Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2019) 27: 2681 – 2696
© TÜBİTAK
doi:10.3906/elk-1805-82

Early reliability assessment of component-based software system using colored petri net

Amir HOSSEINZADEH-MOKARRAM[*], **Ayaz ISAZADEH**, **Habib IZADKHAH**
Department of Computer Science, Faculty of Mathematical Sciences, University of Tabriz, Tabriz, Iran

Received: 13.05.2018 • **Accepted/Published Online:** 23.01.2019 • **Final Version:** 26.07.2019

Abstract: Assessment of reliability in the early stages of software development from architectural models is one of the major challenges that many studies have addressed in this field in the last decade. The main drawbacks of existing methods are the following: 1) considering equal impact for all parts of the software architecture on system reliability, and 2) inability to determine the contribution of each part of the software in the system failure. This paper introduces the extended version of the colored petri net as an underlying verifiable model to evaluate the reliability of a software system. The proposed model enhances reliability assessment of the software system by measuring the density of failure for each part of the software system, predicting the reliability during execution of a scenario, and estimating the reliability of every structural part of a program, such as loops and conditions. These innovations enable software architects to cost-effectively identify and correct the vulnerable parts of a system in the early stages of software development. The high-level model of the scenario is taken as a UML sequence diagram. Synthesis of the formal model is conducted using an introduced graph called a fragments dependency graph. The practicality of the suggested approach is illustrated by a case study.

Key words: Software architecture, reliability prediction, colored petri net, component-based software system

1. Introduction

Component-based (CB) software development is known as the main solution to overcome the major challenges of software systems. CB systems are formed by the existing independent components, which are related to each other to provide services for users. The evaluation of nonfunctional requirements such as the reliability of each component is one of the research areas today. If we consider that every component of the system has sufficient quality, combining the components together does not usually guarantee the quality of the CB system [1, 2]. Therefore, in addition to evaluating each component, it is also necessary to evaluate how they interact with each other [3, 4]. Software reliability is defined as the probability of failure-free operation of the software under stated conditions and environment for a specified period of time [1]. Many critical tasks are dependent on the reliable execution of complex software systems. The development of reliable software systems with time and budget constraints becomes a challenging objective. This purpose could be achieved to some extent through the early estimation of software failures. This reduces the cost of development because it provides an opportunity to make early corrections during the development process.

Reliability assessment in the design phase is useful when it helps system designers to identify and correct the vulnerable and critical areas [5]. This goal is achieved by early determining of the contribution of all

[*]Correspondence: amir_hosseinzadeh1981@yahoo.com

parts of the software in system failure. These parts could be components, the connection between components, or structural design blocks like loops, conditions, etc. In previous works, only the determination of critical components was taken into account [5–9]. Furthermore, the measuring of failure probability at each point of the system execution was neglected, too. This could increase the accuracy of prediction of the statistical distribution of failures between the parts of a system. Software architecture is usually expressed in a semiformal language such as Unified Modeling Language (UML) or Architecture Definition Language (ADL), which are mastered well by software engineers [1]. However, to prevent ambiguities and for precise verification, it is necessary to convert semiformal models into formal models that have a strong mathematical foundation such as the Markov chain (MC), petri net (PN), and Bayesian model [10–14]. Therefore, first we must introduce an appropriate architectural model, and then we must present a formal model and a synthesis algorithm and finally calculate the reliability of the system and determine the critical parts of it.

We propose an extended definition of CPN, which provides the ability to predict the statistical distribution of failures between the elements of a system during the execution of each scenario early. The proposed model enhances the capabilities of designers to exactly analyze the causes of CB software system failures. For this purpose, in contrast to previous works [15–17], we distinguish the source of failures by assigning a different color to each location where there is a possibility of failure. We can thus determine the contribution of each part of the software in the system failure. As a result, this innovation allows more precise prediction of the reliability of the system, determination the critical parts of the system, and identification and prioritization of test cases.

1.1. Paper outline

The remaining parts of this paper are organized as follows: in Section 2, first we introduce the petri net, and then we review various software architecture description models and reliability evaluation models. In Section 3, our proposed method, including the transformation algorithm of the UML diagram to CPN and the reliability model, will be discussed. In Section 4, our proposed method will be applied in a case study. Finally, in Section 5, we present the conclusion and future work.

2. Background and related works

In this section, we first introduce petri nets. Then we review various software architecture description models and reliability evaluation models. Comparing and evaluating these models, we will express our reasons for choosing UML to describe software architecture at a highly abstract level and CPN to evaluate the reliability of a software system from architecture.

2.1. Petri net

The PN is suitable for quantitative analysis of quality attributes in complex and scenario-based dynamic behavior of software [18]. Modeling of sequential and ordered interactions between objects in a PN is easy and flexible. The mathematical base of a PN is highly powerful [19]. A PN model is closer to a designer's intuition about the software system description than a Markov model. In Markov models, the state space grows much faster than the number of modules being modeled [15]. A PN's graphical representation is simple and composed of four basic elements: places, transitions, arcs, and tokens [18].

Formally, a petri net is a 5-tuple $PN = \{P, T, I, O, M\}$, where:

- $P = \{p_1, \cdots, p_n\}$ is a finite set of places;

- $T = \{t_1, \cdots, t_n\}$ is a finite set of net transition such that

  $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$;
- $I : P \times T \to N$ is an input function that defines directed arcs from places to transitions, and N is the set of nonnegative integers;
- $O : T \times P \to N$ is an output function that defines directed arcs from transitions to places;
- $M : P \to N$ is the initial marking function assigning an initial number of tokens to places.

2.2. Related works

One of the solutions to evaluate software architecture is to create an executable model of the architecture. An executable model of architecture is a formal description of it that can be used to observe and check the final behavior of the software system before implementing the architecture. In other words, using the executable model, the architecture can be evaluated. From this executable model, it can be analyzed to what extent the proposed architecture meets the expected requirements, especially nonfunctional requirements. There are several formal models to display an executable architecture. Some of these models are petri nets, Bayesian models, process algebra, and Markov chains.

