Progress in Flight Dynamics, GNC, and Avionics **6** (2013) 487-500
DOI: 10.1051/eucass/201306487

SIMULATION MODELING BASED METHOD FOR CHOOSING AN EFFECTIVE SET OF FAULT TOLERANCE MECHANISMS FOR REAL-TIME AVIONICS SYSTEMS

A. G. Bakhmurov, V. V. Balashov, A. B. Glonina, V. N. Pashkov, R. L. Smeliansky, and D. Yu. Volkanov

Laboratory of Computer Systems
Department of Computational Mathematics and Cybernetics
M. V. Lomonosov Moscow State University
1-52 Leninskiye Gory, Moscow 119991, Russia

In this paper, the reliability allocation problem (RAP) for real-time avionics systems (RTAS) is considered. The proposed method for solving this problem consists of two steps: (*i*) creation of an RTAS simulation model at the necessary level of abstraction and (*ii*) application of meta-heuristic algorithm to find an optimal solution (i. e., to choose an optimal set of fault tolerance techniques). When during the algorithm execution it is necessary to measure the execution time of some software components, the simulation modeling is applied. The procedure of simulation modeling also consists of the following steps: automatic construction of simulation model of the RTAS configuration and running this model in a simulation environment to measure the required time. This method was implemented as an experimental software tool. The tool works in cooperation with DYANA simulation environment. The results of experiments with the implemented method are presented. Finally, future plans for development of the presented method and tool are briefly described.

1 INTRODUCTION

In this paper, RTAS are considered. Modern RTAS consist of multiple devices connected by a network of data transfer channels. Each real-time avionics device runs a specific mix of applications and of system software. Some parts of this software have deadline constraints. Real-time avionics systems are characterized by four fundamental properties:

(1) functionality;

(2) performance;

(3) dependability; and

(4) cost.

One of the RTAS development goals is reaching a high level of dependability while providing required functionality and meeting constraints on performance and cost. Different methods are used to guarantee the necessary level of dependability. All these methods involve structural, time, informational, or space redundancy. It should be noted that for aircraft and space systems, there are strict constraints on total weight, space, and power consumption. In this paper, a method for reaching the goal mentioned above is described.

Dependability of RTAS is its ability to deliver service that can be justifiably trusted [1]. For estimation of dependability, the notion of reliability is used. Reliability of an RTAS is defined as the conditional probability that the system is operational during the interval $(0, t_{\text{end}})$ provided that it was operational at time $t = 0$ [2]. Different approaches to provide dependability have been developed. One of them is fault tolerance (FT). Fault tolerance is an approach that enables an RTAS to continue operating correctly in case of failure of some of its components. In the following section, some FT mechanisms are considered that are used in RTAS.

2 FAULT TOLERANCE MECHANISMS

Fault tolerance mechanisms are designed to allow a system to tolerate faults that remain in the system after its development. Fault tolerance mechanisms are employed during development of the system. When a fault occurs, these mechanisms provide means for the system to prevent system failure from occuring.

In this paper, the following mechanisms are considered: N-version programming (NVP/0/1 and NVP/1/1) and recovery blocks (RB/1/1) [3, 4].

N-version programming [3] involves a deciding module called the voter and N independently developed software versions, which are functionally equivalent. N is usually an odd number. In NVP, all N software versions for the same task are executed at the same time (i. e., concurrently), and their outputs are collected and evaluated by the voter. The majority of the outputs determines the voter's decision. Any hardware failure can cause the associated software running on it to produce unacceptable results. Two subclasses of NVP mechanism are considered:

(1) the NVP/0/1 mechanism [4] has zero hardware faults tolerated and a single software fault tolerated. The system architecture consists of N independent software versions (components) running concurrently on a single hardware component; and

(2) the NVP/1/1 mechanism [4] involves N independent software versions, each running on a separate hardware component. The system is operational if more than half software versions (on operational hardware) are operational.

The RB/1/1 mechanism [3, 4] involves an adjudication module called acceptance test and at least two software components called alternates. When execution of the first (or primary) alternate is finished, output of this alternate is tested for acceptance. If it fails, the process rolls back to the beginning and then executes the second alternate and tests its output for acceptance. This process continues until the output from some alternate is accepted or all outputs of the alternates have been tested and have failed.

