UDC 004.021
DOI https://doi.org/10.32782/tnv-tech.2023.5.7

CONCEPT OF BUILDING A LIBRARY OF TASKS AND SOLUTIONS

Paulin O. M. – *Doctor of Technical Sciences,*
Associate Professor at the Department of Software Engineering
National University "Odesa Polytechnic"
ORCID ID: 0000-0002-2210-8317

Komleva N. O. *– Doctor in Engineering, Associate Professor,*
Head of the Department of Software Engineering
National University "Odesa Polytechnic"
ORCID ID: 0000-0001-9627-8530

Nikitchenko M. I. – *Graduate Student*
National University "Odesa Polytechnic"
ORCID ID: 0009-0007-9560-7057

The article develops the concept of building a library of tasks and their solutions in the form of computational processes (CPs) and Petri nets (PNs) with the preliminary transformation of CPs into PNs. The conversion includes the extraction of macro operations (MOs) from CPs as a result of their analysis, the construction of PNs and their fragments as models of CPs and MOs, and the modeling of PNs and their fragments. The architecture, structure and interface of the library are proposed.

Computational process is a sequence of time-ordered operations and procedures of varying degrees of complexity. The set of PNs, as well as the set of problems for which PNs are solutions, is divided into classes and subclasses. A class includes problems such as Combinatorial, Linear Algebraic Equations, etc.; the Combinatorial class includes subclasses of problems such as Sorting, Search, Sets, Graphs, etc. A macro operation is a fragment of a GP that is a functionally complete computational construct. MOs can have different levels of complexity, starting with the elementary one; they can be built hierarchically. A Petri net is a bipartite graph with two interconnected objects: event-condition, process-resource, etc. Fragments of the PN are models of the MO.

The components of the library are tree nodes supplemented with files of their verbal descriptions built according to certain rules.

The architecture, structure, interface, and operations of the library provide convenient management, editing, and expansion of knowledge about computational processes and macro operations, making the library open for further research and development by the developer. All files are available to the user, but without the right to edit.

The paper presents an example of the initial filling of the library for the Sorting task.

Key words: *concept, library, architecture, structure, tree, task, subtask, tree component, verbal description, computational process, macro operation, Petri net, Petri net fragment, modeling, example.*

Паулін О. М., Комлева Н. О., Нікітченко М. І. Концепція побудови бібліотеки задач та рішень

У статті розроблено концепцію побудови бібліотеки задач та їхніх розв'язків у вигляді обчислювальних процесів (ОП) і мереж Петрі (МП) з попереднім перетворенням ОП у МП. Перетворення охоплює виділення макрооперацій (МО) із ОП у результаті їхнього аналізу, побудову МП і їхніх фрагментів як моделей ОП і МО, а також моделювання МП і їхніх фрагментів. Пропонуються архітектура, структура та інтерфейс бібліотеки.

Обчислювальний процес – послідовність упорядкованих у часі операцій і процедур різного ступеня складності. Безліч МП, як і безліч задач, для яких МП є рішеннями, розбивається на класи та підкласи. До класу входять задачі «Комбінаторні», «Лінійні алгебраїчні рівняння» тощо; до класу «Комбінаторні» входять підкласи задач

«Сортування», «Пошук», «Множини», «Графи» тощо. Макрооперацією називатимемо фрагмент ОП, що являє собою функціонально закінчену обчислювальну конструкцію. МО можуть мати різний рівень складності, починаючи з елементарного; вони можуть бути збудовані ієрархічно. Мережа Петрі – дводольний граф із двома взаємопов'язаними об'єктами: подія-умова, процес-ресурс тощо. Фрагменти МП є моделями МО.

Компонентами бібліотеки є вузли дерева, доповнені файлами їхніх словесних описів, побудованих за певними правилами.

Архітектура, структура, інтерфейс та операції бібліотеки забезпечують зручне управління, редагування та розширення знань про обчислювальні процеси та макрооперації, роблячи бібліотеку відкритою для подальших досліджень і розвитку розробником. Користувачеві доступні всі файли, але без права редагування.

У роботі наведено приклад первинного заповнення бібліотеки для задачі «Сортування».

Ключові слова: *концепція, бібліотека, архітектура, структура, інтерфейс, дерево, задача, підзадача, компонент дерева, словесний опис, обчислювальний процес, макрооперація, мережа Петрі, фрагмент мережі Петрі, моделювання, приклад.*

Introduction. In general, a computational process (CP) is understood as a sequence of time-ordered operations and procedures of varying complexity. The computational process must necessarily form the basis of software, since it largely determines the quality of software. There can be many program implementations for one CP taking into account different technologies and programming languages. That is why it is important to build CP qualitatively. However, at present there are no quality standards for CP.