These scenarios can be used to improve reliability prediction in the early stages of software system development. CB software reliability assessment based on scenarios is discussed in [6–8]. A UML sequence diagram (SD) specifies scenarios of the processing of operations because of their clear graphical layout, but the semantics presented in the OMG specification for SD only give a basic idea of how the SD should work [11]. Indeed, the SD, as a semiformal model, is ambiguous and lacks precise semantics for its notations [11, 20, 21]. As a result, this chart should be initially converted to an appropriate formal model to prevent different understandings and decipher a unique meaning. Models such as finite state machines, state charts, and Markov chains are completely state-based models rather than scenario-based. They are not able to exactly represent the sequential behavior of software for realizing a scenario that can easily be shown by SD. When a SD is converted to these models, some valuable and detailed information about the dynamic behavior of software during execution of the scenario may be lost. Therefore, the reliability prediction of a scenario during its execution will not be possible. Also, an accurate reliability calculation of structural parts of the program such as loops, alt, and opt could be impossible. Determination of the critical parts of the software system may be imprecise, too.

Yacoub et al. [6] proposed a technique for reliability analysis called scenario-based reliability estimation. Scenarios are converted to proposed graphs, called component dependency graphs (CDGs), as a probabilistic model to calculate the reliability of the entire system. New features of SD 2.x such as fragments opt, loop, and alt are not considered. This approach is not capable of predicting the reliability of each scenario. Rodrigues et al. [8] presented a method to predict software system reliability based on message sequence charts as the architectural model of scenarios. The approach captures the probability of component failure and scenario transition probabilities derived from an operational profile of the system. In this work, the importance of implied scenarios detection is considered to enhance reliability calculation. Singh et al. in [4] proposed a method to estimate the reliability of a software system as a function of its components reliability. They use probabilistic-labeled transition systems as the formal model to specify a scenario.

Yin et al. in [15] presented a method for early-stage system-level software reliability estimation based on PNs. This work divided faults of the system into two categories: inherent faults and propagated faults. To distinguish between the two sets, different 'colors' are assigned to tokens. Reussner et al. in [22] measured the particular component reliability relative to its execution time or the number of its calls by using the proposed PN model. A PN-based framework of an intrusion detection system was introduced in [16]. Azgomi et al. in [23] proposed an approach to reliability evaluation of grid services using the CPN. In their CPN model, tokens carry some attributes, so one can model the transmission or execution time of a subtask as the colors of tokens for each subtask. Mahato et al. in [17] proposed an approach to model the load balanced scheduling and reliability of a grid transaction processing system based on a CPN. They used the CPN to combine a PN with the functional programming language Standard Markup Language. Their model formally describes the process of transaction distribution and execution within the on-demand computing environment.

Prediction reliability during the execution path of a scenario is neglected in previous works and it could help the software engineer to analyze time-dependent failure behavior of the software system precisely. In some approaches, the determination of critical parts is taken into account [6–8]. However, despite our work, where it is possible to determine critical structural design blocks like loops, conditions, and critical components, they only indicate the critical components.

3. Proposed approach

The proposed solution consists of the following steps:

- Present suitable formal model;
- Present conversion method;
- Obtain reliability of the scenario and its parts and determine critical sections.

The following section will explain each of these stages.

3.1. Proposed model

There are many extensions to PNs. Measuring the density of failure for each element of the system, predicting the reliability during execution of the scenario, and estimating the reliability of every structural part of a program such as loops and conditions are our goals in this research. To this end, the location of each failure should be recorded. A PN lacks this ability, but a CPN [24] can be extended for this task. Therefore, a CPN is considered as the base model. In a CPN, colors are added to the objects representing the system elements, thus making it possible to "fold" the representation without losing information about which element executes an action.

A CPN for our purpose is defined formally by a tuple
$CPN = \{P, T, I, O, M, \Sigma, C, W, \lambda\}$, where:

- P, I, O, M have the same meaning as those of a PN;
- $\Sigma = \{c_1, \cdots, c_n\}$ is a finite set of colors and $n = |T|$;
- $C : T \to \Sigma$ is a color function associating a color to a transition;
- T is a finite set of transitions partitioned into two subsets: T_λ, T_W, where transitions $t \in T_\lambda$, $t \in T_W$ are associated with functions λ and W respectively such that $T_\lambda \cap T_W = \emptyset$ and $T_\lambda \cup T_W \neq \emptyset$;

- $\lambda : T_\lambda \to \{0, \cdots, 1\}$ is a function associating a real number r $(0 \leq r \leq 1)$ to $t \in T_\lambda$ to represent a probability of selecting a path (in operators like *opt*, *loop*);
- $W : T_W \to \{0, \cdots, 1\}$ is a function associating a real number r $(0 \leq r \leq 1)$ to $t \in T_W$ to represent a reliability value of each component and path between two components.

In the proposed model, there are two types of transitions: T_W and T_λ. T_W represents the transitions that mark the points where the failure is possible. This type of transition will help us to identify the critical area of the system. It also can be used to measure the reliability during the execution of the scenario. The second type of the transitions T_λ marks the points where a path is divided into more than one path, representing the probability of selection of a path.

In the proposed model, first, each location with a probability of failure in the software is shown by transition $t_i \in T_W$, where it has a unique color as a specific character of t_i. Secondly, tokens are divided into normal and failed groups, displayed with white and nonwhite colors, respectively. During the execution of the scenario, when passing through transitions of T_W, the tokens will take on the color commensurate with their rate of failure. As a result, software failure density will finally be determined at each point of the scenario. In other words, in the initial state of the system, all of the tokens are in the starting place and colored in $c_0 = white$, which indicates nonfailed. We want to be able to determine the exact place of each failure and density of failure at any point of the path, so when tokens go along from the starting to the ending place, in every transition $t_i \in T_W$, depending on its failure rate, nonfailed tokens will be colored with that transition color.

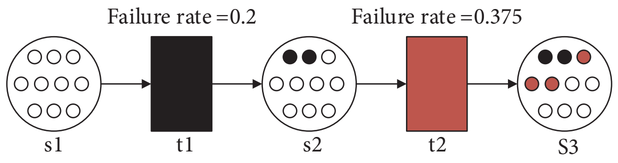


Figure 1. To specify the failure location, a unique color is considered for each transition $t_i \in T_W$ and failed tokens are colored in the same color as where they failed.

