I.J. Image, Graphics and Signal Processing, **2023, 6, 81-93**
Published Online on December 8, 2023 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijigsp.2023.06.07

Modeling and Simulating Mutual Testing in Complex Systems by Using Petri Nets

Viktor Mashkov
University J. E. Purkyne Ceske mladeze 8, Usti nad Labem 40096, Czech Republic
E-mail: Viktor.Maskov@ujep.cz
ORCID iD: https://orcid.org/0000-0001-9817-3388

Volodymyr Lytvynenko*
Kherson National Technical University Berislavskoye Shosse St. 24, Kherson, 73008, Ukraine
Email: immun56@gmail.com
ORCID iD: https://orcid.org/0000-0002-1536-5542
*Corresponding Author

Irina Lurie
Ben-Gurion University of Negev, David Ben Gurion Blvd 1, Beer Sheva, 8410501, Izrael
Email: lurieira@gmail.com
ORCID iD: https://orcid.org/0000-0001-8100-1846

Received: 15 June, 2023; Revised: 06 August, 2023; Accepted: 15 October, 2023; Published: 08 December, 2023

Abstract: The paper tackles the problem of performing mutual testing in complex systems. It is assumed that units of complex systems can execute tests on each other. Tests among system units are part of system diagnosis that can be carried out both before and during system operation. The paper considers the case when tests are executed during system operation. Modelling and simulating mutual tests will allow evaluation of the efficiency of using joint testing in the system. In the paper, the models that use Petri Nets were considered. These models were used for simulating the execution of tests among system units. Two methods for performing such simulations were evaluated and compared. Recommendations for choosing a more appropriate way were made. Simulation results have revealed minor model deficiencies and possible implementation of mutual testing in complex systems. Improvement of the model was suggested and assessed. A recommendation for increasing the efficiency of system diagnosis based on joint testing was made.

Index Terms: Complex system, mutual testing, modelling, simulation, Petri Nets

1. Introduction

Nowadays, complex systems are applied in nearly all spheres of human activities, such as production (manufacturing), transportation, communication, environment monitoring, etc. For example, vehicles, aeroplanes, trains, and even private cars are equipped with and controlled by complex systems [1, 2]. Examples of complex systems are:

- sensor networks [3, 4];
- multi-agent systems;
- ad-hoc wireless networks of consumer devices;
- intelligent matter systems [5];
- many-core processors;
- a group of cooperating robots, etc.

The malfunction of these complex systems may directly impact safety and cause severe consequences on human life and the environment. Given this, checking complex system state (i.e., error detection) plays a significant role during the system's life span. Error detection can be performed before or after regular or during normal system operation (so-called concurrent error detection). Error detection can be achieved by using different methods. One option is that system units can be tested by an external tester (or diagnoster). Another option consists of using self-testing [6, 7]. This kind of

testing can be conducted either by a particular system unit performing diagnosis tasks only or by system units themselves (i.e., by mutual testing [8, 9, 10]).

Mutual testing has several advantages, such as

- ability to perform testing for an extended period of system autonomous work;
- ability to provide fault-tolerance of testing procedures;
- the ability to perform system diagnosis correctly even when some system units have low reliability and could fail;

To realize these advantages of mutual testing (i.e., to deploy joint testing in complex systems efficiently), it is necessary to verify if it is possible to implement mutual testing in complex systems efficiently. This can be done with the help of appropriate modelling and simulation of joint testing in complex systems.

Developing the appropriate model of mutual testing is a problem that can be solved by using one of the following facilities:

- Multi-agent platform;
- programming language like Python;
- Petri Nets;
-agent UML;
- Erlang/OTP platform, etc.

Among the existing facilities, we chose the Petri Nets and developed the corresponding model of mutual testing. Nowadays, models of joint testing that use Petri Nets exist and are exploited to simulate test execution in the systems [14]. It is worth noting that these models do not consider some system states and parameters of the testing procedure. Given this, the task arises to modify the existing models by way of their extension (i.e., by including new system states and by taking into account new testing parameters).

Another problem consists of conducting simulations by using the developed model. Such simulation can be performed on software programs or using special tools (e.g., SHARPE [16]). The choice of the appropriate simulation is a task that should be solved by considering system parameters (e.g., number of system units, system state transition/change parameters, etc.).

A simulation that uses software programs (also called mathematic simulation) was considered in [15]. In this paper, we intend to evaluate some parameters of such simulation and determine their optimal values.

The rest of the paper is organized as follows. Section 2 presents the related works. Section 3 offers the basics of Petri Nets. In Section 4, mutual testing is considered as part of system diagnosis. Section 5 shows how Petri Nets can be used for modelling tests among system units. In Section 6, a simulation of mutual testing performed by using the tool SHARPE is considered. In Section 7, simulation conducted by using software programs is considered and assessed. The conclusion summarizes the novelty of the performed research and evaluates the obtained results.

2. Related Works

Since the task of technical diagnosis of complex systems is topical, many researchers work in this scientific field. In particular, a number of works have been devoted to the theory and fundamentals of system diagnosis that use the results of tests performed by system components (so-called, system-level self-diagnosis), such as those presented in [19]. In this paper, the authors propose the automatic fault diagnosis for systems with multiple faults. Diagnosis is based on the decomposition of the system on identifiable components (units). The authors proposed the optimal testing assignment for diagnosis procedures. They introduced a novel approach to digital system diagnosis, yielding valuable results and potential contributions to the field, particularly focusing on special testing assignments like single-loop systems and highlighting open research problems in specific system topologies.

Paper [20] discusses a probabilistic approach to system diagnosis, including probabilistic models of diagnosis test results and methods for estimating system state. It also introduces a test selection method using Bayesian statistics, contributing significantly to the application of probabilistic methods in this field.