Traditionally, the quality of CP is defined by the absence of errors and the level of optimality. Among the types of possible errors are looping, freezing, as well as emergency stop of the CP in case of fatal errors, for example, division by zero. Optimization of CP is most often carried out by the criterion of minimum complexity.

Indirectly, the quality of this CP can be judged by analyzing the program developed on its basis for compliance with the requirements according to ISO 12207, ISO 9000, CMM standards. Non-compliance with such requirements leads to the necessity of error correction, re-engineering and repeated testing of the program code, which requires additional resources, including time, and is much more expensive than error elimination at the initial stage of development. However, in practice the development of a quality CP, as the basis of quality software, is not always given due attention, which causes the described problems. That is why the construction of quality CP is topical.

An effective way to improve the CP is to model it, and different approaches and tools are used [1].

In this paper we propose a new approach, which consists in breaking down the CP into elementary computational processes/constructions, modeling and debugging of these constructions by Petri nets, final assembly of separate constructions into a complete network corresponding to the whole CP, and its modeling. The application of this approach allows to increase the quality of CP and, accordingly, the quality of its program implementations.

With all the variety of CPs there is a possibility of their formalization, which allows to choose a tool for description and further analysis of these processes. In the best known tools (finite automata, SWITCH-technology, Petri nets, unified modeling language – UML) were reviewed and analyzed; it was concluded that Petri nets have significantly greater expressiveness of their language, as they occupy [2] an intermediate position between a finite automaton and a Turing machine. Petri nets reflect the mutual dependence "event-condition" ("process-resource"), which makes it possible to use them widely for modeling. In [3], in particular, a review and analysis of applications of Petri nets for the last 25 years is carried out.

As can be seen from the analysis, despite the rather large range of topics of works, the authors have not made attempts to adjust the original CP\algorithm using Petri nets built on their basis as a tool to improve the quality of CP.

The aim of the work is to improve the quality of CPs by selecting the concept of building a library of CPs and their models based on Petri net.

In order to achieve this goal, the following objectives should be accomplished:

– development of the library architecture;

– development of the library structure.

It should be noted that in [1] the rules of macrooperations (MO) extraction from the CP were formulated, the method of building the model of MO and CP on the basis of PN was developed; the method is realized in the form of a procedure, which at each stage reproduces one of the operations of the method. The procedure interacts with the library. The rules for transforming MO into a fragment of PN and assembling a complete PN from fragments are proposed.

Main part

1. Preliminary information

Let’s introduce some definitions and provisions for consideration.

Definition 1: An elementary computational process (ECP) is a minimal functionally complete computational process.

In this paper, the term “computation” is understood in a broad sense: computation is the processing of a variety of data, from the simplest to multimedia data. The only limitation for CP is that there must be an algorithm for it, i.e. CP must satisfy the following requirements: definiteness, convergence and mass.

The quality of an CP is understood not only as the absence of gross errors (freezing, looping, etc.), but also as its optimality in the sense of minimum complexity. Optimality is achieved by eliminating repetitions, rearranging modules, etc., as well as by special techniques (e.g., splitting a task into subtasks and balancing them).

Definition 2. We will call a *macrooperation (*MO) a fragment of an CP (elementary and more complex functionally complete computational constructs) arranged hierarchically.

MOs include simple arithmetic and logical operations and expressions at the lower level, and permutations of array elements, shifts of sequence elements, comparison of table rows/columns, etc. at the next level. Thus, for all sorts, the MO “compare and rearrange” for each pair of elements of the sorted sequence is characteristic. In this case, the formation of a particular pair of elements is determined by the sorting method, but this is the next level. Note that sorting itself can be considered as an MO of a higher hierarchy level.

CP can be represented in the form of two components: computing and controlling. The first component is macro-operations, which are functionally complete operations of different hierarchical levels. The second component provides the organization of CP control, i.e. building the process in a certain order. We have proposed the following elements of CP control: following, selection and transition; the theorem of functional completeness of these control elements (CE) has been proved. From CEs we can construct the known control structures: alternative and cycles of three kinds: counting, conditional of the 1st kind (with precondition) and conditional of the 2nd kind (with postcondition). Thus, the theorem is fundamental: it allows us to formalize the recording of CPs and opens the possibility of constructing the algebra of control structures.