For example, consider Figure 1, where the assumptions are as follows:

$c_0 = white$;

$t_1.color = c_1 = black; t_2.color = c_2 = red$;

$t_1.failureRate = 0.2; t_2.failureRate = 0.375$;

$|s_1.tokens| = 10$.

Initially, in state s_1, we have 10 white tokens. In the passage through t_1, two of them change their color to black and eight remain white since the failure rate of t_1 is 0.2 and its color is black. Now, in s_2, we have 8 nonfailed tokens and 2 tokens failed in t_1. The color of t_2 is red and its failure rate is 0.375. Passing through t_2, three white tokens are also colored red because $8 \times 0.375 = 3$. This means that 3 other nonfailed tokens fail, too. Finally, at s_3 we have:

$|s_3.tokens| = 10$;

$|s_3.tokens.white| = 5; //nonfailed$;

$|s_3.tokens.non - white| = 5; //failed$;

$|s_3.tokens.black| = 2; //failed - in - t_1$;

$|s_3.tokens.red| = 3; //failed - in - t_2$.

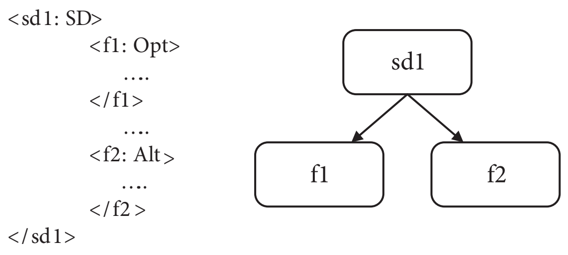


Figure 2. A simple SD and its corresponding FDG (for simplicity, the fragment type is not shown).

There are 10 tokens in the final state (s_3); 5 tokens are nonfailed (white color) and 5 tokens are failed. Thus, the reliability is 0.5. Two tokens are black. This means the contribution of t_1 in the system failure is 0.2. Similarly, the contribution of t_2 is 0.3. By the proposed CPN model, we are not only able to estimate the probability of failure, but we can also determine the contribution of each part of the software in the system failure.

3.2. Conversion method

In the SD, design blocks such as loop and condition are demonstrated by related fragments like loop, opt, and alt; therefore, to calculate the reliability of each design block, the reliability of each fragment should be calculated. A fragment can be integrated into another one, and all fragments are integrated into the main fragment. When fragment B is defined within fragment A, it is said that the latter depends on the former. The dependence model of each scenario is modeled by a new graph, called a fragments dependency graph (FDG). The CPN model of each fragment and eventually the scenario is obtained by performing a bottom-up traversal of the graph. The fragments at the lowest level, i.e. leaves, do not depend on each other at all. On the other hand, the main fragment of the specific scenario is located at the highest level, i.e. the root. In the following, the FDG is formally defined, and then the rules required for the modeling of all fragments are expressed.

3.2.1. Fragment dependency graph

An FDG is a directed graph, where each node represents a fragment used in the UML SD, and an edge represents the definition of a fragment in another one. For example, if fragment b is located within fragment a, an edge will be drawn from a to b. This graph has no cycles. The root node is the main fragment (represents the behavior of the scenario) and other fragments of the scenario are located within it. Thus, the graph is a connected graph. The FDG is actually a tree. Each node contains the fragment ID and type. Formally:

$FDG = (V, E)$,
$V = (ID, Type)$,
$Type = SD|Opt|Alt|Loop$,
$E = (V1, V2)$.

The vertex that starts the edge is called the parent, and the target vertex of the edge is called the child. The order of child edges is important. Figure 2 depicts a SD and its corresponding FDG (fragment type is not shown). The vertex sd1 is the parent and f1 and f2 are child 1 and child 2, respectively. The type of sd1 as root is SD and the types of f1 and f2 are Opt and Alt, respectively.

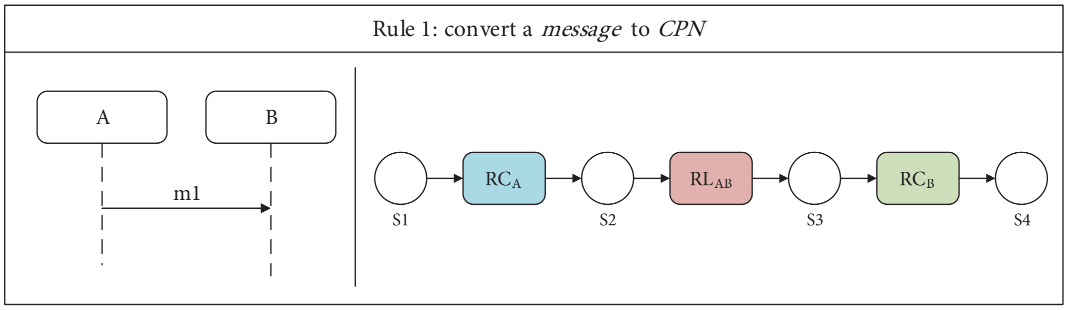


Figure 3. Convert a message to CPN.

3.2.2. Conversion rules

This section presents the rules of conversion of SD operators, i.e. opt, alt, and loop, into the proposed CPN. First we express the rule for conversion of a message between two components. It is assumed that all transitions are synchronous. Note that the provided formulas only help to indicate the outcome attained from the execution of the model and their reliabilities are directly obtained from the model.

1. **Rule 1: Message**

   A message is composed of three parts: sender component C_A, link between two components L_{AB}, and recipient component $(C_B) : M_{AB} = (C_A, L_{AB}, C_B)$. Each part of the message may fail. Given that in the proposed CPN the locations with the probability of failure are depicted by transitions, it is necessary to have three transitions with unique colors and four states to model a message. Rule 1 is shown graphically in Figure 3.