Work [21] focuses on developing a theory of fault diagnosis in multiprocessor systems, emphasizing their complexity compared to single-processor systems. The paper claims that this theory can serve as the basis for creating efficient and reliable diagnosis algorithms. It is crucial for identifying faults in operational multiprocessor systems.

In the paper [22], the authors address the consensus problem. They emphasize its significance for fault-tolerant computing which uses synchronization, communication, resource allocation, task scheduling, diagnosis, and reconfiguration. They focus on a specific aspect of the consensus problem, such as how to achieve the correct results in the presence of faulty processors. They discussed the history of two dominant approaches—system diagnosis and Byzantine consensus. The paper also explored the practical implementation of these concepts in real systems and proposed directions for future research with an emphasis on the practicality and applicability of the proposed method in fault-tolerant parallel and distributed systems.

The paper [23] discusses the importance of system-level diagnosis for modern reliable computing systems and the challenges and implications of complex system diagnosis. The authors showed how to improve the reliability and performance in parallel and distributed computing environments.

The paper [24] introduces the model and detection procedure for intermittent faults in computer systems. The paper presents a valuable contribution to the field of fault tolerance by addressing the challenging issue of intermittent faults. It proposes a method for their detection that has wide implications for enhancing the reliability of computing systems.

In the paper [25], the authors S. Mallela and G. Masson, made focused on identifying intermittent faults in complex systems by using system-level self-diagnosis. This research contributes to the field of fault diagnosis and fault tolerance by addressing the challenging issue of intermittent faults and offering insights and solutions to enhance the reliability and performance of computer systems.

The paper [26] explores the diagnosis of intermittently faulty components of the system. The research delves into techniques and methodologies for identifying and addressing intermittent faults at the system level. The paper makes a valuable contribution to the field of fault diagnosis and system reliability. Another part of the work is devoted to practical tasks (application) of mutual testing (i.e.testing performed by system units on other system units).

The paper [3] presents cutting-edge methods for sensors within intelligent integrated sensing systems to self-detect and to diagnose faults autonomously. It aims to enhance the reliability and performance of considered systems. This paper presents advanced methods for autonomous fault detection and diagnosis by sensors as part of intelligent integrated sensor systems to improve their reliability and performance. This research plays a pivotal role in advancing sensor technology, ensuring that sensors operate with enhanced integrity and functionality within complex sensing systems.

In [6] introduces a new approach to self-testing in multimodule systems. It focuses on optimising check-connection structures to enhance self-testing efficiency. This research gives valuable insight into engineering simulation by facilitating the development of reliable and self-testing multimodule systems applicable across various domains.

In [7] presents an in-depth interpretation of the system-level self-diagnosis problem. The paper discusses a comprehensive understanding of mathematical modelling and computational approaches that allow for enhancing system reliability. This work significantly contributes by providing valuable insights into the interpretation and computational aspects of system-level self-diagnostics applicable across diverse fields.

The paper [8] addresses the assignment connection problem in diagnosable systems. This research investigates the optimal method for assigning connections for these systems essential for fault diagnosis. The study contributes valuable insight into the field of electronic computers by addressing the fundamental problem of designing reliable and diagnosable systems.

In [9], the authors research the consensus problem in fault-tolerant computing by discussing its significance and challenges to the reliability of distributed systems. This paper offers valuable insight and an up-to-date overview of consensus algorithms, which serve as a fundamental resource for researchers and practitioners in distributed computing.

In [10], authors investigate mutual tests for diagnosing spacecraft avionics by focusing on designing collaborative fault detection algorithms. Such diagnosis allows for enhancing system reliability and fault tolerance. This research can be helpful in developing robust avionics architectures and algorithms applicable in the aerospace industry.

Many papers and surveys presented the theory of Petri nets, modelling using Petri nets and the application of Petri nets to practical problems.

The paper [11] presents the theoretical foundation for understanding and modelling communication processes using automata. It contributes significantly to the field of computer science and automation theory.

The book [12] comprehensively studies Petri net theory and its applications to system modelling. This work is a valuable resource for researchers, engineers and students interested in understanding and applying Petri nets to modelling complex systems.

Paper [13] discusses the possible use of Petri nets as a modelling and analysis tool in the context of communication networks. It gives researchers and practitioners in computer science and network engineering methodologies.

The paper [17] explores the application of Petri nets as a representation framework for modelling metabolic pathways. This research investigates the use of Petri nets to describe and analyse the intricate biochemical processes within cells, offering valuable insights into the field of computational biology and its potential contributions to understanding metabolic phenomena.

In [18], authors consider Petri nets as a tool for modelling coalition formation processes. This study examines the use of Petri nets to represent and analyse the dynamics of coalition formation. The results can contribute to modelling complex interactions between entities in various domains.

In [14], authors explored the use of Petri Nets to model self-testing in many-core processors when tests are performed randomly. This research demonstrates how Petri Nets can be used to represent self-testing processes in complex many-core architectures. The results of this research can be used to enhance electronic testing mechanisms for advanced processors.

In [15], Petri Nets as a modelling tool for system reliability assessment were explored. The paper considered the principles and methodologies which are used for complex system analysis. It serves as an essential resource for researchers and practitioners in system reliability assessment.

In [16], the authors introduce how to use the SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator) tool for reliability modelling and analysis in complex systems. They contributed to reliability engineering methodologies. This work serves as a significant resource for researchers and practitioners in system reliability, advancing analytical approaches.

The analysis of the cited works allows us to conclude that the authors in their research considered practically all significant problems in the corresponding research areas. However, some issues require more detailed investigation and verification. It concerns some system states and parameters of the testing procedure. Consequently, it can be concluded that the existing modelling and simulation methods have limitations and need to be improved to include new system states and testing parameters.

3. Introduction to Petri Nets

Petri Nets were designed by Carl Adam Petri in 1962 in his Ph.D. thesis 'Kommunikation mit Automaten' [11]. Adam Petri aimed to represent the system states and their changes in a graphical form, particularly, in the form of a graph. Bipartite-directed graph G=G(P,T) was chosen for this purpose (see Fig. 1).

