ease of data entering especially with touch screens the new concept has great prospects for effective application in small devices of mass use (iPhone, iPad).

The new proposals are particularly effective for the elementary training in programming due to their simplicity and visualization. Therefore, they can be included into the system of compulsory education of any specialists. **EVERYONE** should be able to program, and programming should become a part of the universal literacy and culture of the society. The new culture of programming has great promises for the future development of a human-computer society and its *evolution together* with a man. A computer with new architecture and new concept of programming implanted into a human being will control, together with the brain, the physiology of a specific person, compensating its deficiencies from birth and providing treatment through life without using the conventional drugs. And it will provide more effective and strong control than well-known «Pharaoh cylinders».

It is common knowledge that the ***links*** between neurons and regions play the major role in the human brain. ***Arcs*** between the nodes play a similar role in the R-schemes. The number of arcs, their direction and 3D-orientation between different nodes and R-schemes are infinite. The node can remember the number of accesses and have a sensitivity threshold. Possibility is considered for automatic generating of programs based on recorded data generated under the same concept. Nothing like thatcan be found in the contemporary programming. All mentioned above and huge potential of the modern microelectronics and memory devices urges us to analyze *today* the main drawback of our computers since 1947, which is the complex and primitive organization of the programming process. The time has come to analyze and summarize the experience, select all the best and develop both the SOFTWARE and the COMPUTER of principally new **brain-like** generation in this millennium. We estimate that the development of this project will take 6 months and additional 6 months will be required for its implementation and marketing. However, these timelines can be significantly shortened taking account of the current version of R-schemes implementation, clearly defined objectives, and accumulated global experience in program development and step-by-step implementation.

References

[1] "Sergeev V.G.– Chief designer of control systems of rocket and space complexes.", Ukraine, Kharkiv, 2014, 448 p.

[2] Dijkstra, E., «Letters to the editor: go to statement considered harmful», *Communications of the ACM 11 (3)*, doi: 10.1145/362929.362947.ISSN0001-0782, 1968, pp. 147–148.

[3] V.M. Glushkov, I.V. Velbitskiy, «Programming technology and problems of its automation», USIM, №6, 1976, pp. 75-93.

[4] I.V. Velbitskiy, "Programming technology". Tekhnika, Ukraine, 1984, 279p.

[5] «Information technology, Programme constructs and convention for their Representation», International standard ISO/IEC 8631, 1989.

[6] W.K. McHenry, «Technology: Soviet visual programming». *Journal of Visual Languages and Computing,* v.1, № 2, 1990.

[7] I.V.Velbitskiy, «Graphical Programming and Program Correctness Proof», CSIT-13 *IEEE*-DOI: 10.1109 /6710368, 2013, pp. 85-89.

[8] I.V.Velbitskiy, «Graphical programming of new generation. Single universal graphical shell for all languages» Global IT, Las Vegas, DOI 10.1109/CSI Technol.7358261, 2015, pp.111-115.

[9] Valvachev, A.N., http://www.rsdn.ru/article/Delphi/

[10] The visual Google Blockly, *habrahabr.ru*