   If F is the probability of a component failure, its reliability will be equal to $R = 1 - F$. Thus, the reliability of the message can be obtained as (Reliability of Message M_{AB}):

$$RM_{AB} = RC_A * RL_{AB} * RC_B, \tag{1}$$

   where:

   $RC_A = 1 - FC_A \; //FC_A : Failure \; Rate \; of \; C_A,$
   $RL_{AB} = 1 - FL_{AB} \; //FL_{AB} : Failure \; Rate \; of \; L_{AB},$
   $RC_B = 1 - FC_B \quad //FC_B : Failure \; Rate \; of \; C_B.$

2. **Rule 2: Serial**

   If there are n consecutive messages or fragments, the CPN can be derived from the sequential connection of them and overall reliability is obtained using the following equation:

$$R = \Pi_{i=1}^{n} R_i, \tag{2}$$

   where:

   $R :< R_1, R_2, \cdots, R_n >, R_i \in M \cup T, (1 \leq i \leq n),$
   $M : set\,of\,messages = M_1, M_2, \cdots, M_m,$
   $T : set\,of\,operators = T_1, T_2, \cdots, T_m.$

3. **Rule 3: Opt**

   The operator *opt* indicates optional behavior, which has an operand (O_{opt}) executing with P_{Opt} probability, including one or several messages or other operators. Rule 3 is shown visually in Figure 4a, where the operand is a simple message. This operator is similar to the structure of *if* in the programming languages. According to the CPN model, the reliability of the opt operator is obtained as follows:

$$R = (1 - P_{opt}) + (P_{opt} * R_{opt}), \tag{3}$$

   where:

   R_{opt}: Reliability of operand.

4. **Rule 4: Alt**

   The operator *alt* indicates alternative behavior. This operator can have one or more operands with different probabilities such that only one of them can be run. Rule 4 is shown visually in Figure 4b. This operator is similar to the structures *if / (else if)* / else* and *switch case* in programming languages.

   An *alt* operator includes n operands of $O_1, \cdots, O_n$. If execution probabilities of operands are respectively $P_1, \cdots, P_n$ with the reliability of $R_1, \cdots, R_n$, the alt operator reliability can be obtained as follows:

$$R = \Sigma_{i=1}^{n} P_i * R_i. \tag{4}$$

   Figure 5 shows how the reliability is obtained when $P_{t1} = 0.8$ and $P_{t1} = 0.2$ and $FR_{t1} = 0.25$ and $FR_{t2} = 0.5$. In the final state, 7 tokens are white (nonfailed), so the reliability of the fragment is 0.7. Also, 2 tokens are black and 1 token is red so the contributions of t1 and t2 in the system's failure are 0.2 and 0.1, respectively. Note that regardless of the structure Alt, the failure rates of t1 and t2 are 0.25 and 0.75.

5. **Rule 5: Loop**

   Computer programs rely heavily on repetition to perform any significant operations. The operator *loop* indicates repetitive behavior. This operator has an operand and is applied in two ways. In the first case, a loop operator consists of an operand O_{loop} that will be executed with the possibility of p_{loop}, and the behavior will be repeated. According to the CPN model shown in Figure 4c, if the reliability of O_{loop} is R_{loop}, the reliability of the loop is finally obtained as follows:

$$R = \frac{1 - P_{loop}}{1 - (P_{loop} * R_{loop})}. \tag{5}$$

6. **Rule 6: Loop(n)**

   In the second type of loop, for which the conversion rule is shown in Figure 4d, a loop operator consists of an operand (O_{loop}) that will be executed n times. As a result, its simulation is done with n copies of the operand. If the reliability of O_{loop} is R_{loop}, then the reliability of the loop operator is obtained as follows:

