UDC 004.042

Oleg N. Paulin[1], Doctor of Technical Sciences, Associate Professor of the System Software Department, E-mail: paolenic@yandex.ru, ORCID: https://orcid.org/0000-0002-2210-8317
Nataliia O. Komleva[1], Candidate of Technical Sciences, Associate Professor of the System Software Department, E-mail: nkomlevaya@gmail.com, ORCID: https://orcid.org/0000-0001-9627-8530
Stanislav U. Marulin[1], Candidate of Technical Sciences, Associate Professor of the System Software Department, E-mail: stanislavmaru@gmail.com, ORCID: https://orcid.org/0000-0003-0755-0104
Anatolii O. Nikolenko[1], Candidate of Technical Sciences, Associate Professor of the Information Systems Department, E-mail: anatolyn@ukr.net, ORCID: http://orcid.org/0000-0002-9849-1797
[1]Odessa National Polytechnic University, Shevchenko Avenue, 1, Odessa, Ukraine, 65044

METHOD FOR CONSTRUCTING THE MODEL OF COMPUTING PROCESS BASED ON PETRI NET

Abstract. *The aim of the work is to improve the quality of the computational process that solves the problem, due to its modeling and debugging based on the Petri net. The quality of the computational process is understood as the absence of errors (looping, paralysis, unreliability of some fragment, etc.) and its optimization according to the criterion of minimum complexity.*

The new approach to the analysis of the computational process, based on preliminary modeling by Petri nets of both fragments of computational processes and complete computational processes, is proposed. This will reveal many errors at the stage of modeling the computational process. The computational process is considered as a set of macrooperations, which are functionally, completed operations of various hierarchical levels. To locate macrooperations in a computational process, it is decomposed into elementary (basic) computational constructions. A statement that any computing process can be constructed on the basis of a relatively small number of macrooperations is formulated. To implement the new approach, the task of developing a method for constructing a Petri net according to a given computational process is formulated and solved. The essence of the proposed method consists in dividing the computational process into macrooperations, building a Petri net fragment for each macrooperation, modeling all fragments, assembling a complete Petri net from network fragments and modeling it. To implement the method, a procedure for constructing a computational process model is being developed. The stages of this procedure are described: decomposition of the computational process into macrooperations according to the proposed rules, translation of macrooperations into fragments of the Petri net and their modeling, collection of the complete Petri net by the proposed rules, and modeling the resulting Petri net. The results of the implementation of all stages of the procedure are recorded in the library, the aim of which is the accumulation of knowledge about the computational processes corresponding to them Petri nets and modeling results. This allows us to simplify the process of modeling a new computing process through the use of already debugged fragments. If the computational process contains errors or is not optimal, it is corrected, which allows to improve its quality according to the above criteria. By the example of sorting by inserts, the correctness of the operation of the constructed Petri net using the declared method is experimentally confirmed.

Keywords: *computational process; macrooperation; method; procedure; Petri net; modeling; library*

Introduction

In general terms, a computing process is a sequence of time-ordered operations and procedures of varying degrees of complexity. The computational process (CP) should underlie the software and largely determine its quality, while for one CP, you can consider many software implementations, taking into account different technologies and programming languages. To date, there are no quality standards for CP. Traditionally; the quality of the CP is determined by the absence of errors and a certain level of optimality. Among the types of possible errors, a loop, a hang, and also an emergency stop of a CP during fatal errors, for example, division by zero, are distinguished. CP optimization can be carried out according to different criteria; the criterion of mini-mum complexity is most often used.

Indirectly, the quality of a particular CP can be evaluated by analyzing software developed on its basis for compliance with requirements in accordance with ISO 12207, ISO 9000, CMM (Capability Maturity Model). Failure to comply with such requirements entails the need for error correction, reengineering and re-testing of program code, which requires additional resources, including time, and is much more expensive than fixing errors at the initial stage of development. However, in practice, the development of a high-quality CP as a basis for high-quality software is not always given due attention, which causes the described problems. That is why the construction of a high-quality CP is actual.

An effective way to improve CP is its modeling, while using different approaches, taking into account, among others, the apparatus of Petri nets. The work offers the new approach, which involves decomposing the CP into elementary computing structures, modeling and debugging these

DOI: 10.15276/aait.04.2019.1
ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

structures by Petri nets, the final assembly of separate structures into a complete net of whole CP and its modeling. Applying of this approach allows to improve the quality of the computing process, and this, as a result, will lead to quality improvement of its software implementations.

Analysis of recent publications and formulation of the problem

With all the variety of computational processes, there is the possibility of formalizing them, which makes it possible to choose an apparatus for describing and further analyzing these processes. At one time, much attention was paid to the automatic approach to the representation of processes and their software implementations [1]. In the automated approach, two of the most developed ways can be distinguished: SWITCH-technology and SM-technology (State machine) of development [2]. These ways are distinguished by the implementation of the logic of automatic programs. However, such an approach is a programming paradigm, since it is a program or its fragment that is interpreted as a model of some formal automaton. At the same time, it is more expedient to associate computational constructions with the automaton. In addition, the disadvantage of the automaton approach is the small computing power expressed by the automaton language, because the automaton does not reflect a more complex concept than "operation", that is, the concept of "event".