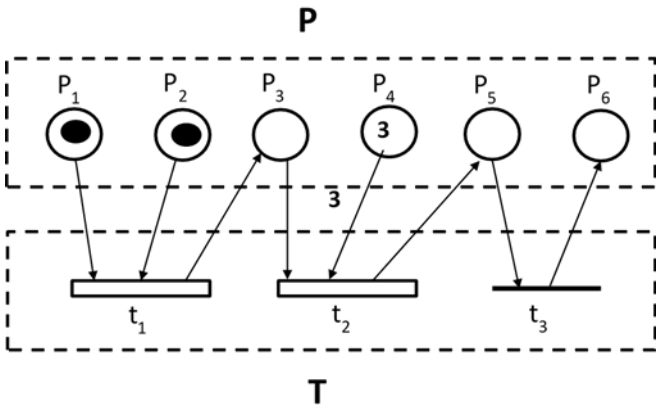


Fig. 1. Example of Petri Net

In the given case, a bipartite-directed graph has two sets of vertices denoted as P={pi}, i=1,..,n and T={tj}, j=1,..,m. The first set took the name of the places, and the second one the transitions. Since locations P and T are the sets of vertices of a bipartite graph, they can be connected by directed edges. Edges can refer both to vertices which belong to set P and to set T.

It should be noted that any two vertices of set P cannot be connected. It also applies to any vertices of set T. Elements of set P called places can be considered system states or conditions. Similarly, elements of set T, which are called transitions, can be considered as actions or events.

For each transition t, it is possible to determine places connected to this transition. These places represent pre-conditions and post-conditions of the event depending on the direction of the connecting edge.

For example, for vertex t1 (see Fig. 1), places p1 and p2 represent the pre-conditions and place p3 means the post-condition. When pre-conditions are met, the action can be executed (i.e., the state of a system will change). Change of a system state should be depicted in Petri Net. For this purpose, tokens are used. A pass in a Petri Net is a small filled circle put in some vertex representing a place.

As a rule, to indicate the situation that condition is met, only one token is used. However, there are also cases when more than one token is used to indicate that the condition is met. Instead of putting the tokens into place, there is a possibility to write the number of tokens (usually small non-negative integers) inside a home. In the given case, the directed edge from place to the transition should have the weight which takes the values from the interval 1..k, where k is the number of tokens in the area.

If the weight value is not depicted in a Petri Net, it means that this weight equals "1".

Tokens are used not only to show that the system state has been changed but also to indicate what resources are necessary to enable the system state change. In the given case, a condition can be quantified by way of converting the required amount of resources into the number of tokens. It is assumed that one token corresponds to a certain amount of resources.

For example, shown in Fig. 1, place p4 has three tokens. It means that the amount of resources needed to satisfy the condition can be expressed as three tokens. When all pre-conditions are satisfied, the corresponding transition will be performed. In the terminology of Petri Nets, they use the term "fires". Return to the example shown in Fig. 1, transition t2 fires only when places p2 and p4 contain one and three tokens, respectively.

Some transitions in a Petri Net can require a certain amount of time to fire (so-called, timed transitions) and some can be executed immediately (so-called immediate transitions). In Fig. 1, transition t3 is an immediate transition. Timed

transitions in Petri Net are depicted as either filled or not-filled rectangles. Immediately, transitions are depicted in Petri Net as bold lines.

To indicate how many tokens each place has, m-vector (i.e., marking) was introduced. Marking has the form

$$M = (m_1, m_2, \ldots, m_n), \tag{1}$$

where n is the integer which is equal to the number of places in PN;

m_i (i=1,..,n) is equal to the number of tokens which place pi has.

After the system state has changed, values mi also change since the tokens indicate changes in system state. For the example presented in Fig. 1 initial marking is M0 =(1,1,0,3,0,0). After transition t1 has fired, the system state can be expressed by marking M1 =(0,0,1,3,0,0). Then transition t2 fires and system transfers to the state which can be expressed by marking M2 =(0,0,0,0,1,0). Next, after transition t3 has fired, the final marking M3 looks like (0,0,0,0,0,1). This way, after consideration of all possible sequences of firing of transitions, the reachability graph GR(M) can be obtained (see Fig. 2).

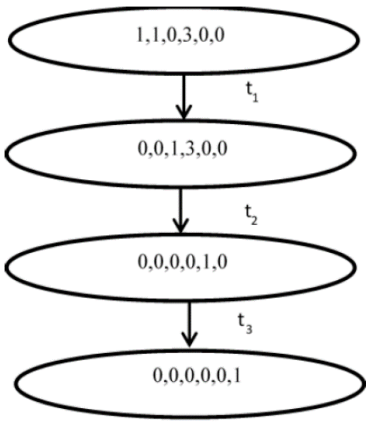


Fig. 2. Graph GR(M) for the Petri Net under consideration.

When a number of places in Petri Net is large, it is difficult to represent such PN graphically. In the given case, it is possible to express a Petri Net mathematically with the help of matrices (e.g., input matrix D- and output matrix D+) [17].

For the example under consideration these matrices look like as follows

$$D^- = \begin{array}{c|cccccc} & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \\ t_1 & 1 & 1 & 0 & 0 & 0 & 0 \\ t_2 & 0 & 0 & 1 & 1 & 0 & 0 \\ t_3 & 0 & 0 & 0 & 0 & 1 & 0 \end{array} \tag{2}$$

$$D^+ = \begin{array}{c|cccccc} & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \\ t_1 & 0 & 0 & 1 & 0 & 0 & 0 \\ t_2 & 0 & 0 & 0 & 0 & 1 & 0 \\ t_3 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \tag{3}$$

Petri Nets can be used for modelling a large range of complex systems. They allow the modelling of different system actions and properties such as concurrency, synchronization, sequences, conflicts, etc.[18].