In the following section, some terms are formulated and the RAP for RTAS is introduced.

3 RELIABILITY ALLOCATION PROBLEM

Formal model of an RTAS used in this paper is based on program behavior invariant [5]. As it has been already noted in Introduction, RTAS are characterized by four fundamental properties: (i) functionality; (ii) performance; (iii) dependability, and ($i\nu$) cost. Functionality and performance are described by program behavior invariant; dependability and cost are additionally introduced in this model.

Real-time avionics system consists of subsystems and links between subsystems. Structure of the RTAS can be represented as a graph. Subsystems correspond to the nodes of this graph and links between subsystems correspond to its edges. Program behavior invariant is a tuple $\langle \mathrm{Cr}, \mathrm{LE}, \mathrm{Bh}, G \rangle$ where Cr is the initial structure of the system (represented as a graph); LE is the behavior of operating system; Bh is the behavior of application software; and G is the interpretation of LE and Bh on Cr. An algebra of LE, Bh, and G is defined in [5]. In brief, LE and Bh describe functionality of the RTAS and G describes performance of the RTAS.

Each subsystem consists of a hardware component, a software component, and an optional FT mechanism. For each hardware and software component, there is a set of versions of this component. Number of component versions used in the subsystem and number of copies for one version are defined by the FT mechanism chosen for the subsystem. Dependability of real-time avionics is determined by choice of hardware components, software components, and FT

mechanisms for nodes of Cr. Let introduce notations for description of Cr, dependability, and cost:

C (System) — cost of the RTAS configuration;

C_{ij}^{hw} and C_{ij}^{sw} — cost of H_{ij} and S_{ij};

$C_{\max}$ — maximum system's cost;

D_i — maximum allowed execution end time for the software component of subsystem U_i. Execution time is measured from the RTAS execution start;

F_i — FT mechanism chosen for the subsystem U_i; $F_i \subseteq \mathrm{FT}_i$;

$\mathrm{FT}_{\mathrm{avail}}$ — set of available FT mechanisms for the RTAS. Here, $\mathrm{FT}_{\mathrm{avail}} = \{\mathrm{None}, \mathrm{NVP}/0/1, \mathrm{NVP}/1/1, \mathrm{RB}/1/1\}$ is considered. "None" corresponds to the absence of FT mechanism;

FT_i — set of available FT mechanisms for the subsystem U_i; $\mathrm{FT}_i \subseteq \mathrm{FT}_{\mathrm{avail}}$;

H_i — hardware component of subsystem U_i;

$H_i^{F_i}$ — multiset of versions H_{ij} chosen for the hardware component H_i of the subsystem U_i;

H_{ij} — the jth version of the hardware component H_i, $\forall\, i \in [1; n]$, $\forall\, j \in [1; p_i]$;

i — index of subsystem, $i \in [1; n]$;

n — number of subsystems within the distributed RTAS;

p_i — number of hardware component versions available for subsystem U_i;

P_{all}^i — probability of failure from related fault among all software versions for the subsystem U_i;

P_d^i — probability of failure of decider or voter for the subsystem U_i;

P_{rv}^i — probability of failure from related fault between two software versions for the subsystem U_i;

q_i — number of software component versions available for subsystem U_i;

R (System) — reliability of the RTAS configuration;

R_{ij}^{hw} and R_{ij}^{sw} — reliability of H_{ij} and S_{ij};

S_i — software component of the subsystem U_i;

$S_i^{F_i}$ — multiset of versions S_{ij} chosen for the software component S_i of the subsystem U_i;

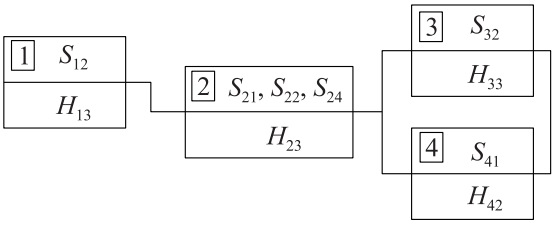

Figure 1 Example of RTAS with 4 subsystems

S_{ij} — the jth version of the software component for subsystem U_i, $\forall i \in [1; n]$, $\forall\ j \in [1; q_i]$;