The above disadvantages are deprived of Petri nets, which have a significantly greater expressiveness of their language. As shown in [3], they occupy an intermediate position between the state machine and the Turing machine. Petri nets reflect the interrelation "event – condition", which made it possible to widely use them for modeling [4].

Thus, the publications [5-6] show the construction of Petri nets for a wide range of different algorithmic processes. In [7] it is shown how the fulfillment of the conditions and restrictions imposed on such processes affects the characteristics of the constructed Petri nets. In the work [8], the relationship between the complexity criteria of the original algorithms and the sizes of Petri nets constructed from them was studied. The article [9] describes the optimization mechanisms for Petri nets using the example of minimizing the number of places using special heuristics. A number of studies expand the scope of Petri nets to more complex multilevel processes, in which algorithmic processes as such are fragmented, with the construction of Petri nets for such fragments [10-12].

As can be seen from the analysis, despite a fairly wide range of work related to computational processes [13; 14], the authors did not attempt to adjust the initial CP or algorithm using Petri nets built on their basis as a reverse control action to improve the quality of the CP.

The aim of the work is to improve the quality of the computing process by eliminating errors and optimizing it by modeling and debugging based on the Petri net of the computing process and its fragments.

To achieve this aim, the following **tasks** must be solved:

– development the method for constructing a CP model;

– development the procedure that implements this method;

– development the rules for converting macrooperations into fragments of Petri nets and the further assembly of a complete Petri net.

1. Brief description of Petri nets

Petri net is a bipartite graph with set of vertices $P \bigcup T$, containing two kinds of vertices: transition $p \in P$ (depicted by a dash) and a position $t \in T$ (in some sources, a place; depicted by a circle), and, accordingly, two types of relation: $p \to t$ (triggered transition gives permission to fulfill the condition) and $t \to p$ (implemented condition allows the transition to trigger). A transition identifies an event, and a position identifies a condition. Transitions and positions alternate. Transitions and positions can have several inputs and outputs.

A dot (chip) in the position indicates the possibility of fulfilling the condition; the number of points in the position means the number of possible conditions. The allocation of points by position (network marking) characterizes the state of the net.

A markup M of the net N is represented as a vector of numbers m_i, denoting the number of points in the position p_i, $m_i = M(p_i)$: $M = (M(\mathrm{p}_1), M(\mathrm{p}_2), ..., M(p_n))$.

During the operation of the Petri net, the points migrate over the net. When a transition is triggered, one point is taken from the previous positions, and one point is added to each output position. Petri net has *reachability* and *survivability* properties. A markup M' is achievable in the network from the markup M as a result of the *sequence of operations* of the transitions $\tau = t_1, t_{2,...,} t_l$, if there is a sequence of consecutive markings $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots \xrightarrow{t_l} M'$. A markup M is reachable on the net N, if there exists τ such that $M_0 \xrightarrow{\tau} M'$, where M_0 is an initial markup.

Let $R(N)$ be the set of all markups reachable in the net N. Then the transition t is reachable from the markup M in the net N, if markup M' exists in $R(N)$

and there is such τ, that $M_0 \xrightarrow{\tau} M'$ and transition t can work with M'. A transition is reachable in the net N, if it is reachable from M_0.

A transition t is *alive* if it is reachable from any markup from $R(N)$. A Petri net is *alive* if *all its transitions* are alive.

2. Development of a method for constructing a computational process model based on a Petri net

We introduce some definitions and thesis.

Definition 1. An elementary computational process (ECP) will be called a fragment of an CP that performs a functionally minimal complete computation process, and the term "computation" is understood in a broad sense: computation is the processing of a variety of data, from simple to multimedia.

In the paper, CPs is considered irrespective of the form and classes of problems being solved. The only restriction for the CP is that an algorithm must exist for it, i.e. CP must satisfy the following requirements: certainty, convergence and mass character.

The quality of the CP, as mentioned above, is understood not only as the absence of gross errors (freezing, looping, etc.), but also its optimality in the sense of a minimum of complexity.

There are several criteria for evaluating the complexity of an algorithm, or CP. Most often, the *order of growth* of time and memory capacity needed to solve the problem is used, with an increase in the amount of input data. We associate with each concrete task a certain number n, called its *size*, which would express a measure of the amount of input data. For example, the size of the matrix multiplication task may be the largest size of the matrix of factors, the size of the task on the graph may be the number of vertices/edges of a given graph, etc.

In this case, asymptotic estimates of complexity in the worst case are used (top marks) – $O(f(n))$, where f expresses the growth rate of complexity, for example, $f = n \log n$.

In addition, eliminating repetitions, rearranging modules in order to speed up the CP, reducing the complexity of the CP due to special techniques (for example, splitting a task into subtasks and balancing them, etc.) also leads to optimization of the CP.

Definition 2. Macrooperation (MO) is a fragment of CP (ECP and more complex functionally completed constructions) endowed with the following attributes: name, designation, function, parameters with details of their definition.

For example, the name MO is "Counting cycle", the designation is "MO_{CC}", the function is "iteration count"; parameter – a variable called a cycle parameter, for example, i, the range of parameter values $i = 1 .. n$.

MOs include at the lower level of the hierarchy simple arithmetic operations and arithmetic expressions, at the next level – rearrangement of array elements, offsets of sequence elements, comparison of rows/columns of a table, etc.