Initially, Petri Nets were suggested without any notion of time. Consequent studies of Petri Nets resulted in the development of timed Petri Nets and stochastic Petri Nets [18]. In this paper, we use stochastic Petri Nets for modelling mutual testing in complex systems.

4. Mutual Testing

Mutual testing is considered in the context of system-level self-diagnosis of complex systems. System-level self-diagnosis was first described and analyzed in [19].

The term "self-diagnosis" implies that external diagnosis facilities are not needed for system diagnosis. System components perform diagnosis by themselves. It is assumed that system components are capable of providing not only assigned functions but also diagnosis tasks.

The term "system level" expresses that fault containing region is a system component. In the given case, they are going to determine if a system component is faulty or not.

The whole procedure of self-diagnosis includes three main parts, namely:

- execution of tests which system components perform on other system components (it was called a mutual testing);
- exchanging of test results among the system components according to the predefined method;
- execution of diagnosis algorithm by one of the system components.

Tests, which are used for system diagnosis, have binary result: either 0 when test indicates that tested system component (hereafter system unit or simply unit) has passed a test and 1 otherwise.

Thus, all test details are abstracted (i.e., they are not taken into account, and each particular test is considered as atomic).

There are four main issues which should be solved to perform system self-diagnosis of complex system.

The first issue is determining testing assignment. Testing assignment defines the possible set of tests among the system units. Testing assignment relies and depends on physical connections among the system units. Testing assignment can be represented via a directed testing graph (see Fig. 3).

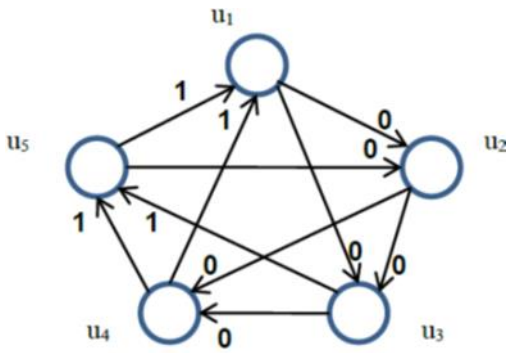


Fig. 3. Example of testing graph.

In Fig. 3, vertex u_i, i=1,..,5 denotes the system unit. Edges denote the tests among the system units. The test result is represented by a value either 0 or 1 and is shown as a label placed beside the corresponding edge.

The test result is a discrete random variable (which can take values either 0 or 1) since there is always the probability that testing unit evaluates a tested unit incorrectly [20].

For the case when system unit u_i tests unit u_j, probabilities of possible results for this test, r_{ij}, are shown in Table 1. These probabilities depend on the states of units u_i and u_j.

Table 1. Probabilities of test results.

r_{ij} (probability)		**u_j**	
		fault-free	faulty
u_i	fault-free	0(P_1)	1(P_2)
	faulty	0(P_3)	1(P_4)

Probabilities P_k, k=1,..,4, shown in Table 1. can be described as follows:

P_1 - probability that fault-free unit u_i correctly evaluates the state of fault-free unit u_j;

P_2 - probability that fault-free unit u_i correctly evaluates the state of faulty unit u_j;

P_3 - probability that faulty unit u_i correctly evaluates the state of fault-free unit u_j;

P_4 - probability that faulty unit u_i correctly evaluates the state of faulty unit u_j;

In [19], some assumptions have been made about these probabilities. Particularly, P_1=1, P_2=1, P_3=0.5, P_4=0.5. Such assumptions can be interpreted as a symmetric invalidation model. The asymmetric invalidation model implies that P_4 =1. This is the only one difference from the symmetric invalidation model. This model was introduced in [20].

The second issue is the determining of assumptions that underlie a diagnosis algorithm. These assumptions are related to possible faults and to test results. They directly impact the quality of system diagnosis (i.e., correctness and completeness).

Different faulty assumptions can be taken into consideration, such as permanent and intermittent faults, and also hybrid faulty situations. Several studies of the problem of how to analyze and sort out the obtained results have been conducted [22, 23].

For example, Preparata at. al. [19] considered only permanent faults of system units. In many cases, system units can have not only permanent but also intermittent faults [24,25,26]. There are also hybrid faulty situations when some system units are permanently faulty whereas some other system units are intermittently faulty.

The third issue is the determining organization of test execution which includes test scheduling, test repetition and test performing (either randomly or deterministically).

The fourth issue relates to the problem of determining the unit which will be assigned the task of executing a diagnosis algorithm. The result of the diagnosis algorithm can be sent to the system environment.

As concerns the problem of organization of test execution, they differentiate between the tests performed for the purpose of system self-checking and tests performed for the purpose of system self-diagnosis.

Self-checking is the process which aims at discriminating between two system states (i.e., faulty and fault-free). The result of self-checking doesn't indicate which of the system unit(s) has failed. It only testifies to the presence of fault(s) in the system.

For providing system self-checking, it is not necessary to form a syndrome (i.e., set of test results) and, consequently, not necessary to perform any processing of obtained results. Only a message or signal informing the system environment about the system's fault-free state is sufficient. This can be done by any system unit.

It is assumed that tests are executed during system operation. Thus, it is not possible to determine in advance which of the system units will be idle at the particular moment, and, thus, will be able to test (or be tested by) another system unit.

Consequently, it is not known if all of the system units have been tested for elapsed time. In view of this, the task arises to determine the duration of the self-checking procedure which ensures that all system units will be tested (i.e., will be checked) with high probability.

The goal of the self-diagnosis procedure is to identify the faulty unit(s). It is worth noting that the self-diagnosis procedure may use the organisation of test execution differently from the organisation used for system self-checking.

System-level diagnosis is an abstraction of high level and, thus, its practical implementation to particular instances of complex systems is the task that requires additional investigation, both theoretical and practical.