System — a particular configuration of RTAS. It includes a set of $H_i^{F_i}$, $S_i^{F_i}$, and F_i for each subsystem U_i;

Systems — set of all configurations of RTAS. It includes all $H_i^{F_i}$, $S_i^{F_i}$, and F_i combinations in all subsystems U_i, System $\in$ Systems;

T_i (System) — execution end time of the software component of subsystem U_i. It depends on initial data for the subsystem, choice of hardware version, software version, and FT mechanism; T_i is measured via simulation;

U_i — subsystem i;

x_{ij} — number of copies of H_{ij} for H_i;

y_{ij} — number of copies of S_{ij} for S_i.

According to this notation, Cr $= [\{U_1, \ldots, U_i, \ldots, U_n\}, \ldots, (U_i, U_j), \ldots]$ with $U_i = \{H_{i1}, S_{i1}, \text{None}\}$ and System $= [\{U_1, \ldots, U_i, \ldots, U_n\}, \ldots, (U_i, U_j), \ldots]$ with $U_i = \{H_i^{F_i}, S_i^{F_i}, F_i\}$, $i \in [1; n]$.

An example of RTAS with 4 subsystems is shown in Fig. 1. Here, second subsystem has FT mechanism NVP/0/1 with three different software versions running in parallel on one hardware component.

The total cost of the system is:

$$C\,(\text{System}) = \sum_{i=1}^{n} C_i = \sum_{i=1}^{n} \left(\sum_{j=1}^{p_i} x_{ij} C_{ij}^{hw} + \sum_{j=1}^{q_i} y_{ij} C_{ij}^{sw} \right) = \sum_{i=1}^{n} \sum_{j=1}^{p_i} x_{ij} C_{ij}^{hw} + \sum_{i=1}^{n} \sum_{j=1}^{q_i} y_{ij} C_{ij}^{sw}\,.$$

Let assume that the kth hardware component and the lth software component for subsystem U_i are choosen. The reliability of subsystem without FT mechanisms is:

$$R_i = R_{ik}^{hw} R_{il}^{sw}\,, \qquad k \in [1, p_i]\,,\ \ l \in [1, q_i]\,.$$

For FT mechanisms NVP/0/1, NVP/1/1, and RB/1/1, estimation of reliability is given in [4]. For example, the reliability of U_i in case of FT mechanism RB/1/1 is considered. The k_1th and k_2th hardware components and l_1th and l_2th software components are assumed to be chosen. In this case, the reliability is:

$$\begin{aligned} R_i = 1 - \big(&P_{rv}^i + (1 - P_{rv}^i)P_d^i + (1 - P_{rv}^i)(1 - P_d^i)P_{\text{all}}^i \\ &+ (1 - P_{rv}^i)(1 - P_d^i)(1 - P_{\text{all}}^i)(1 - R_{ik_1}^{hw})(1 - R_{ik_2}^{hw}) \\ &+ (1 - P_{rv}^i)(1 - P_d^i)(1 - P_{\text{all}}^i)(1 - (1 - R_{ik_1}^{hw})(1 - R_{ik_2}^{hw}))(1 - R_{il_1}^{sw})(1 - R_{il_2}^{sw})\big)\,. \end{aligned}$$

Total reliability of the system is:

$$R\,(\text{System}) = \prod_{i=1}^{n} R_i\,.$$

Now, the RAP for RTAS can be formalized:

Given:

n;

p_i, q_i, $i \in [1, n]$;

$\langle \text{Cr}, \text{LE}, \text{Bh}, G \rangle$;

$C_{i,j}^{hw}$, $R_{i,j}^{hw}$, $i \in [1, n]$, $j \in 1, p_i]$;

$C_{i,j}^{sw}$, $R_{i,j}^{sw}$, $i \in [1, n]$, $j \in [1, q_i]$;

FT_i, $i \in [1, n]$;

P_{rv}^i, P_d^i, P_{all}^i, $i \in [1, n]$;

D_i, $i \in [1, n]$; and

$C_{\max}$.