We proposed [15] to consider CP in the form of two components: computational and control. The first component is macrooperations, which are functionally completed operations of different hierarchical levels. The second component provides the organization of control of CP, that is, the alignment of the process in a certain order. The controls are proposed – "follow", "select", and "transition"; a theorem on the functional completeness of these control elements (CE) is proved. Well-known control structures are built from CE [16]: an alternative and cycles of three types: countable, conditional of the first kind (with a precondition) and conditional of the second kind (with a postcondition).

Thus, the theorem is fundamental: it allows us to formalize the CP recording [17] and opens up the possibility of constructing algebra of control structures.

Note that control structures (composition, alternative, iteration, and more complex structures) can be built into the MO. So, a simple sorting by inserts can be considered as a MO of the 3rd hierarchy level with two cycles controlling the sorting process.

In [18], the MOs were located; MO lists are ordered taking into account the hierarchy of CP fragments. The final list contains 19 MOs.

In [19; 20], the consideration of another class of CPs – the search for coverage – was begun; the search algorithms for coverage by boundary search and reduction of the coverage table using the

corresponding theorems are analyzed. A feature of this class of tasks is the work with tables. For the mentioned algorithms, the total number of elementary MOs was 22, which gives reason to assert a relatively small variety of MOs and the possibility of forming their complete list for certain classes of problems.

Statement. Any CP can be represented by a combination of a relatively small number of MOs and control elements/structures.

In accordance with the proposed approach, for the developed method of constructing a CP model on the basis of a Petri net, it provides operations for decomposing a CP into an ECP, debugging them on the corresponding Petri net, assembling the complete net and modeling it. The essence of the method is

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

the sequential transformations of the CP into a Petri net and modeling the resulting net.

The **method** of constructing a CP model based on a Petri net includes:

1) decomposition of CP into ECPs;

2) formalization ECP in the form of MO;

3) building a model of ECP in the form of a fragment of Petri net for each MO;

4) Petri net fragment modeling;

5) building a complete Petri net;

6) modeling a complete Petri net.

Procedure that implements the method

This method is implemented as a **procedure** that at each stage implements one of the operations of the method. At the same time, interaction with the **library** is carried out**.**

Here is a description of the proposed procedure.

Input: initial CP in the form of a verbal description of the algorithm.

Output: MOs, Petri net fragments, corrected CP and a complete Petri net.

Verbal description of the procedure

1. The initial CP is divided into elementary computing processes.
2. ECPs identified with MOs with the assignment of certain attributes.
3. MOs, according to certain rules, are transformed into Petri net fragments.
4. Fragments of the Petri net are modeled. If the result is positive (no errors), then the MO and the corresponding fragment of the Petri net are entered into the library if they are not there. In the case of an unsatisfactory result, it is necessary to return to the original CP and correct the error (correct the CP) and then repeat steps 2 and 3.
5. Next, according to certain rules, fragments of the Petri net are assembled to obtain a complete network.
6. The resulting Petri net is modeled and the results are analyzed. In the case of an unsatisfactory result (the desired quality of the CP is not achieved), it is necessary to return to the original / previous version of the CP and correct the errors. If the desired quality is achieved, then the last versions of the CP and Petri nets and the corresponding fragments of the CP and Petri nets are entered into the library.

Rules for transforming a MO into a fragment of a Petri net and assembling a complete Petri net

Representation of MO by fragments of the Petri net is carried out according to the following rules.

1. MO is represented by a transition, which is framed by the input and output positions; point is placed in the entry position.
2. The selection operation is represented by a position with two outputs that are assigned priority 0 and 1.

The complete Petri net corresponding to the initial CP is constructed from fragments of the Petri net according to the rules:

1. The sequence of MOs is represented by the sequence of transitions corresponding to them; positions are placed between transitions; the sequence is framed by the input and output positions. A point is placed at the input position.
2. At the merging of individual sequences, the output position of the previous sequence merges with the input position of this sequence. The input position of the resulting sequence will be the input position of the previous sequence, and the output position will be the output position of this sequence.

Note that the obligatory alternation of positions and transitions leads to the need to introduce *fictitious* positions and transitions.

Library of descriptions of CPs and corresponding Petri nets

The library is designed to store debugged MOs, fragments of Petri nets, CPs, complete Petri nets and their descriptions. The library is organized hierarchically.

Four operations are defined for the library of descriptions: adding a new component (MOs, fragments of Petri nets, CPs, full Petri nets) to the library, searching for the specified component, selecting the specified component and deleting it.

The library is the core of the knowledge base, structured by classes of tasks: sorting, finding coverage, tasks on graphs, etc.

There are also components used in different classes of tasks. These common components are allocated in a separate group.

In certain sections of the library, the formulas of MOs and CPs should also be stored.

The example of the allocation of MO

Consider, as an example, the allocation of MOs in sorting by simple inserts.

Fig. 1 shows the graphical model of such sorting in ascending order. The graphic model contains a numerical axis on which dots indicate the sequence of array elements. Considered ideas retain their meaning when sorting has descending order.

The sorted sequence is divided into two subsequences: *finished* and *input*. The arrow shows the possible movement of the boundary (current) element.

The basic concept of "*border*" is introduced; this border separates the finished and input

subsequences. It uses the specification of: current, next and the initial border. A boundary element is an element immediately beyond the border.