Note that control structures (composition, alternative, iteration and more complex structures) can be embedded in MOs. Thus, a simple insertion sorting can be considered as an MO of the 3rd level of hierarchy with two cycles controlling the sorting process.

Assertion. Any CP can be represented by a collection of a relatively small number of MOs and governance elements/structures.

In accordance with the proposed approach for the developed method of constructing an CP model on the basis of PN (PN-model), it provides the operations of decomposition of CP into ECPs, their debugging on the corresponding Petri net, assembly of the complete network and its modeling. The essence of the method consists in the sequential transformation of CP into PN and modeling of the resulting network.

2. Development of a library of the CP\MO and their PN-models

The library is intended for storing debugged CPs and MOs (CP\MOs), their PN-based models (PN-models) and descriptions, and the filling and editing of library components are operations external to the library. The library is based on the classification represented by a tree; the main sign of the library classification is the class of solved problems. The library is open for extension.

The following provisions are utilized in the construction of the MO library:

– the library is built hierarchically according to the classification of tasks;

– library contains knowledge about the problem, their solution methods, as well as files of the problem solution in general form;

– components of the library are the category of the problem, its type (class), genus (solution method) and corresponding description files, including solution files (CP\MOs and their PN-models).

Knowledge is formed by expanding the tree structure by attributing descriptions to its nodes in the form of theoretical material corresponding to the node name (Name + Attributes/parameters). This material includes: brief theoretical information and references to sources, where you can learn more about the essence of the problem and methods of its solution.

2.1. The architecture of the library of CP\MOs and their PN-models

The library includes an open part, which is accessible to the user, and a closed part, which is maintained by the developer.

The library architecture includes a tree that realizes the classification of problems, methods of their solution and component description files; operations on components belong to the closed part of the library.

The component description files for each level of the library hierarchy are different according to the meaning of the component.

The solution files are: CPs (verbal descriptions of algorithms) and MOs, which constitute CPs, and PN models of CP\MOs (complete PN and its fragments).

There are 5 operations defined for library components: adding a new component to the library, editing it, searching for a given component, selecting a certain component, deleting a component.

2.2. The structure of the library of CP\MOs and their PN-models

The library structure is represented as a tree. Here 0-level is the root with the name TASK; it is the entrance to the library. Level 1 – categories of tasks: deterministic and random (stochastic) tasks. Level 2 – problem classes; Level 3 – problem subclasses; Level 4 – problem solving methods; Level 5 – solution files.

– Consider the classes of tasks for initial library completion.

– Combinatorial problems: combinatorics, sorting, searching, covering, ...

– Problems on sets: union, intersection and difference of sets, ...

– Problems on graphs (networks): finding optimal paths and routes, independent cycles, shortest coloring, ...; optimal flows, ...

– Tasks on trees: tree traversal algorithms (width, depth, internal – for a binary tree); converting a regular tree to a binary tree; tree balancing; ...

– Solution files: CPs, represented by verbal descriptions of their algorithms, and MOs of different levels of complexity; PN-models of CP\MOs.

There are common MOs used in different classes of tasks; these are distinguished in a separate group.

Note that MOs are endowed with the following attributes: *name, designation, function, parameters with details of their definition.*

Example. Let’s consider briefly filling the library with information about sorting and detailed information about sorting by inserts.

Note. The CP\MOs for the other sorts and their corresponding PNs will be considered in the next article.

The “Combinatorial” node is assigned a file with a list of tasks and their brief characterization, which are essentially combinatorial or can be represented as combinatorial [5].

The “Sorting” node is assigned a file of general sorting characteristics. Sorting according to Wirth [6] is a process of ordering, for example, in ascending order of a given sequence of numeric *keys,* which designate the sorted items, for example, by weight. So far only *internal* sorts, i.e. sorts for numbers stored in internal memory, have been considered. From a variety of sorting types 6 popular sorts are selected: 3 simple sorts (by exchange, selection, insertions) and 3 complex but faster sorts (Schella, pyramidal, fast). Sorting quality is evaluated by the number of comparisons of pairs of sequence elements and permutations of elements in a pair.

A table comparing the quality of the sorts is given in [6, paragraph 2.3.5].

For simple sorts, we introduce the notion of a “boundary” separating the sorted part from the original part, called the *finished* part.

The “Method” node has one method assigned to it, the insert method.