$$R = R_{Loop}^{n}. \tag{6}$$

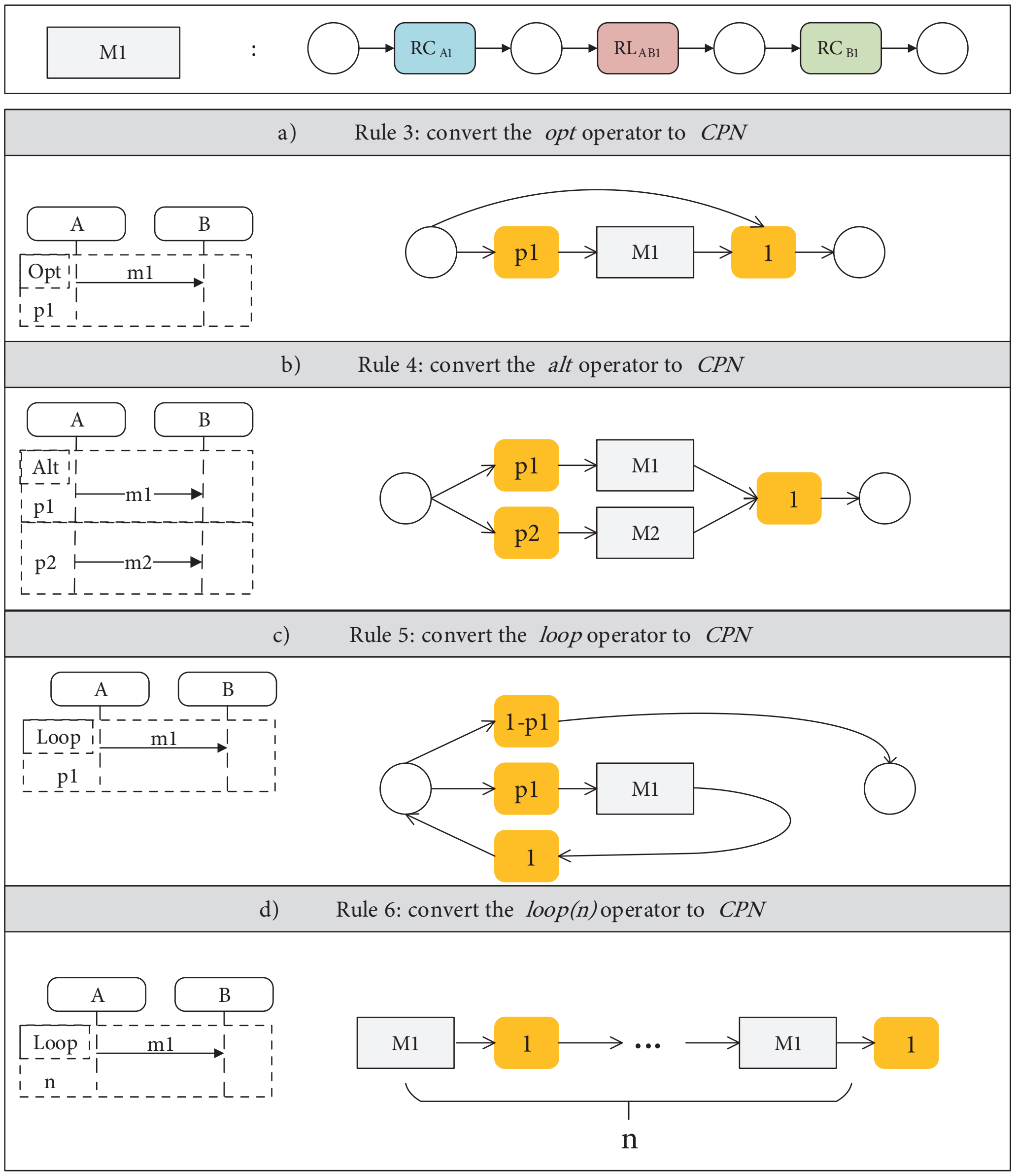

M1
:
RC A1
RL AB1
RC B1
a) Rule 3: convert the *opt* operator to *CPN*
A
B
Opt
m1
p1
p1
M1
1
b) Rule 4: convert the *alt* operator to *CPN*
Alt
m1
p1
p2
m2
M2
c) Rule 5: convert the *loop* operator to *CPN*
Loop
1-p1
d) Rule 6: convert the *loop(n)* operator to *CPN*
n
...

Figure 4. Conversion rules of operators opt, alt, and loop to the proposed CPN (the picture on the left shows the model of a fragment in UML SD and the right image shows the related CPN).

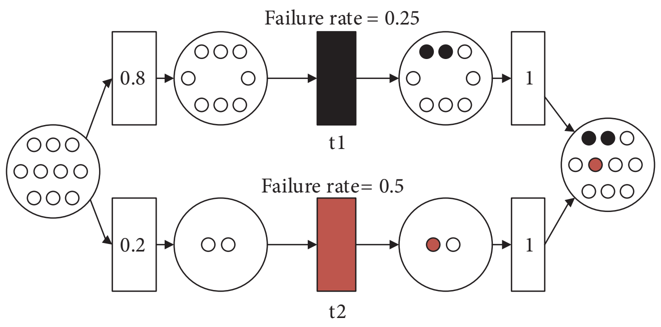


Figure 5. An example of CPN modeling of an operator *Alt*.

Table 1. Failure rates of components.

Component	C1	C2	C3	C4	C5
Reliability	0.007	0.101	0.007	0.011	0.003

Table 2. Failure rates of links.

	C1	C2	C3	C4	C5
C1	0	0.019	0.006	0.008	0.002
C2	0.019	0	0.004	0.004	0.006
C3	0.011	0.011	0	0.003	0.005
C4	0.008	0.019	0.009	0	0.001
C5	0.002	0.022	0.011	0.001	0

3.3. Extracting the CPN of the scenario

The CPN of a scenario is obtained by postorder traversal of the FDG of the scenario. Every time a node is visited, its CPN is extracted using rules 1 to 6. If the node is a leaf, it means that no other fragment is defined in the node. If the node is not a leaf, other fragments are certainly defined inside it. Since the algorithm is a depth-first approach, when visiting a parent node, the child node CPN models are already extracted. By combining children models with the parent CPN model, the full parent CPN model can be achieved. The last node that will be visited is the root node that represents the full model of the scenario.

3.4. Obtaining the reliability of the scenario and its parts

The reliability of the scenario and its parts can be obtained directly from the extracted CPN model of the scenario. In this model, any location where there is a probability of failure is modeled by a specific transition $t_i \in T_W$, which has a unique feature (or color). Also, tokens are divided into normal (white) and failed (nonwhite). In the initial state of the system, all of the tokens are white, but during the execution of the scenario, when passing through transition t_i, some normal tokens will take on the color of t_i depending on its rate of failure. The ratio of white tokens (or nonfailed tokens) to the overall tokens in the final state of the CPN indicates the reliability of the scenario. In addition, the frequency of each color shows the probability of failure in each part during the execution of the scenario. Sorting, based on the frequency of each color, can be used to measure the critical parts of the system.

4. Case study

In this section, we consider the scenario presented by sequence diagram sd1 according to Figure 6. Based on the given information about the system, first we extract the CPN of the scenario, and then we analyze the results.

4.1. Problem hypotheses

It is assumed that the prerequisite data are available based on historical data from past experience or predicted by analyzers. Failure rates of components and links between the components are according to Tables 1 and 2. The probabilities considered for the necessary operators are given in Table 3.

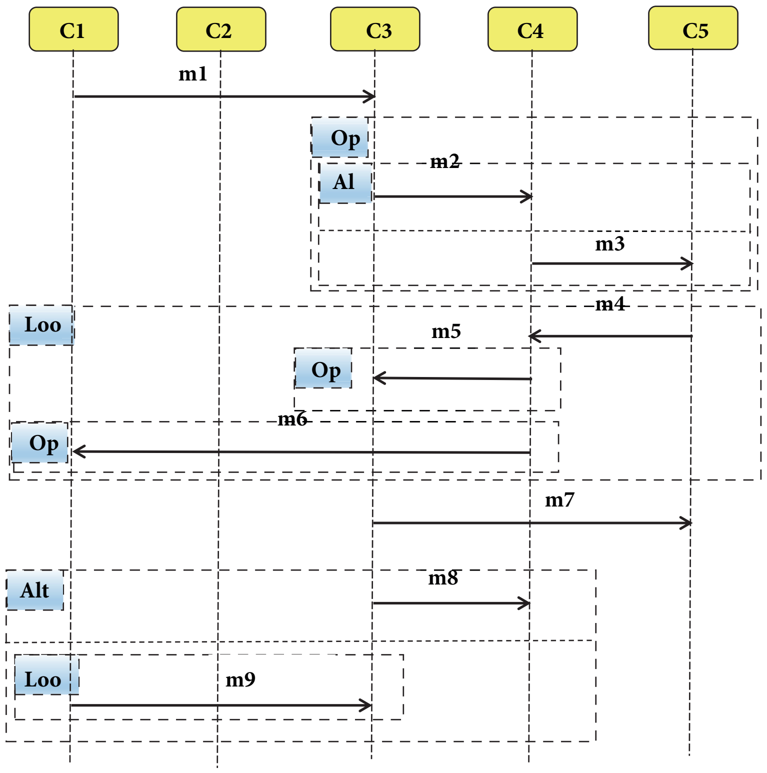


Figure 6. UML SD of the scenario.