Find:

Configuration System $\in$ Systems, which maximizes the system reliability $R\,(\text{System})$ under constraints $C\,(\text{System}) \le C_{\max}$ and $\forall\ i \in [1; n]:\ T_i\,(\text{System}) < D_i$.

Reliability allocation problem can be formulated as a discrete optimization task:

$$\begin{cases} \max\limits_{\text{System}\in\text{Systems}} R\,(\text{System})\ ; \\ C\,(\text{System}) \le C_{\max}\,; \\ T_i\,(\text{System}) < D_i\,,\ \ \forall i \in [1,n]\,. \end{cases}$$

4 RELATED WORK

The scientists worked on the RAP since 1960s. Since RAP is an nondeterministic polynomial-time hard (NP-hard) problem [6], its solution possesses a great practical and theoretical value. Originally, this problem had been solved by exact methods, but as the number of components grew, the scientists started to use heuristic methods. In 1975, Holland proposed the first metaheuristic scheme — genetic algorithms [7]. During the 1980s and 1990s, other metaheuristic schemes were suggested, such as simulated annealing, ant colony algorithms, and immune algorithms. All these approaches are actively used in our days to solve RAP. Current state of the art is described in detail in [8, 9]. Surveys [10–12] should also be noted.

The following metaheuristic algorithms and other methods are applied to RAP for the last years: ant colony algorithms, genetic algorithms, hybrid genetic algorithms, tabu search, simulated annealing algorithms, immune algorithms, cellular evolutionary approach, heuristic methods, and exact methods including dynamic programming.

Unfortunately, in most of the abovenoted papers, only redundancy is considered as an example of FT mechanism. In these papers, the reliability of either only hardware or only software is considered. It should also be noted that exact methods, including dynamic programming, are effective for small dimension tasks only [13]. As it was shown in [5], separate consideration of RTAS hardware and software components does not allow adequate describing of system characteristics. To evaluate the reliability of the whole system, both hardware and software reliability must be considered together as, for example, described in [4,13]. The authors of [4] and [13] describe the use of FT mechanisms such as NVP/0/1, NVP/1/1, and RB/1/1. However, deadline constraints for software tasks are not considered in [4,13]. In this paper, checking of deadline constraints for software tasks is performed by simulation [5].

Genetic algorithm is a population-based directed random search technique inspired by the principles of evolution. Though it provides only appximate solutions, it can be effectively applied to almost all complex combinatorial problems [9]. To improve the computational efficiency, or to avoid premature convergence, numerous researchers have been inspired to seek effective combinations of genetic algorithms with heuristic algorithms, simulated annealing methods,

neural network mechanisms, steepest descent methods, or other local search methods. These combinations are generally called hybrid genetic algorithms and they represent one of the most promising development directions in optimization techniques.

In this paper, an adaptive hybrid genetic algorithm (AHGA) for solving RAP is presented. This algorithm includes application of an adaptive control procedure to automatically regulate the genetic algorithm's parameters.

5 METHOD FOR SOLVING THE RELIABILITY ALLOCATION PROBLEM

In this section, the proposed method for solving the RAP for RTAS is described.

The method consists of the following steps:

1. Creation of an RTAS simulation model with necessary level of detail.
2. Searching for an optimal solution (i. e., optimal set of fault tolerance techniques) with AHGA.

In AHGA, the solutions of the RAP are coded by chromosomes. Chromosome contains integer string coded information for each subsystem. For a subsystem, this string includes:

- the number of hardware component versions set ($H_i^{F_i}$);
- the number of software component versions set ($S_i^{F_i}$); and
- the type of FT mechanism (F_i).

For each solution in the population, the goal function defines the quality of this solution. For RAP, the goal function is the reliability of the whole system. This function is being maximized.