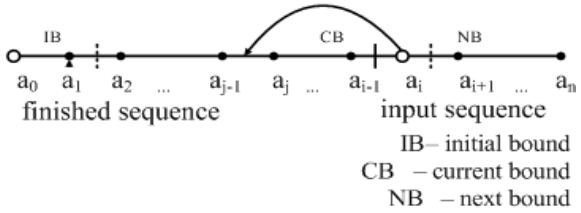

Fig. 1. CP model "Sorting by simple inserts"

The current state of the sorting process is considered and the new positions of the border, as well as its initial and starting position, are indicated. During the operation of the simple insertion sorting method for the current boundary element *x*, a suitable place in the finished subsequence is sought, for which a double inequality is used: $a_{j-1} \le x \le a_j$. Since *x* may be in the first position, and in this case the left side of the double inequality is missing, a dummy element is introduced a_0, called the "barrier. For a guaranteed hit *x* in the first position take $a_0=x$.

Fig. 2 shows a flowchart, constructed in accordance with the considered model of the sorting process. Iteration counters for internal and external loops are shown. The dashed-dotted line in the diagram represents the inner loop. This flowchart fully displays the CP.

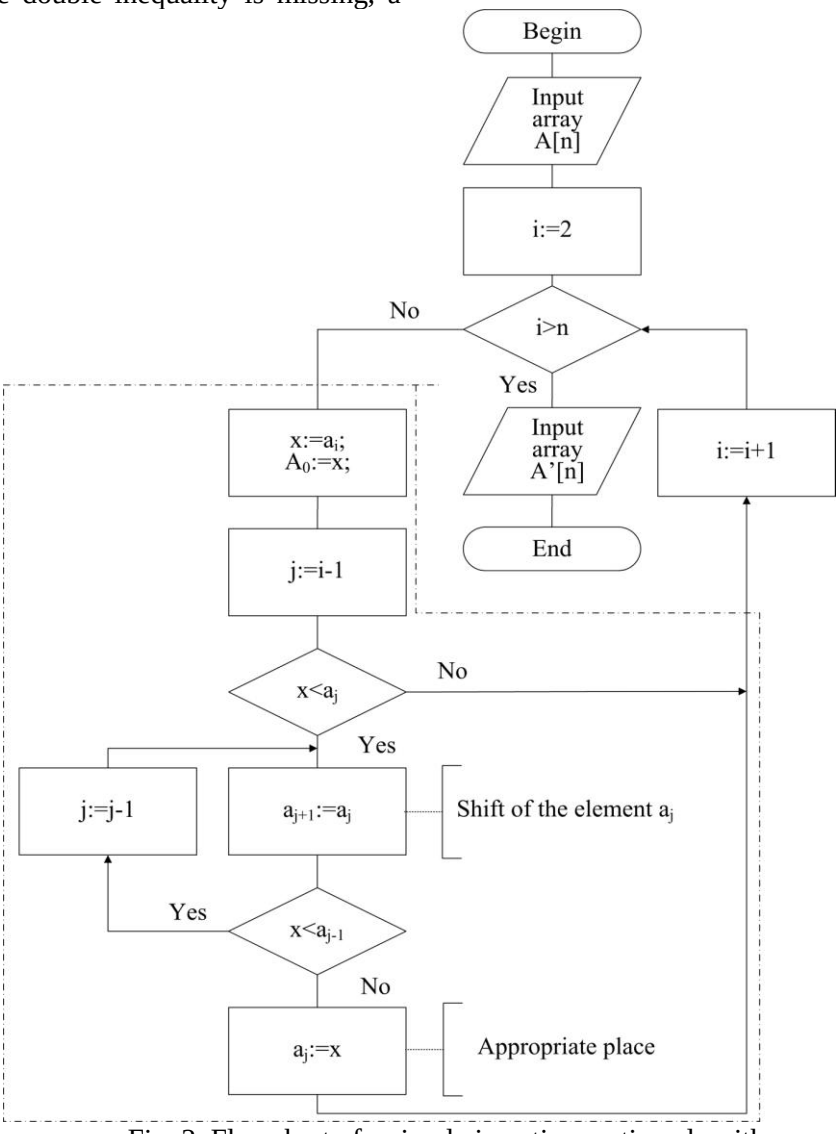

Fig. 2. Flowchart of a simple insertion sorting algorithm

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

From CPs for simple and selected complex sortings, MOs are selected and collected in the corresponding lists. These lists are systematized by combining into a common list with its subsequent minimization. Specific to the considered sortings is the MO "Compare – rearrange". In addition, for complex sortings, the MO "Sublisting" is specific. Two dozen MOs turned out to be sufficient to describe this class of sortings. At the same time, some MOs can also be used in CPs of other classes.

Here is a list of MOs for sorting by simple inserts:

1. MO "Search for a suitable place" (this is a joint implementation of MO "Counting Cycle" and MO "Comparison of Double Inequality").
2. MO "Shift an element by one position to the right".
3. The organization of the cycle under the condition.

3. Experimental results and evaluation

Consider an example of constructing and modeling a Petri net for the given sorting by simple inserts. Fig. 3 shows the Petri net constructed according to the rules defined above for this CP.