Table 3. Assumptions for the scenario.

Operator	Opt1	Loop1	Alt1	Opt2	Opt3	Loop2	Alt2
Value	0.5	0.4	0.5, 0.5	0.2	0.6	(n=2)	0.3, 0.7

4.2. Synthesizing of CPN

The CPN of an sd1 can be derived from its FDG traversal in the bottom-up approach. Figure 7 represents the extracted FDG, which consists of $alt1, opt2, opt3, loop2, opt1, loop1, alt2$, and $sd1$ fragments. The CPN obtained for the scenario is represented in Figure 8; for simplicity, tokens and the unique color assigned for each transition are not shown.

4.3. Analysis of extracted model

In the provided model, the latest state, called the final state, includes tokens with different colors, which indicate the share of each part (including components and links) in the system failures. Predicted failure rates of components and links between them in the scenario are represented in Table 4. Sorting, based on the frequency of each color, can be used to measure the critical sections. As shown in Figure 9, while initial failure rates of C4 and L43 (link from C4 to C3) are highest, the contributions of components C1, C3, and C4 and link L13 (link from C1 to C3) in the scenario failure are greater than the other parts. The ratio of white tokens (or nonfailed tokens) to the overall tokens in the final state of the CPN indicates the reliability of the scenario. The predicted reliability of sd1 is 0.903. Figure 10 represents the statistical distribution of failures between the parts

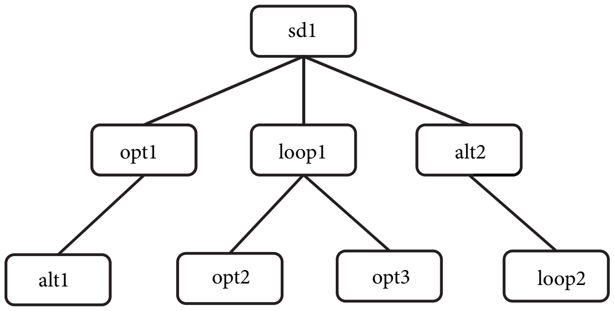


Figure 7. FDG of the scenario.

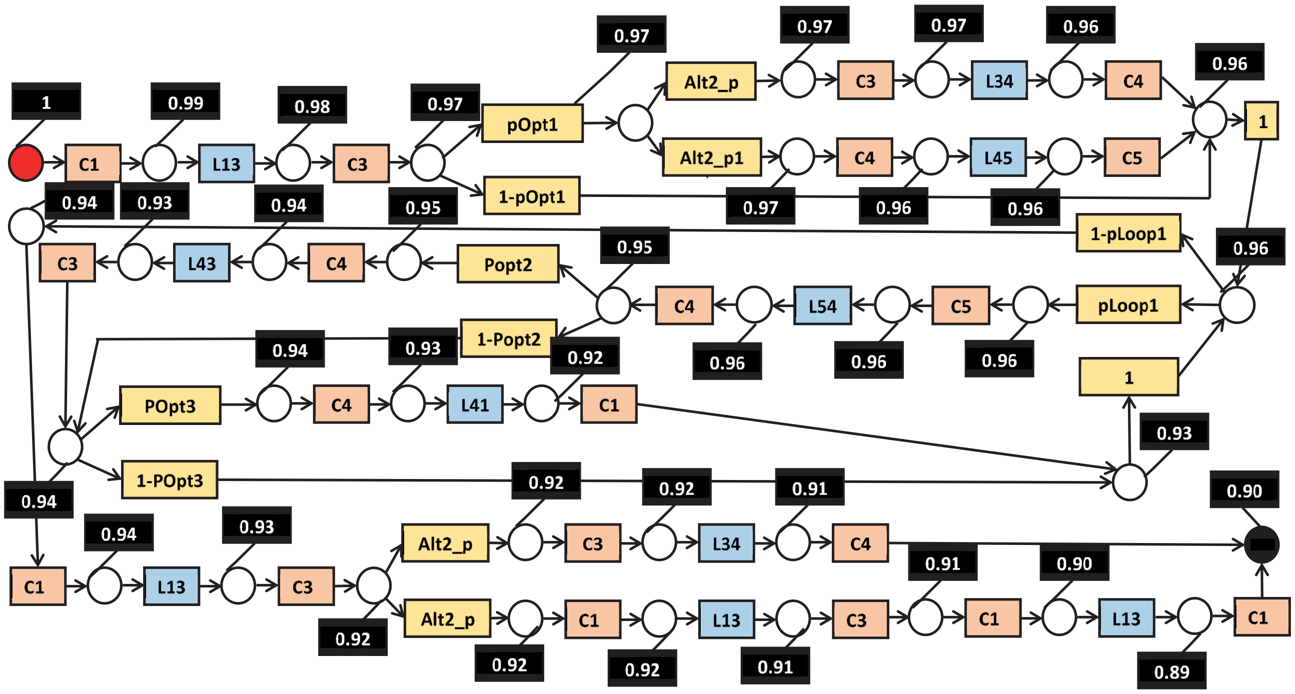
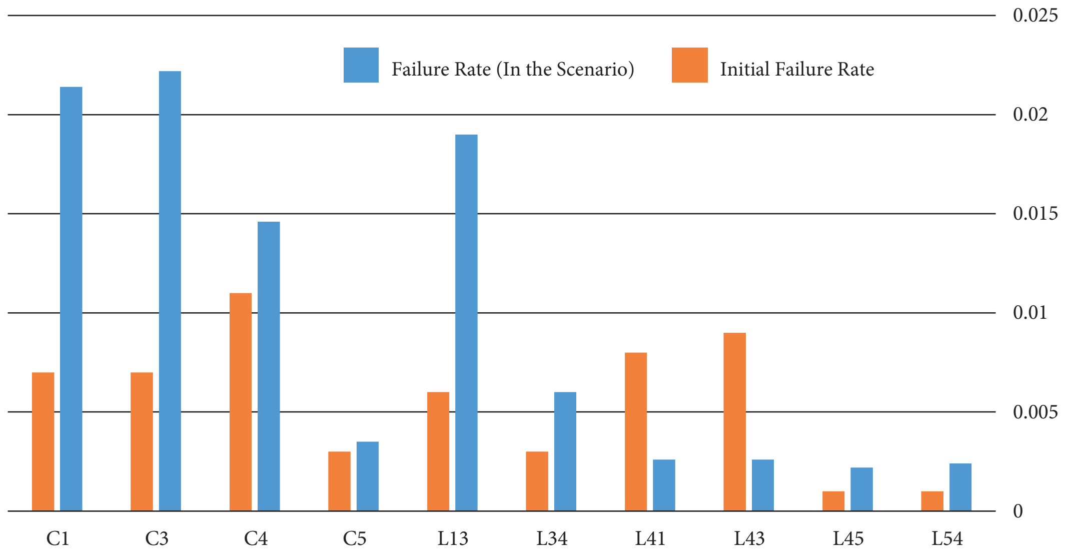


Figure 8. Extracted model of the scenario and predicted reliability at each point of execution of the scenario.