Each class of complex systems (such as multiprocessor systems, multi-agent systems, web service compositions, teams of robots, etc.) has its own dependability requirements expressed in the specification and its own functional restrictions.

One of the challenging tasks is how to improve the model of system self-diagnosis so that to include more specific parameters of concrete classes of complex systems.

5. Presenting Tests Among System Units with the help of Petri Net

To insight into this problem, we start with a simple example. Let system consists of two units u_1 and u_2 which are capable of testing each other.

We assume that tests can be performed only when system units are free from performing their assigned tasks (i.e., their proper functions described in the system specification). That is why we introduce two states for each system unit. Particularly, idle and engaged states. Test between units can be executed only when both units are in idle state. In Petri Net (PN), it can be shown as follows (see Fig. 4).

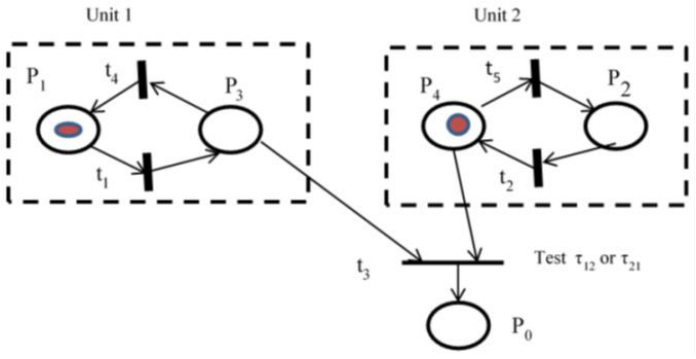


Fig. 4. PN that models the test between system units u_1 and u_2.

Places P_3 and P_4 correspond to idle states of units u_1 and u_2 respectively. Whereas, places P_1 and P_2 correspond to engaged states of these units. In this simple example, we do not discriminate which unit acts as a testing unit. Thus, in PN, only one transition t_3 (which corresponds to either test τ_{12} or τ_{21}) is used. The event indicating that the test was executed is modelled in PN by place P_0. In order to model the initial states of system units, PN uses the tokens. Putting a token in a place, for example, in place P_1, means that unit u_1 is in an engaged state. When place P_0 has a token, it means that a test between system units was executed. In Fig. 4, timed transitions t_4 and t_5 model the transitions of units u_1 and u_2 from idle to engaged states. Whereas timed transitions t_1 and t_2 model the transition of units u_1 and u_2 from engaged to idle states. Immediate transition t_3 models the execution of test either τ_{12} or τ_{21}. In cases when the order of

test executions is important, it is possible to discriminate between tests τ_{12} and τ_{21}. For this case, the PN looks like it is shown in Fig. 5.

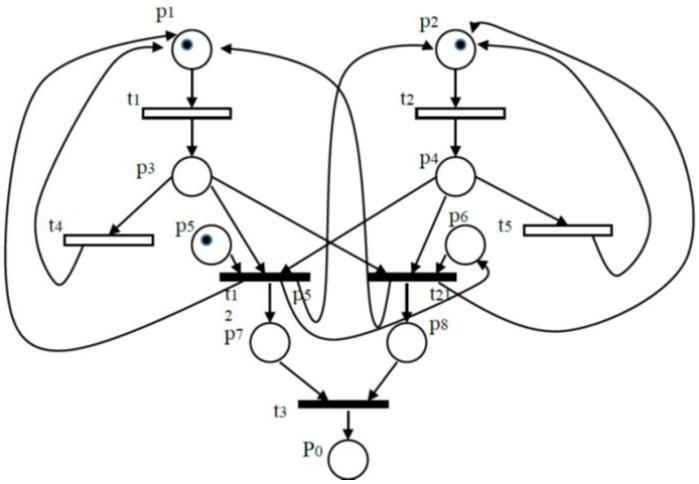


Fig. 5. PN that models test τ_{12} and τ_{21} between units u_1 and u_2.

This PN allows modelling the order of tests execution and their repetition (i.e., how many times the test can be executed). Places P_5 and P_6 are used to model the schedule of test execution (test τ_{12} before test τ_{21}) and test repetition. If for example, it is needed to repeat test τ_{12} three times, it can be modelled by putting three tokens into place P_5. The fact that place P_5 has a token and place P_6 doesn't have a token indicates that test τ_{12} will be executed before test τ_{21}.

PN which models the mutual testing in the system with more than two units is more complex and contains a great number of places and transitions [14].

Petri Nets formed according to the described procedure can be used for the simulation of mutual testing in complex systems consisting of units capable of performing tests on other system units.

6. Simulation of Mutual Testing by using SHARPE

Since it is assumed that each system unit can transfer from idle state to engaged state and vice versa at arbitrary time, the time of transition (e.g., transitions t_1, t_2, t_4 and t_5 in Fig. 4) is a random value that can follow arbitrary distribution. In case the time of transaction firing follows an exponential distribution, it is possible to use the tool SHARPE [16]. The advances of using this tool are described in [14]. By using SHARPE, it is possible to simulate the test execution in the system. The main goal of such a simulation is to reach the system state when all tests needed for diagnosis are executed. Usually, the event that all needed tests are executed can be modelled in PN by the token in a special place (usually denoted as P_0).

PN allow simulating the test execution for different system parameters, such as

- different values of the rate of system transition from one state to another;
- different duration of each particular test;
- different initial system states.

In either case, the goal of the simulation is to determine the time when special place P_0 has a token and to evaluate how different system parameters can influence this time.

SHARPE allows simulating mutual testing in the system for different system parameters mentioned above. It is also important to evaluate how the total number of system units can impact the time of testing. This task was also solved by using SHARPE. In Fig. 6, the result of the simulation is shown for the case when the rate of transition from idle state to engaged state is equal to 5. The rate of transition from engaged to idle state is equal to 1, and the rate which corresponds to single test execution is equal to 100.