Because most of the solutions from the search space may not satisfy cost and time constraints, a penalty function is added into the algorithm. Penalty function is the factor by which the goal function value should be multiplied. The following penalty function for time and cost constraints was proposed:

$$R^{**}\,(\text{System}) = R^{*}\,(\text{System})\,E_{\text{cost}}\,(\text{System})$$

where

$$R^{*}\,(\text{System}) = \prod_{i=1}^{n} R_i^{*}(\text{System})\,;$$

$$R_i^{*}\,(\text{System}) = R_i E_{\text{time}}^{i}\,(\text{System})\;;$$

$$E^i_{\text{time}}\,(\text{System}) = \begin{cases} 1 & \text{if } T_i\,(\text{System}) < D_i\,; \\ \dfrac{D_i}{T_i\,(\text{System})} & \text{otherwise}\,; \end{cases}$$

$$E_{\text{cost}}\,(\text{System}) = \begin{cases} 1 & \text{if } C\,(\text{System}) < C_{\max}\,; \\ \dfrac{C_{\max}}{C\,(\text{System})} & \text{otherwise}\,. \end{cases}$$

If the solution meets the cost (time) constraint, then the penalty function is equal to 1; else, it is inversely proportional to the cost (time).

The AHGA execution can be divided in the following steps:

I Generation of initial population of solutions based on the information about RTAS.

II Population estimation: calculating of cost, execution times, goal function for population members, and average and maximum reliability of the population. Fixating the best current solution.

On the step II, for each solution, the RTAS simulation model is updated to take in account the timing overhead of FT mechanisms defined by this solution. Then, the updated model is executed to calculate the execution times of the software on each subsystem for the given solution.

III Choosing solutions for the next population. At this moment, the algorithm uses the proportional selection scheme to choose an element from the population.

IV Crossover operation is applied to elements of the new population. The elements for crossing are taken from the set chosen on step III. A single-point crossover is used, where crossover point corresponds to the bound of a subsystem's integer substring. The probability of crossover is P_{cross}. New population is created after crossover, which contains $(100 - N_{\text{cross}})\%$ of best solutions from the current population and $N_{\text{cross}}\%$ of best solutions after crossover.

V Elements of the new population are mutated. Defined percentage (N_{nmut}) of best solutions are not mutated. All other solutions are mutated with probability P_{mut}.

VI Repeat step II.

VII If the number of iterations exceeds the given number without improvement of the goal function's value, then go to step IX; otherwise, go to step VIII.

VIII Perform the adaptive control procedure and go to step III.

IX Finish the execution.

All of the parameters P_{cross}, N_{cross}, N_{nmut}, and P_{nmut} are initially specified by the researcher and automatically modified during the execution of the method.

Adaptive control procedure is applied for automatic change of values of crossover and mutation parameters. Detailed description of this procedure is given below.

For previous and current populations, the average and the maximum reliabilities of the population are estimated. Let introduce some notations:

R_i^{prev} — reliability of the ith member of the previous population;

R_0^{av} — average reliability of solutions from the previous population:

$$R_0^{\text{av}} = \sum_{i=1}^{n} \frac{R_i^{\text{prev}}}{n};$$

$R_0^{\max}$ — maximum reliability of solutions from the previous population:

$$R_0^{\max} = \max_{i=1,\ldots,n} R_i^{\text{prev}};$$

R_i^{curr} — reliability of the ith member of the current population;

R_1^{av} — average reliability of solutions from the current population:

$$R_1^{\text{av}} = \sum_{i=1}^{n} \frac{R_i^{\text{curr}}}{n};$$

$R_1^{\max}$ — maximum reliability of solutions from the current population:

$$R_1^{\max} = \max_{1,\ldots,n} R_i^{\text{curr}}.$$

Each of selection, crossover, and mutation parameters take one of three different values ("large," "normal," and "small") which are defined by the researcher. These values are assigned as shown in Table 1.

For example, if average reliability of the population increases from previous to current population, then, according to Table 1, it is needed to increase the number of solutions after crossover entering the next population and probability of crossover; and otherwise, to decrease the number of solutions for crossover and probability of crossover.

The proposed method was implemented as a software tool integrated with DYANA environment [14]. See [15] for detailed description of this tool.