Table 4. Failure rates of links.

C1	C2	C3	C4	C5
0.0214	0	0.0222	0.0146	0.0035
L11	L12	L13	L14	L15
0	0	0.0190	0	0
L21	L22	L23	L24	L25
0	0	0	0	0
L31	L32	L33	L34	L35
0	0	0	0.0060	0
L41	L42	L43	L44	L45
0.0026	0	0.0026	0	0.0022
L51	L52	L53	L54	L55
0	0	0	0.0024	0

of scenario sd1. The predicted reliability of the fragments is obtained based on their CPN models, as shown in Figure 11. The results help to identify critical structural sections of the program such as loops and conditions. Using the CPN obtained for a scenario, the decline of the reliability of the scenario during its execution can be calculated. Figure 8 represents the result for sd1. For simplicity, the different colors of error areas are not shown.



	C1	C3	C4	C5	L13	L34	L41	L43	L45	L54
Failure Rate (In the Scenario)	0.0214	0.0222	0.0146	0.0035	0.019	0.006	0.0026	0.0026	0.0022	0.0024
Initial Failure Rate	0.007	0.007	0.011	0.003	0.006	0.003	0.008	0.009	0.001	0.001

Figure 9. Comparison of initial failure rates of components and links and predicted contribution of them in the failure of the scenario sd1.

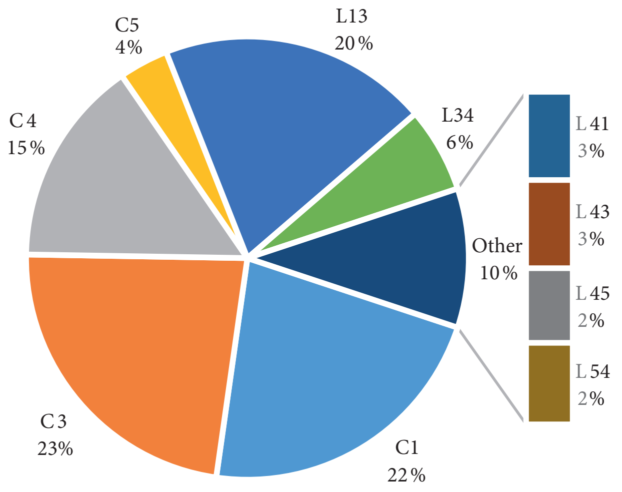


Figure 10. The statistical distribution of failures between the parts of scenario sd1.

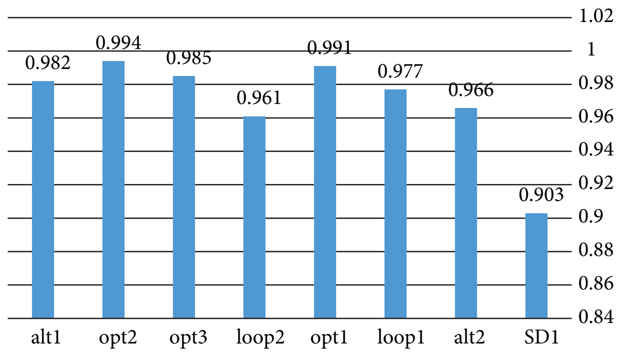


Figure 11. The reliability of the fragments.

5. Conclusions

Assessing the reliability in the early stages of the software development process based on architectural models plays a major role in the success of component-based software systems. In the literature, early determination of the contribution of all parts of the software in the system failure and measuring of failure probability at each point of the system execution are neglected. We presented a method to improve the reliability of component-based software systems using proposed colored petri nets. Initially, the complete internal behavior of a scenario is taken in the form of a UML SD and then a CPN is synthesized as a verifiable formal model. Conversion into a CPN is conducted by traversing an introduced new graph called a FDG. Also, the necessary rules to model the data control operators including serial, opt, loop, and alt are presented.

Unlike related works, in the offered model, a unique color is given to any location where there is a risk of failure (represented by special transitions in the suggested CPN). If a failure occurs, depending on the probability of failure, one or more tokens are colored in its color and thus it becomes possible to calculate the density of failure for each component or link between components. Thus, in addition to assessing the reliability of the system, our method is also capable of assessing the reliability of any structural part of the program such as loops and conditions, which are depicted by segments. The results enable software engineers to identify and correct the vulnerable and critical areas with low cost. The proposed method is suitable for applications whose analysis is based on valid scenarios with sequence diagrams and the failure rates of components and link between them are available.

The following topics can be considered as future works based on this research:

1. We only considered synchronous messages, while asynchronous messages also have an important role in describing the behavior of distributed systems [5, 25]. In addition, we only provided rules to convert opt, loop, and alt operators of sequence diagrams to the proposed CPN model. In the future, we intend to cover other operators such as par. Also, in this paper, the theoretical aspects of the proposed model are presented. We intend to provide the appropriate tools for the proposed model.

2. In the presented method for reliability evaluation, we did not consider some important and effective parameters for system reliability such as error propagation and fault tolerance [26–30]. By considering these parameters in the proposed approach, the system reliability can be assessed more accurately.

References

[1] Pressman RS. Software Engineering: A Practitioner's Approach. 7th ed. New York, NY, USA: McGraw-Hill, 2010.