File **“Method by Inserts”**. Let us describe the method by insertions [6]. Sequence elements are divided into a *ready* subsequence a_1 , a_2, ...,a_{i-1} and an *input* subsequence a ,...,a_{in} (Fig.1). Then the idea is used: the next element from the input subsequence is inserted at a *suitable place* in the ready subsequence.

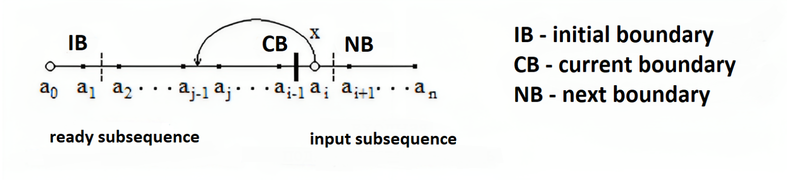

Fig. 1. Model of the sorting task by simple inclusions

The sorting process proceeds as follows.

We denote by *x the* current element to be inserted, as which we take the first element a_i *(x*:=a_i) immediately after the boundary and insert it at a suitable place in the ready subsequence; this place is defined by the relation: a_{j-1} $x<a_j$. In the case $x<a_j$, we shift the elements a_j ..a_{i-1} one position to the right, starting from the right end of the ready subsequence: a_i :=a_{i-1} ; a_{i-1} :=a_{i-2} ; ... a_{j+1} := a_j . We decrease the value of *j* by 1 and check again the condition $x<a_j$. If YES, we insert the current element *x* into the prepared subsequence: a_j :=*x*. Otherwise ($x<a_{j-1}$) move the boundary to the right: *i*:=i+1 (this is the number of the element immediately after the boundary). Move to the beginning.

The process of comparing the current element with the next element of the ready subsequence and shifting (if necessary) the next element to the right is called *sifting in* [6].

A peculiarity of the sorting process: it may turn out that the current element should be placed on the first position. However, this does not create a pair of elements between which the current element should be placed. Such a pair is created artificially – a so-called "barrier" is introduced in the form of *an element* a_0. Based on the comparison relation (see the beginning), we take the value of the barrier to be equal to the current element: $a_0 = x$. In this case, we need to expand the range of indices in the description of the array *A* to *0, ..., n*.

Analysis of *extreme cases*. At the beginning of the process we consider element a_1 as sorted and element a_2 as current, i.e. we take $i_{нач} = 2$. The process ends by inserting element a_n at a suitable place, i.e., $i_{кон} = n$. Thus, the boundary moves between 2 and *n*.

The "Solution" node includes 2 files: "CP and MO" and "PN models".

CP and MO **solution file.** General consideration of the method of solving the sorting problem allows us to formulate a *verbal description of the* algorithm. Based on it, it is quite easy to compose a program in the chosen programming language.

A verbal description of the simple inclusion algorithm

0. A_1 is written *in*to the ready subsequence a, into the input $a_2 ,..., a_n$.

1. $i := 2$.

2. Transfer the element $x=a_i$ from the input subsequence to the ready subsequence so that it remains sorted. To do this:

a) We extend the ready subsequence to the left with the barrier $a_0 := x$.

b) the parameter of the cycle of searching for a suitable place takes the value j:=i-1;

c) as long as $x<a_j$, the element a_j is shifted to the right ($a_{j+1} := a_j$) and *j is* decreased by one (*j*:=j-1);

d) $a_{j+1} := x$.

2. $i := i+1$. If $i \leq n$, then go to step 2, otherwise sorting is finished.

Augmentation. The analysis shows that the following specific MOs can be identified for this method: MO **"Boundary Move"**, MO **"Shift element one position to the right",** MO **"Search for a suitable place"**. The **"Counting cycle"** MO is placed in the general group.

PN-models" file. Note that for this method it is not reasonable to build PN- models for macrooperations due to their simplicity.

Let us consider the complete PN model for sorting by inserts (Fig. 2), constructed by SOA. Here the numbers 0 and 1 near the arc leaving the position denote: 1 – the process continues, 0 – transition to an alternative process.

Table 1 and Table 2 summarize the description and purpose of the PN components.

Based on the structure of the PN, we define scenarios for triggering transitions, covering all possible branches/contours of the structure. We have:

1) p0→t0→p1→t1→p2; 2) NOTp1→t2→p3→t3→p3→t3;

3) NOTp1→t2→NOTp3→t4→NOTp1→t2→NOTp3→t4..... .

The first scenario is trivial and actually checks the possibility of performing a sorting operation. The second scenario describes the loop of searching for a suitable place and shifting a subsequence element. The third scenario describes the loop of inserting element *x* to a suitable place.