Traditionally, the Petri net contains the usual net links along which the chips are moved according to the scenario specified by the algorithm. However, for a full description of the CP, this is not enough – additional facilities are needed to implement the operation of choosing alternatives. The use of inhibitory bonds that implement the logic NOT turned out to be artificial. As the analysis showed, an introduction to the consideration of priorities is more appropriate: "1" corresponds to the TRUE alternative, and "0" corresponds to the FALSE alternative. For modeling, we use a special class of Petri nets with a priority mechanism – these are E-nets [21] (Evaluation nets). Emulators are known [22], which are most suitable for our purposes; we have chosen a freeware program PIPE 2 [23].

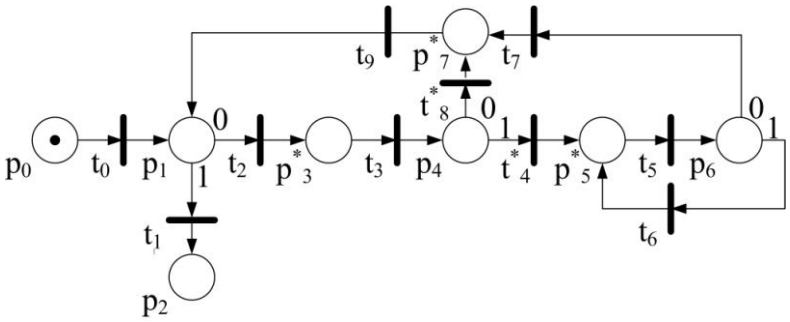

Fig. 3. Petri net for CP of sorting inserts

The selected Petri net editor uses the cross-platform programming language Java, which allows to run nets on various operating systems. PIPE2 allows creating, editing and simulating stochastic Petri nets, including with inhibitor arcs. The editor has a number of additional analysis modules that expand its functionality.

The description of the components of the Petri net as a model for CP sorting by simple inserts is given in Table 1 and Table 2.

Based on the structure of the Petri net, we define transition scenarios that encompass all possible branches / contours of the structure.

We have:

1) p0→t0→p1→t1→p2;

2) p0→t0→NOTp1→t2→p3→t3→NOTp4→t8 →p7→t9→p1→t1→p2;

3) p0→t0→NOTp1→t2→p3→t3→NOTp4→t8 →p7→t9→NOTp1→t2→p3→t3→NOTp4→…→t9 →p1→t1→p2;

4) p0→t0→NOTp1→t2→p3→t3→p4→t4→p5 →t5→p6→t6→p5→…→NOTp6→t7→p7→t9→ NOTp1→t2→p3→t3→p4→t4…→t3→NOTp4→… →t9→ p1→t1→p2.

Table 1. Positions and their meaning

Positions	Meaning
p_0	Start
p_1	i>n
p_2	End
p_3	∅
p_4	$x<a_j$
p_5	∅
p_6	$x<a_{j-1}$
p_7	∅

We believe that in the position p_i condition p_i is checked.

Table 2. Transitions and their meaning

Transitions	Meaning
t_0	input A; i:=2
t_1	output A’
t_2	x:=a_i; a_0:=x
t_3	j:=i–1
t_4	NOP
t_5	a_{j+1}:= a_j
t_6	j:=j–1
t_7	a_j:=x
t_8	NOP
t_9	i:=i+1

The first scenario is trivial and actually tests the ability to perform a sort operation. The second scenario occurs if there are only two numbers in the array to be sorted. The third scenario corresponds to the situation when the source array is already sorted. There is a cycle … NOTp1→t2→p3→t3→NOTp4 →t8→p7→t9→NOTp1…, exit from which occurs when checking all numbers of the array by condition p1: i>n. The fourth scenario is the main one with an inner loop … p5→t5→p6→t6→ 5… with an exit from it by condition NOTp6, and external … NOTp1→t2→p3→t3→NOTp4→t8→p7→t9→NO Tp1…, exit from which occurs when the condition p1: i>n is true, that is, after sorting all the elements of the array.

Note that triggering transitions t1 or t2, t4 or t8, t6 or t7 occurs depending on the value of the condition being checked p1, p4, p6 respectively in the corresponding position (truth is 1, false is 0). On the net scheme, this is indicated by the numbers 0 and 1 near the arc leaving the position.

The modeling results showed that each of the scenarios is executed, i.e., the net is alive, and with the correct initial data the modeling ends in a finite number of steps.

Conclusion

In this paper, we propose a new approach to improving the quality of the computational process (CP), which consists in preliminary modeling of the CP by the Petri net and transferring the improved CP to the stage of software implementation. The choice of a Petri net for modeling is based on the fact that the Petri net has a more powerful description language than other known models (for example, automatic). To implement this approach, the method for constructing a CP model based on a Petri net was developed. The method is implemented in the form of a procedure consisting of a sequence of the following steps: decomposition of the CP into the macrooperations (MOs), transformed them into Petri net fragments, assembly of the complete net and its research. The proposed approach is consistent with the criterion of “scientific innovation”.

Petri net modeling allows us to determine the reachability of any vertex from a given one, i.e. verify all branches and contours of the CP, as well as the survivability of the net. In this case, CP errors are detected: looping, hovering, unrealizability of some of its fragment. CP optimization is also possible: detection and elimination of repetition of CP fragments, rearrangement of CP modules to accelerate its execution, etc.

The examples of simple sorting and solving the coverage task show the possibility of isolating MOs and minimizing their number. However, to complete this stage, a methodology should be developed for the general case.