[2] Singh L K, Tripathi AK, Vinod G. Software reliability early prediction in architectural design phase: overview and limitations. Journal of Software Engineering and Applications 2011; 4 (3): 181-186. doi: 10.4236/ jsea.2011.43020

[3] Sinha S, Goyal NK, Mall R. Early prediction of reliability and availability of combined hardware-software systems based on functional failures. Journal of Systems Architecture 2019; 92: 23-38. doi: 10.1016/j.sysarc.2018.10.007

[4] Singh LK, Vinod G, Tripathi AK. Reliability prediction through system modeling. ACM SIGSOFT Software Engineering Notes 2013; 38 (6): 1-10. doi: 10.1145/2532780.25328012

[5] Goševa-Popstojanova K, Trivedi K. Architecture-based approach to reliability assessment of software systems. Performance Evaluation 2001; 45 (2-3): 179-204. doi: 10.1016/S0166-5316(01)00034-7

[6] Yacoub S, Cukic B, Ammar H. A scenario-based reliability analysis approach for component-based software. IEEE Transactions on Reliability 2004; 53 (4): 465-480. doi: 10.1109/TR.2004.838034

[7] Ali A, Jawawi DN, Isa MA. Modeling and calculation of scenarios reliability in component-based software systems. In: IEEE 2014 Malaysian Software Engineering Conference (MySEC); Langkawi, Malaysia; 2014. pp. 160-165.

[8] Rodrigues G, Rosenblum D, Uchitel S. Using scenarios to predict the reliability of concurrent component-based software systems. In: Springer 2005 International Conference on Fundamental Approaches to Software Engineering; Berlin, Germany; 2005. pp. 111-126.

[9] Dobrica L, Niemela E. A survey on software architecture analysis methods. IEEE Transactions on Software Engineering 2002; 28 (7): 638-653. doi: 10.1109/TSE.2002.1019479

[10] Karimpour J, Isazadeh A, Izadkhah H. Early performance assessment in component-based software systems. IET Software 2013; 7 (2): 118-128. doi: 10.1049/iet-sen.2011.0143

[11] Micskei Z, Waeselynck H. The many meanings of UML 2 Sequence Diagrams: a survey. Software and Systems Modeling 2011; 10 (4): 489-514. doi: 10.1007/s10270-010-0157-9

[12] Choppy C, Klai K, Zidani H. Formal verification of UML state diagrams: a petri net based approach. ACM SIGSOFT Software Engineering Notes 2011; 36 (1): 1-8. doi: 10.1145/1921532.1921561

[13] Wiper MP, Palacios AP, Mareen J. Bayesian software reliability prediction using software metrics information. Quality Technology and Quantitative Management 2012; 9 (1): 35-44. doi: 10.1080/16843703.2012.11673276

[14] Meedeniya DA. Correct model-to-model transformation for formal verification. PhD, University of St Andrews, St Andrews, UK, 2013.

[15] Yin ML, Hyde CL, James LE. A petri-net approach for early-stage system-level software reliability estimation. In: IEEE 2000 Annual Reliability and Maintainability Symposium, International Symposium on Product Quality and Integrity; Los Angeles, CA, USA; 2000. pp. 100-105.

[16] Gou Z, Ahmadon MA, Yamaguchi S, Gupta BB. A Petri net-based framework of intrusion detection systems. In: IEEE 2015 Global Conference on Consumer Electronics; Osaka, Japan; 2015. pp. 579-583.

[17] Mahato DP, Singh RS. Load balanced scheduling and reliability modeling of grid transaction processing system using coloured Petri nets. ISA Transactions 2019; 84: 225-236. doi: 10.1016/j.isatra.2018.08.022

[18] Peterson JL. Petri nets. ACM Computing Surveys 1977; 9 (3): 223-252. doi: 10.1145/356698.356702

[19] Volovoi V. Modeling of system reliability Petri nets with aging tokens. Reliability Engineering & System Safety 2004; 84 (2): 149-161. doi: 10.1016/j.ress.2003.10.013

[20] Bowles J, Meedeniya D. Formal transformation from sequence diagrams to coloured petri nets. In: IEEE 2010 Asia Pacific Software Engineering Conference; Sydney, Australia; 2010. pp. 1216-225.

[21] Staines TS. Transforming UML sequence diagrams into Petri Net. Journal of Communication and Computer 2013; 10 (1): 72-81.

[22] Reussner RH, Schmidt HW, Poernomo IH. Reliability prediction for component-based software architectures. Journal of Systems and Software 2003; 66 (3): 241-252. doi: 10.1016/S0164-1212(02)00080-8

[23] Azgomi MA, Entezari-Maleki R. Task scheduling modelling and reliability evaluation of grid services using coloured Petri nets. Future Generation Computer System 2010; 26 (8): 1141-1150. doi: 10.1016/j.future.2010.05.015

[24] Jensen K, Kristensen L, Wells L. Coloured petri nets and CPN tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer 2007; 9 (3-4): 213-254. doi: 10.1007/s10009-007-0038-x

[25] Ciardo G, Muppala J, Trivedi K. Analyzing concurrent and fault-tolerant software using stochastic reward nets. Journal of Parallel and Distributed Computing 1992; 15 (3): 255-269. doi: 10.1016/0743-7315(92)90007-A

[26] Cheraghlou M, Khadem-Zadeh A, Haghparast M. A survey of fault tolerance architecture in cloud computing. Journal of Network and Computer Applications 2016; 61: 81-92. doi: 10.1016/j.jnca.2015.10.004

[27] Qiu X, Ali S, Yue T, Zhang L. Reliability-redundancy-location allocation with maximum reliability and minimum cost using search techniques. Information and Software Technology 2017; 82: 36-54. doi: 10.1016/j.infsof.2016.09.010

[28] Hu H, Jiang CH, Cai KY, Wong WE, Mathur AP. Enhancing software reliability estimates using modified adaptive testing. Information and Software Technology 2017; 55 (2): 288-300. doi: 10.1016/j.infsof.2012.08.012

[29] Singh LK, Vinod G, Tripathi AK. Impact of change in component reliabilities on the system reliability estimation. ACM SIGSOFT Software Engineering Notes 2014; 39 (3): 1-6. doi: 10.1145/2597716.2631647

[30] Zio E. Reliability engineering: Old problems and new challenges. Reliability Engineering & System Safety 2009; 94 (2): 125-141. doi: 10.1016/j.ress.2008.06.002