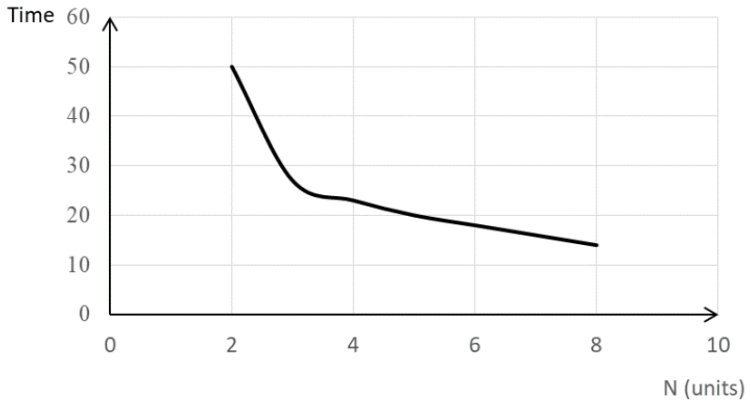


Fig. 6. Influence of the number of system units on the testing time.

It is worth noting that SHARPE doesn't allow simulating mutual testing when the firing time of timed transitions doesn't follow the exponential distribution. In this case, it is needed other methods which would be capable of providing a simulation of test execution under given conditions.

7. Simulation of Mutual Testing by Using Software Programs

The essence of such an approach consists of the following (see Fig. 7).

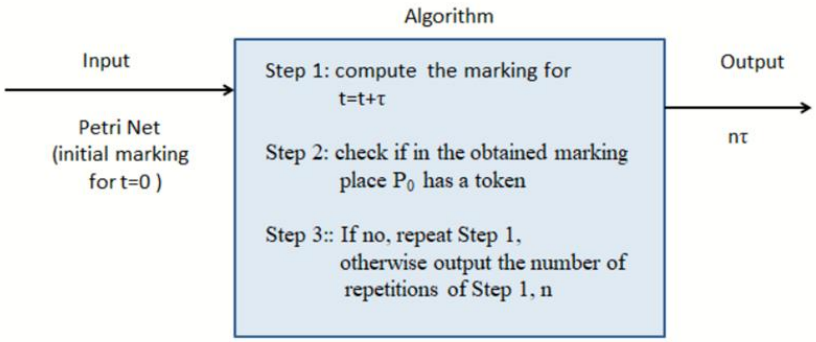


Fig. 7. Block diagram of the algorithm.

As an input, the PN with initial marking is used. The transition of a system from one state to another is modelled in PN as changing of marking. Therefore, after a certain period of time τ (usually called a clock) the new system state or new marking of PN is determined. The event indicating that all needed tests are executed (or the marking of PN with a token in the targeted place P_0 is reached) is of special interest. As soon as place P_0 has a token, the algorithm is terminated. Output value n_τ gives the information on how long the testing procedure lasts (i.e., information about the duration of mutual testing).

The main part of this algorithm is Step 1: marking determining. Determining the new marking requires considering several issues such as, for example, time of firing of transitions. It can be explained by using a simple example with two places. Particularly, with place P_1 (which has a token) and place P_2 (without a token). After firing of timed transition t_1, a token will be transferred from place P_1 to place P_2.

Timed transition t_1 doesn't fire immediately. It takes some time to fire. This time is denoted as t_A. As a rule, clock τ is lesser than the mean value of t_A. Time t_A is a random value that can have exponential or any other distribution.

In the case of the exponential distribution, for each value of t_A the probability P_f, where $P_f = 1-e^{-\lambda t_A}$, can be calculated. Knowing P_f, it is possible to perform simulation by using the function "random". Function *rand()* returns the result which allows us to conclude if transition t_1 has fired or not.

Similarly, all timed transitions in the PN that model the mutual testing in complex systems can be examined. In order to perform such a simulation of mutual testing, it is important to choose the correct value of clock τ. For the case when random value t_A has an exponential distribution with mean value $t_{mean}=1$, the average time of transition firing for different values of clock (see Table 2) was determined.

Table 2. Impact of the clock on the simulation of transition firing.

τ	Average time of transition firing
1	2
0.5	1
0.25	0.5
0.17	0.5
0.1	0.5

As it follows from Table 2, the value of the clock should be less than or equal to 0.25. In the given case, the value of the clock doesn't have an impact on the average time of transition firing. Generally, the correct clock τ can be determined from the condition $\tau \leq t_{mean}/4$ (when firing time follows exponential distribution). For other distributions, clock τ should be set from the condition that it doesn't impact the average time of transition firing.

The next problem that should be considered is the assessment of the impact of initial marking on the time of mutual testing. For example, presented in Fig. 4, the following results (see Table. 3 and Table. 4) were obtained.

Table 3. Results of simulation for the case when P[0]=0, P[1]=0, P[2]=1, P[3]=1 and P[4]=0. Initial marking M_0=(0,0,1,1,0).

$k=t_{mean}/\tau$	**t_{mean}**	**Average time of mutual testing**
4	1	0.5
4	10	2.5
4	20	5

Table 4. Results of simulation for the case when P[0]=0, P[1]=1, P[2]=1, P[3]=0 and P[4]=0. Initial marking M0 =(0,1,1,0,0)

k	tmean	Average time of mutual testing when timed transitions meet the condition: t1,t2>>t4,t5 (mean values)
4	1	2
4	10	15
4	20	35

In these Tables, 3 and 4, the denotation P[i]=1, i=1..m, means that place P_i has a token, if P[i]=0 then place P_i doesn't have a token. When t_1 approximately equals t_5, the average time of mutual testing can be very long.