A more detailed study of the organization of a library of descriptions of CPs and their fragments and corresponding fragments of the Petri net, as well as debugged CPs and their corresponding complete Petri nets, is required. The stage of interaction with the library was worked out sketchy – further extensive research is required here. This would simplify the description of the CP and its modeling.

An example of sorting by inserts shows the transformation of a CP into a Petri net, as well as a study of the operability of this Petri net for all net behavior scenarios; the correctness of the Petri net operation for this CP is shown. For the general case of studying the behavior of a Petri net, it is necessary to develop a method for converting a CP into a Petri net and a method for modeling a CP in the general case based on the corresponding Petri net.

References

1. Shulga, T. E., Ivanov, E. A., Slastihina, M. D. & Vagarina, N. S. (2016). “Developing a software system for automata-based code generation” [Text] *Programming and Computer Software*, Vol. 42, Issue 3, pp. 167-173, https://doi.org/10.1134/s0361768816030075.

2. Lyubchenko, V. (2006) “K probleme sozdaniya modeli parallel'nyh vychislenij” [To the Problem of Creating a Parallel Computing Model]

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

[Text] *Proceedings of the Third International Conference "Parallel Computing and Control Problems, PACO'2006"* (in Russian).

3. Truhin, M. P. (2018). "Modelirovanie signalov i system" [Modeling Signals and Systems] [Text] Setevye modeli: Uch. Posobie, *Publ. Izd-vo Ural. Un-ta*, Yekaterinburg: Russian Federation, 204 p. (in Russian).

4. Latsou, C., Dunnett, S. J. & Jackson, L. M. (2019). A new Methodology for Automated PetriNet Generation: Method Application [Text] *Reliability Engineering and System Safety*, Vol. 185, pp. 113-123, https://doi.org/10.1016/j.ress.2018.12.017.

5. Badouel, E., Bernardinello, L., & Darondeau, P. (2015). "Petri Net Synthesis". *Springer Publ*., Heidelberg: 339 p. http://dx.doi.org/10.1007/978-3-662-47967-4.

6. Reisig, W. (2013). "Understanding Petri Nets", "Modeling Techniques, Analysis Methods, Case Studies", 211 p. *Springer Publ*., Hiedelberg: http://dx.doi.org/10.1007/978-3-642-33278-4.

7. (2016). Kleijn, J. (eds) "Transactions on Petri Nets and Other Models of Concurrency XI". "Lecture Notes in Computer Science", Vol. 9930, *Springer Publ.*, Berlin, Heidelberg: DOI https://doi.org/10.1007/978-3-662-53401-4_9.

8. (2015). Knight, S., Lanese, I., Lluch Lafuente A., Vieira, H. T. (Eds.). *8th Interaction and Concurrency Experience*, (ICE 2015) EPTCS 189, pp. 53-67, doi:10.4204/EPTCS.189.6.

9. Schlachter, U. (2016). "Petri Net Synthesis for Restricted Classes of Nets". "Application and Theory of Petri Nets and Concurrency". PETRI NETS 2016. "Lecture Notes in Computer Science", Vol. 9698, *Springer Publ*., Cham. DOI https://doi.org/10.1007/978-3-319-39086-4_6.

10. Amparore, E. G., Donatelli, S., Beccuti, M., Garbi, G., & Miner, A. (2018) "Decision Diagrams for Petri Nets: A Comparison of Variable Ordering Algorithms". "Transactions on Petri Nets and Other Models of Concurrency XIII", "Lecture Notes in Computer Science", Vol 11090. *Springer Publ.*, Berlin, Heidelberg: DOI https://doi.org/10.1007/978-3-662-58381-4_4.

11. Shershakov, S. A., Kalenkova, A. A., & Lomazova, I. A. (2017) "Transition Systems Reduction: Balancing Between Precision and Simplicity". "Transactions on Petri Nets and Other Models of Concurrency XII". "Lecture Notes in Computer Science", Vol 10470. *Springer Publ*., Berlin, Heidelberg: DOI https://doi.org/10.1007/978-3-662-55862-1_6.

12. Ribeiro, Joel & Carmona, Josep. (2016). "A Method for Assessing Parameter Impact on Control-Flow Discovery Algorithms". "Transactions on Petri Nets and Other Models of Concurrency XI". "Lecture Notes in Computer Science", Vol. 9930. *Springer Publ*., Berlin, Heidelberg. DOI https://doi.org/10.1007/978-3-662-53401-4_9.

13. Verlan, A. F., Polozhaenko, S. A., Prokofieva, L. L., & Shylov, V. P. (2019). Algorithmization of the Failed Subschemes Localization Process, *Herald of Advanced Information Technology*, Odessa, Ukraine, *Publ. Science & Technology*, Vol. 2 No. 1, pp. 37-46, doi: 10.15276/hait.02.2019.4.

14. Kalnauz, D. V., & Speranskiy, V. A. (2019). Productivity Estimation of Serverless Computing, *Applied Aspects of Information Techology*, Odessa, Ukraine, *Publ. Science & Technology*, Vol. 2 No. 1, pp. 20-28, doi: 10.15276/aait.02.2019.2.

15. Paulin, O. "O funkcional'noj polnote elementov upravleniya vychislitel'nymi processam". (2016). [On the Functional Completeness of Control Elements of Computing Processes], Scientific looking to the future. Odessa, Ukraine, *Publ. Institute of Maritime and Enterprise,* Release 4, Vol. 4, pp. 4-8 (in Russian). DOI: 10.21893/2415-7538-2016-04-4-018.