The simulation results showed that each of the scenarios is executed, i.e., the network is survivable, and with correct input data, the simulation ends after a finite number of steps.

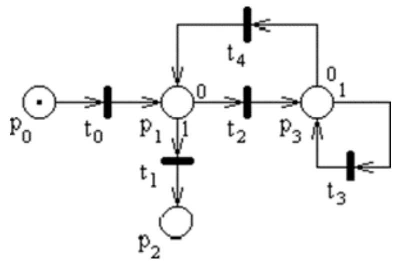

Fig. 2. PN model for sorting by insertion method

Table 1

Positions and their meanings

Position	Significance
p_0	Beginning
p_1	i>n
p_2	The end
p_3	$x<a_j$

Table 2

Transitions and their meanings

Transition	Significance
t_0	input A; i:=2
t_1	output A’
t_2	$x:=a_i$; $a_0:=x$; j:=i–1
t_3	$a_{j+1}:= a_j$; j:=j–1
t_4	$a_j:=x$; i:=i+1

Note that the mandatory alternation of positions and transitions leads to the necessity of either introducing *fictitious* positions and transitions or combining the conditions/operators. In this case, the unification of operators is used.

Conclusion. The paper considers the concept of building a library of problems, their categories, classes and subclasses, solution methods and solution results in a general form. The latter are represented by computational processes, macrooperations and their models in the form of PNs (their fragments).

The architecture and structure of the library are developed. The architecture describes important information for the user. In particular, that all files are available to the user, but without the right to edit them. The structure of the library is a tree with six levels, with level zero being the root of the tree, which is the entrance to the library. The structure reflects the classification of tasks, their classes and subclasses, solution methods and solution files.

We used the previously proposed approach [1], which consists in preliminary modeling of CP by Petri net and transferring the improved CP to the stage of program writing. To realize this approach, we developed a method for building an CP model based on Petri net. The method is implemented in the form of a procedure consisting of the following stages: decomposition of the CP into MOs, mapping of Petri net (PN) fragments to them, assembly of the complete network and its modeling.

This library content provides valuable information that allows for detailed descriptions of problems, methods of solving them, the solutions themselves in general terms, and the corresponding models. In doing so, the work of users and researchers becomes more meaningful, providing a deeper understanding of the library’s content. This content makes the library a powerful tool for managing and analyzing computational processes and their models, greatly increasing accessibility and relevance.

BIBLIOGRAPHY:

1. Method for Constructing the Model of Computing Process Based on Petri Net / N. Komleva et al. *Applied Aspects of Information Technology*. 2019. Vol. 2, no. 4. P. 260–270.

2. Peterson J. L. Petri net theory and the modeling of systems. Englewood Cliffs : Prentice-Hall, 1981. 290 p.

3. Паулін, О. М., Нікітченко, М. І . Вибір засобу моделювання обчислювальних процесів для підвищення якості програмного забезпечення. *Таврійський науковий вісник. Серія: Технічні науки*. 2023. №. 4. С. 69–78.

4. Introduction to Algorithms / V. J. Rayward-Smith et al. *The Journal of the Operational Research Society*. 1991. Vol. 42, no. 9. P. 816.

5. Reingold E. M. Combinatorial algorithms: Theory and practice. Englewood Cliffs : Prentice-Hall, 1977.

6. Wirth N. Algorithms & Data Structures. Pearson Education, Limited, 1986. 288 p.

REFERENCES:

1. Komleva, N., Marulin, S., Nikolenko, A., & Paulin, O. (2019). Method for Constructing the Model of Computing Process Based on Petri Net. *Applied Aspects of Information Technology, 2*(4), 260–270.

2. Peterson, J. L. (1981). *Petri net theory and the modeling of systems*. Englewood Cliffs: Prentice-Hall.

3. Paulin, O. M., Nikitchenko, M. I. (2023). Selection of a computational process modeling tool for improving software quality. *Taurida Scientific Herald. Series: Technical Sciences,* (4), 69-78.

4. Rayward-Smith, V. J., Cormen, T. H., Leiserson, C. E., Rivest, R. L. (1991). Introduction to Algorithms. *The Journal of the Operational Research Society, 42*(9), 816.

5. Reingold, E. M. (1977). *Combinatorial algorithms: Theory and practice*. Englewood Cliffs: Prentice-Hall.

6. Wirth, N. (1986). *Algorithms & Data Structures*. Pearson Education, Limited.