In the example shown in Fig. 4, both tests τ_{12} and τ_{21} will be performed only when both units are in an idle state (i.e. when tokens are in places P_3 and P_4). It may require a long time. However, when a unit (e.g., unit 1) is in idle state it can send a test to unit 2. If unit 2 is also in an idle state, the test τ_{12} is performed without delay (i.e., immediately). Otherwise, when unit u_2 is in engaged state, test τ_{12} is performed only when unit u_2 transfers from engaged to the idle state and is able to send the reply to unit u_1. Test which unit u_1 sent to unit u_2 can be saved by unit u_2 and then it can be executed when unit u_2 will be idle. Thus, in order to execute test τ_{12} the condition that both units should be in idle state may be relaxed. It means that less time will be needed for performing test τ_{12}.

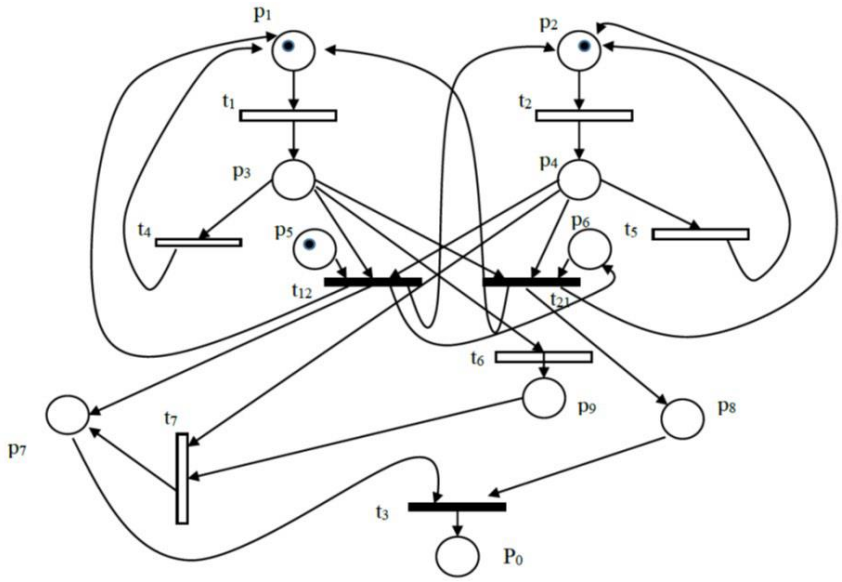


Fig. 8. Modified Petri Net

For this purpose, some changes in the considered Petri Net should be made. Particularly, place P_9 and transitions t_6 and t_7 should be added to the PN shown in Fig. 4. Place P_9 will have a token when place P_3 has a token (i.e., unit u_1 is in idle state). When place P_4 will have a token (i.e., unit u_2 is in an idle state) transition t_7 fires (i.e., test τ_{12} is executed). Transition t_6 is used to put a token in place P_9 when place P_3 has a token (thus, to record the event that the test was sent). For simplicity reasons, it is assumed that test τ_{21} can be executed only when both units are in an idle state. Modified PN looks as in the Fig.8.

For real complex systems it means that in each system unit, an additional process or thread should be introduced. This process will be responsible for storing the data received from other system units (e.g., data related to the test reply from the tested unit). Now, a unit after transitioning from engaged to idle state can immediately process these data.

8. Conclusion

In this paper, the mutual testing in complex systems is considered. This means that external diagnosis facilities are not needed for system diagnosis. System units (components) perform diagnosis by themselves. It is assumed that system units can serve not only their assigned functions but also the diagnosis tasks, i.e., they can execute tests on other system units and a diagnosis algorithm.

The efficiency of mutual testing for system diagnosis can be assessed by modelling and following the simulation of joint testing in complex systems.

In the paper, the models based on Stochastic Petri Nets (SPN) were considered and used for the simulation of test execution. SPN was chosen for this purpose because models using SPN have some advances over other models. Models that use SPN can consider more system states and parameters, and their modification requires minor efforts.

Having developed the model of mutual testing, it is possible to conduct the simulation of test execution in the system. In the paper, the simulation using the tool SHARPE and the simulation using software programs were considered and compared. Each method of simulation has its advantages and disadvantages. The SHARPE simulation can be applied when the total number of system units is small and the time of system state change follows an exponential distribution.

The simulation that uses software programs can be applied when the time of system state change may follow arbitrary distribution. The bottleneck of such simulation consists of the difficulty of choosing the correct clock (i.e., the period after the expiration of which the new system state is determined). In the paper, the appropriate clock was selected and evaluated. Using this simulation method, the impact of initial marking (i.e., the system's initial state) on the total time of mutual testing was assessed. Simulation results have shown that situations when the entire time of joint testing may be very long, are possible. Given this, we recommend improving the implementation of mutual testing in natural complex systems by adding specific software (described above in Section 6) to each system unit. As a deficiency of the simulation that uses software programs, we can name the difficulty of making changes in the program after some system or testing parameters have been changed.

Simulation results can be used to assess the efficiency of system diagnosis using mutual testing (e.g., completeness and correctness of diagnosis, total time of diagnosis, etc.).

We can conclude that the novelty of the research presented in this paper consists of the following:

- the existing models that use Petri Nets were modified so that now they can take into account more system and testing parameters;
- simulations that use the above-considered models were conducted for different systems and testing parameters. Based on simulation results, the recommendation for implementation of mutual testing in complex systems was made;
- recommendation for deciding on the appropriate simulation method was made based on the results of their comparison.

References

[1] Ribbens W. B. Electronic control system diagnosis. In Understanding Automative Electronics, 2017, doi: 10.1016/c2016-0-00011-6.

[2] Rentschler M., Kehrer S., Zangl C. P. System self-diagnosis for industrial devices. IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA), pp. 1-8, 2013.

[3] Zhu M., Li J., Wang W., Chen D. Self-detection and self-diagnosis methods for sensors in intelligent integrated sensing system. In IEEE Sensors Journal, 2021, doi: 10.1109/JSEN.2021.3090990.