Table 1 Adaptive control procedure

Condition	$R_0^{\text{av}} < R_1^{\text{av}}$	$R_0^{\text{av}} \approx R_1^{\text{av}}$ ($\sim 3\%$)	$R_0^{\text{av}} > R_1^{\text{av}}$
$R_0^{\max} < R_1^{\max}$	N_{nmut} is large P_{nmut} is small N_{cross} is large P_{cross} is large	N_{nmut} is large P_{mut} is normal N_{cross} is large P_{cross} is normal	N_{nmut} is normal P_{mut} is large N_{cross} is normal P_{cross} is small
$R_0^{\max} \approx R_1^{\max}$ ($\sim 3\%$)	N_{nmut} is normal P_{mut} is small N_{cross} is normal P_{cross} is large	N_{nmut} is small P_{mut} is normal N_{cross} is small P_{cross} is normal	N_{nmut} is small P_{mut} is large N_{cross} is small P_{cross} is small

6 CASE STUDY AND COMPUTATIONAL RESULTS

Previously, AHGA was compared to a method based on classic GA in [16]. In this paper, AHGA is compared to AHGA with penalty function. In this example, RAP is considered for a distributed RTAS with following characteristics:

$n = 8$ — number of subsystems within the distributed RTAS;

$p_i = 5$ — number of hardware component versions available for each subsystem;

$q_i = 5$ — number of software component versions available for each subsystem;

R_{ij}^{hw}, R_{ij}^{sw} are given on the interval $[0.85, 0.99]$;

C_{ij}^{hw}, C_{ij}^{sw} are given on the interval $[5, 35]$;

experiments were performed for total cost constraints of 650, 725, and unlimited;

pure execution times for pairs (software component version, hardware component version) are given in the interval $[100, 300]$;

three classes of time constraints are considered:

(1) HC — hard constraints for every subsystem: 395, 749, 997, 422, 949, 1675, 2189, 2429;

(2) AC — average constraints are hard constraints, increased by 50; and

(3) UC — no time constraints.

Structure of the RTAS is shown in Fig. 2. Vertices of the graph correspond to subsystems, edges — to links between subsystems.

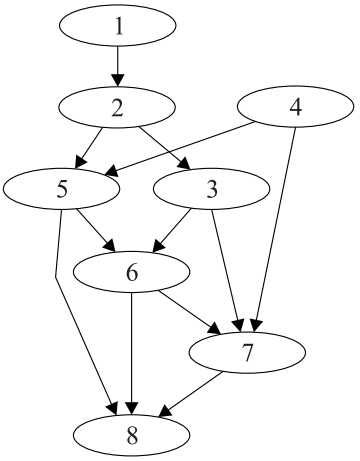

Figure 2 Structure of the RTAS

The following parameters of AHGA were used in experiments. For parameters from Table 1, the following "small," "normal," and "large" values are given:

- percentage of solutions not mutated N_{nmut}: $\{0.05; 0.09; 0.13\}$;
- probability of mutation P_{mut}: $\{0.5; 0.75; 1\}$;
- percentage of solutions chosen for crossover N_{cross}: $\{0.80; 0.85; 0.90\}$; and
- probability of crossover P_{cross}: $\{0.5; 0.75; 1\}$.

Population size of 50 was used. Number of iterations without improvement of the goal function was set to 50.

Table 2 Average RTAS reliability and average number of algorithm's iterations

Cost constraints	Time constraints	AHGA		AHGA with penalty	
		Reliability	Iterations	Reliability	Iterations
650	HC	0.4878	50.78	0.8080	125.79
	AC	0.5410	51.7	0.8112	120.22
	UC	0.7411	58.37	0.8442	124.98
725	HC	0.5798	52.76	0.8310	130.85
	AC	0.6535	56.73	0.8381	131.11
	UC	0.8552	65.05	0.8676	126.18
Unlimited	HC	0.7009	65.41	0.8658	158.48
	AC	0.8142	96.2	0.8734	153.1
	UC	0.8930	148.25	0.8929	144.77

The results of the experiments are given in Table 2.
Table 2 shows that:

- AHGA with penalty function produces better solutions than AHGA;
- AHGA performs lesser number of iterations than AHGA with penalty function and classic GA [16]; and
- if the system has no time constraints, then AHGA performs on par with AHGA with penalty function.