16. Goodman, S. E. & Hedetniemi, S. T. (2002). "Introduction to the Design and Analysis of Algorithms", *Publ. Tata Mcgraw-Hill*, New Delhi. https://doi.org/10.2307/2346822.

17. Paulin, O., Komlevaya, N., & Marulin, S. (2016). "Ob upravleniya vyichislitelnyimi protsessami" [About a Management by the Calculable Processes]. *Trudy XVII International Research and Practice Conference "Modern Information and Electronic Technologies",* Odesa: Ukraine, Vol. 1, pp. 20-21 (in Russian).

18. Paulin, O., Komlevaya, N., & Marulin, S. "O vydelenii makrooperacij iz vychislitel'nyh processov sortirovki massivov dannyh". (2016). [Macro-operations Extraction out of Computation Process array Sorting Data]. CEUR Workshop Proceedings (in Russian) https://www.scopus.com/record/display.uri?eid=2-s2.0 84983598214&origin=inward&txGid=ccde5cf7d4ee8f843deb8079c5883e22.

19. Paulin, O. "Vychislitel'nye modeli algoritmov pokrytiya" (2016). [Computational Models of Coverage Algorithms]. *Computer science and Mathematical Methods in Modeling*, Vol. No. 4, pp. 385-396. (in Russian)

20. Paulin, O. "Vychislitel'nye modeli processa predvaritel'nogo sokrashcheniya tablicy pokrytiya". (2017). [Computational Models of the Process of Preliminary Reduction of the Coverage Table], *Computer science and mathematical methods in modeling*, Vol. No. 4, pp. 333-338 (in Russian).

21. Cabac, L., Haustermann, M. & Mosteller, D. (2018). "Software Development with Petri nets and agents: Approach, Frameworks and tool set" [Text] *Science of Computer Programming*, Vol. 157, pp.56-70. https://doi.org/10.1016/j.scico.2017.12.003.

22. Dingle, N. J., Knottenbelt, W. J. & Suto, T. (2009). "PIPE2: a Tool for the Performance Evaluation of Generalized Stochastic Petri Nets". [Text] *Newsletter ACM SIGMETRICS Performance Evaluation*, Vol. 36, Issue 4, pp. 34-39, NY, USA. https://doi.org/10.1145/1530873.1530881.

23. "PIPE2: Platform-Independent Petri net Editor" [Electronic resource] Access mode: URL: http://pipe2.sourceforge.net, Title from the screen (Active link – 28.10.2019).

Received 17.09.2019
Received after revision 26.11.2019
Accepted 02.12.2019

УДК 004.042

[1]**Паулін, Олег Миколайович,** доктор технічних наук, доцент кафедри системного програмного забезпечення інституту комп'ютерних систем, E-mail: paolenic@yandex.ru, ORCID ID: 0000-0002-2210-8317
[1]**Комлева, Наталія Олегівна,** кандидат технічних наук, доцент кафедри системного програмного забезпечення інституту комп'ютерних систем, E-mail: nkomlevaya@gmail.com, ORCID ID: 0000-0002-2430-0134
[1]**Марулін, Станіслав Юрійович,** кандидат технічних наук, доцент кафедри системного програмного забезпечення інституту комп'ютерних систем, E-mail: stanislavmaru@gmail.com, ORCID ID: 0000-0003-0755-0104
[1]**Ніколенко, Анатолій Олександрович,** кандидат технічних наук, доцент кафедри інформаційних систем інституту комп'ютерних систем, E-mail: anatolyn@ukr.net, ORCID ID: 0000-0002-9849-1797
[1]Одеський національний політехнічний університет, пр. Шевченка, 1, м. Одеса, Україна, 65044

МЕТОД ПОБУДОВИ МОДЕЛІ ОБЧИСЛЮВАЛЬНОГО ПРОЦЕСУ НА ОСНОВІ МЕРЕЖІ ПЕТРИ

Анотація. *Метою роботи є підвищення якості обчислювального процесу, що вирішує поставлену задачу, за рахунок його моделювання і налагодження на основі мережі Петрі. Під якістю обчислювального процесу розуміється відсутність помилок (за циклювання, параліч, неможливість реалізації деякого фрагменту і т.п.) і його оптимізація за критерієм мінімуму складності. Пропонується новий підхід до аналізу обчислювального процесу, заснований на попередньому моделюванні мережами Петрі як фрагментів обчислювальних процесів, так і повних обчислювальних процесів. Це дозволить виявляти багато помилок на стадії моделювання обчислювального процесу. Обчислювальний процес розглядається як сукупність макрооперацій, які є функціонально закінченими операціями різного ієрархічного рівня. Для виділення макрооперацій з обчислювального процесу проводиться його декомпозиція на елементарні (базові) обчислювальні конструкції. Формулюється твердження про те, що будь-який обчислювальний процес може бути сконструйований на основі відносно невеликої кількості макрооперацій. Для реалізації нового підходу ставиться і вирішується завдання розробки методу побудови мережі Петрі по заданому обчислювальному процесу. Суть запропонованого методу полягає в розбитті обчислювального процесу на макрооперації, побудові для кожної макрооперації фрагмента мережі Петрі, моделюванні всіх фрагментів, збірки з фрагментів мережі повної мережі Петрі і її моделюванні. Для реалізації методу*