[4] Morris A. S., Langari R. Measurement and Instrumentation Theory and Application, Elsevier Science, 2015, 726 pages.

[5] Preas B. Smart matter systems, an introduction through examples. Proc. Of the International Symposium on physical design, ISPD'14, pp. 131-132, 2014.

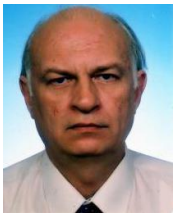
[6] Mashkov V.A., Barabash O.V. Self-testing of multimodule systems based on optimal check-connection structures. Engineering Simulation, 13(3), 479-492 (1996).

[7] Mashkov V.A., Mashkov O.A. Interpretation of diagnosis problem of system level self-diagnosis. Mathemetical Modeling and Computing, 2(1), pp. 71-76 (2015).

[8] Preparata T., Metze G., Chien R. On the connection assignment problem of diagnosable system. IEEE Transactions on Electronic Computers. EC-16, n.12, 848–854 (1967).

[9] M. Barborak M., Malek M., Duhbura A. The Consensus Problem in Fault Tolerant Computing. ACM Computing Surveys, vol. 25, No. 2, (1993).
[10] Laforge L., Korver K.F. Mutual test and diagnosis: architectures and algorithms for spacecraft avionics. Proceedings of IEEE Aerospace Conference. Vol. 5, 295-306 (2000).
[11] Petri A. Kommunikation mit Automaten. Bonn: Institute fur Instrumentelle Mathematik. Schriffen des IIM Nr. 3, 19622. Also, English translation „Communication with Automata“, New York: Griffiss Air Forth Base, Tech., Rep. RADC-TR-65-377, Vol. 1, Suppl. 1, 1966.
[12] Peterson J. L. Petri Net Theory and the Modeling of Systems. Prentice Hall, New York, 1981, 290 pages.
[13] Billington J., Diaz M., Rozenberg C. Application of Petri nets to communication networks. Advances in Petri Nets, LNCS, 1, 1999, 314 pages.
[14] Mashkov V., Barilla J., Simr P. Applying Petri Nets to modeling of many-core processor self-testing when tests are performed randomly. Journal of Electronic Testing, Vol.29, No.1, 25-34 (2013).
[15] Bobbio A. System modeling with Petri Nets. In: A.G. Colombo and A. Saiz de Bustamante (eds.). System Reliability Assessment, Kluwer p.c., 102-143 (1990).
[16] Sahner R.A., Trivedi K.S. Reliability modeling using SHARPE. IEEE Trans. Reliab. R-36(2), 186-193 (1987).
[17] Reddy, V. N., Mavrovouniotis, M. L.: Petri net representation in metabolic pathways. Proc. Int. Conf. Intell. Syst. Mol. Biol. vol. 1, pp. 328-336. (1993)
[18] Mashkov V., Barilla J., Simr P., Bicanek J. Applying petri nets to coalition formation modeling. Advances in Intelligent Systems and Computing, 442, pp. 83-97 (2016).
[19] T. Preparata, G. Metze, R. Chien. On the connection assignment problem of diagnosable system. IEEE Transaction on Electronic Computers. Vol. EC-16, No. 12, pp. 848-854, 1967.
[20] M. L. Blount. Probabilistic treatment of diagnosis in digital systems. In 7th IEEE Int. Symp. On Fault-Tolerant Computing, pp. 72-77, 1977.
[21] T. Barsi, T. Grandoni, P. Maestrini. A theory of diagnosis of multiprocessor systems. In Proc. of the 7th Annual Symposium on Computer Architecture, pp. 31-36, 1980.
[22] M. Barborak, M. Malek, A.T. Dahbura. The consensus problem in fault-tolerant computing. ACM Computing Surveys. Vol. 25, No. 2, pp. 171-220, 1993.
[23] A.K. Somani. System level diagnosis and implications in current context. Chapter on dependable computing systems: paradigms, performance issues, and applications. Edited by Hassan B. Diab and Albert Y. Zomaya. Wiley Series on Paralled and Distributed Computing, 2005 (37 pages).
[24] S. Kamal, C.V. Page. Intermittent fault: a model and a detection procedure. IEEE Trans. Comput., Vol. C-23, No. 7, pp. 713-719, 1974.
[25] S. Mallela, G. Masson. Diagnosable systems for intermittent faults. IEEE Trans. Comput., Vol. C-27, No. 6, pp. 550-566, 1978.
[26] V. Mashkov, J. Fiser, V. Lytvynenko, M. Voronenko. Diagnosis of intermittently faulty units at system level. DATA, 2019, 4(1), 44.

Authors' Profiles



Viktor Mashkov, Doctor of Science in Engineering Docent Department of IT at the University of J.E. Purkyne in Usti nad Labem (Czech Republic). His major research focuses on the dependability of computer systems, software fault tolerance, system level self-diagnosis and multi-agent systems.

Volodymyr Lytvynenko, Professor, Head of the Department of Informatics and Computer Science, Kherson National Technical University, Kherson, Ukraine. Area of scientific interests: Area of scientific interests: inductive modelling of complex systems, computer intelligence systems, development of hybrid algorithms, Bayesian networks, system level self-diagnosis.

Irina Lurie, Junior academic of the Department Industrial Engineering and Management,Ben-Gurion University of Negev, David Ben Gurion Blvd 1, Beer Sheva, Izrael. Area of scientific interests: Inductive modelling of complex systems, computer intelligence systems, development of hybrid algorithms.

How to cite this paper: Viktor Mashkov, Volodymyr Lytvynenko, Irina Lurie, "Modeling and Simulating Mutual Testing in Complex Systems by Using Petri Nets", International Journal of Image, Graphics and Signal Processing(IJIGSP), Vol.15, No.6, pp. 81-93, 2023. DOI:10.5815/ijigsp.2023.06.07