7 CONCLUDING REMARKS

In this paper, the RAP for RTAS is considered and a method for solving this problem is suggested. This problem was previously explored by many researchers [7, 8]. Key features of the suggested method is the use of simulation modeling for execution time estimation of RTAS which implements the chosen FT mechanisms and the use of an adaptive hybrid genetic algorithm. Case study demonstrates the applicability of method based on AHGA to solving RAP.

The directions for future research include:

- more thorough exploration of the proposed method on other examples of RT systems described in literature and on data from real RTAS (unfortunately, the authors of the existing algorithms performed experiments on simplified RTAS examples only);
- research of methods for multiobjective optimization of the system's reliability and real-time performance in the presence of faults; and
- research on estimating execution time of software components: metamodels as a replacement of computer simulation.

REFERENCES

1. Avizienis, A., J.-C. Laprie, and B. Randell. 2004. Dependability and its threats — a taxonomy. *IFIP Congress Topical Sessions*. 91–120.
2. Laprie, J.-C., and A. Coste. 1982. Dependability: A unifying concept for reliable computing. *12th Fault Tolerant Computing Symposium Proceedings*. 18–21.
3. Lyu, M. R. 1996. *Handbook of software reliability engineering*. Ed. M. R. Lyu. IEEE Computer Society Press; McGraw-Hill Book Co. 819 p.
4. Wattanapongsakorn, N., and D. W. Coit. 2007. Fault-tolerant embedded system design and optimization considering reliability estimation uncertainly. *Reliability Eng. Syst. Safety* 92:395–407.

5. Smeliansky, R. L. 1993. Program behaviour invariant as the basis for system performance estimation. *PaCT-93 Conference Proceedings*. Obninsk, Russia.
6. Chern, M. S. 1992. On the computational complexity of reliability redundancy allocation in a series system. *Oper. Res. Lett.* 11:309–15.
7. Holland, J. N. 1975. *Adaptation in natural and artificial systems*. Ann Arbor, MI: Univ. of Michigan Press.
8. Gen, M., and Y. S. Yun. 2006. Soft computing approach for reliability optimization: State-of-the-art survey. *Reliability Eng. Syst. Safety* 91:1008–26.
9. Kuo, W., and R. Wan. 2009. *Recent advances in optimal reliability allocation: Handbook of military industrial engineering*. Eds. A. B. Badiru and M. U. Thomas. No. 10.
10. Tillman, F. A., W. Hwang, and W. Kuo. 1977. Optimization techniques for systems reliability with redundancy — a review. *IEEE Trans. Reliability* R-26:148–55.
11. Misra, K. B. 1986. On optimal reliability design: A review. *Syst. Sci.* 12:5–30.
12. Kuo, W., and V. R. Prasad. 2000. An annotated overview of system-reliability optimization. *IEEE Trans. Reliability* 49:487–93.
13. Wattanapongsakorn, N., and S. P. Levitan. 2004. Reliability optimization models for embedded systems with multiple applications. *IEEE Trans. Reliability* 53:406–16.
14. Bakhmurov, A. G., A. P. Kapitonova, and R. L. Smeliansky. 1999. DYANA: An environment for embedded system design and analysis. *32nd Annual Simulation Symposium Proceedings*. San Diego, CA, USA. 50–57.
15. Volkanov, D. Yu., and A. A. Sharov. 2005. Software tool of fault injection for real-time system's reliability estimation with using simulation. *2nd Conference for Methods and Tools for Information Processing Proceedings*. Moscow, Russia. 457–64. [In Russian.]
16. Bakhmurov, A. G., V. V. Balashov, V. N. Pashkov, R. L. Smeliansky, and D. Yu. Volkanov. 2009. Simulation modeling based method for choosing an optimal set of fault tolerance techniques for real-time systems. *3nd European Conference for Aerospace Sciences Proceedings*.