розробляється процедура побудови моделі обчислювального процесу. Наводиться опис етапів даної процедури: декомпозиція обчислювального процесу на макрооперації за запропонованими правилами, переклад макрооперацій у фрагменти мережі Петрі і їх моделювання, збір за запропонованими правилами повної мережі Петрі і моделювання отриманої мережі Петрі. Результати реалізації всіх етапів процедури заносяться в бібліотеку, призначення якої – накопичення знань про обчислювальні процеси, відповідні їм мережі Петрі та результати моделювання. Це дозволяє спростити процес моделювання нового обчислювального процесу за рахунок використання вже налагоджених фрагментів. У разі виявлення помилки в обчислювальному процесі або його не оптимальності обчислювальний процес коригується, що і дозволяє підвищити його якість за вказаними вище критеріями. На прикладі сортування вставками експериментально підтверджується правильність роботи побудованої мережі Петрі із застосуванням заявленого методу.

Ключові слова: *обчислювальний процес; мікрооперація; метод; процедура; мережа Петрі; моделювання; бібліотека*

УДК 004.042

[1]**Паулин, Олег Николаевич,** доктор технических наук, доцент кафедры системного программного обеспечения института компьютерных систем,
E-mail: paolenic@yandex.ru, ORCID ID: 0000-0002-2210-8317
[1]**Комлевая, Наталия Олеговна,** кандидат технических наук, доцент кафедры системного программного обеспечения института компьютерных систем,
E-mail: nkomlevaya@gmail.com, ORCID ID: 0000-0002-2430-0134
[1]**Марулин, Станислав Юрьевич,** кандидат технических наук, доцент кафедры системного программного обеспечения института компьютерных систем,
E-mail: stanislavmaru@gmail.com, ORCID ID: 0000-0003-0755-0104
[1]**Николенко, Анатолий Александрович,** кандидат технических наук, доцент кафедры информационных систем института компьютерных систем,
E-mail: anatolyn@ukr.net, ORCID ID: 0000-0002-9849-1797
[1]Одесский национальный политехнический университет, пр. Шевченко, 1, г. Одесса, Украина, 65044

МЕТОД ПОСТРОЕНИЯ МОДЕЛИ ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА НА ОСНОВЕ СЕТИ ПЕТРИ

Аннотация. *Целью работы является повышение качества вычислительного процесса, решающего поставленную задачу, за счёт его моделирования и отладки на основе сети Петри. Под качеством вычислительного процесса понимается отсутствие ошибок (зацикливание, паралич, нереализуемость некоторого фрагмента и т.п.) и его оптимизация по критерию минимума сложности. Предлагается новый подход к анализу вычислительного процесса, основанный на предварительном моделировании сетями Петри как фрагментов вычислительных процессов, так и полных вычислительных процессов. Это позволит выявлять многие ошибки на стадии моделирования вычислительного процесса.*
Вычислительный процесс рассматривается как совокупность макроопераций, которые являются функционально законченными операциями различного иерархического уровня. Для выделения макроопераций из вычислительного процесса проводится его декомпозиция на элементарные (базовые) вычислительные конструкции. Формулируется утверждение о том, что любой вычислительный процесс может быть сконструирован на основе относительно небольшого числа макроопераций. Для реализации нового подхода ставится и решается задача разработки метода построения сети Петри по заданному вычислительному процессу. Суть предлагаемого метода состоит в разбиении вычислительного процесса на макрооперации, построении для каждой макрооперации фрагмента сети Петри, моделировании всех фрагментов, сборки из фрагментов сети полной сети Петри и её моделировании. Для реализации метода разрабатывается процедура построения модели вычислительного процесса. Приводится описание этапов данной процедуры: декомпозиция вычислительного процесса на макрооперации по предложенным правилам, перевод макроопераций во фрагменты сети Петри и их моделирование, сбор по предложенным правилам полной сети Петри и моделирование полученной сети Петри. Результаты реализации всех этапов процедуры заносятся в библиотеку, назначение которой – накопление знаний о вычислительных процессах, соответствующих им сетях Петри и результатах моделирования. Это позволяет упростить процесс моделирования нового вычислительного процесса за счёт использования уже отлаженных фрагментов. В случае

выявления ошибки в вычислительном процессе или его не оптимальности вычислительный процесс корректируется, что и позволяет повысить его качество по указанным выше критериям. На примере сортировки вставками экспериментально подтверждается правильность работы построенной сети Петри с применением заявленного метода.

Ключевые слова: *вычислительный процесс; макрооперация; метод; процедура; сеть Петри; моделирование; библиотека*

Paulin, Oleg Nikolaevich, Doctor of Technical Sciences
Area of scientific interests: algorithms, parallel and distributed systems and computing, processing large data streams, computer system components

Komleva, Nataliia Olegovna, Candidate of Technical Sciences
Area of scientific interests: data analysis, analysis of the quality of information sources, machine learning

Marulin, Stanislav Urievich, Candidate of Technical Sciences
Area of scientific interests: database development, object oriented design and programming

Nikolenko, Anatoly Alexandrovich, Candidate of Technical Sciences
Area of scientific interests: modeling systems, digital signal and image processing

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)