DOI: 10.18372/2310-5461.65.19923
UDC 004.4:006.015.5

Ihor Raichev, Ph.D., Associate Professor
State university «Kyiv aviation institute»
https://orcid.org/0000-0003-4058-1236
email: ihor.raichev@npp.kai.edu.ua;

Nataliia Denysenko**,** senior lecturer
State university «Kyiv aviation institute»
https://orcid.org/0009-0007-1968-1909
email: nataliia.denysenko@npp.kai.edu.ua

METHODS AND MEANS OF APPLYING ONTOLOGICAL MODELS IN THE PROGRAM SYSTEMS LIFECYCLE OF FLIGHT CONTROL

Introduction

At the present stage, there is a significant increase in the trend of objectively assessing the compliance of program systems (PSs) under development with quality requirements. The process of certification testing is of particular importance for critical PS, which are widely used in aviation, energy, medicine, environmental monitoring, and other fields directly related to the safety of the controlled object. A distinctive feature of such systems is the high requirements for safety, functional correctness, accuracy, and efficiency. Therefore, for a wide range of critical systems, the task of developing methods for verifying PS compliance with established requirements within the subject area (SA) and analyzing risks and hazards throughout their life cycle (LC) becomes highly relevant.

We will examine a typical class of critical systems, namely flight control software (FCSW) for flight vehicles (FV), which are the objects of control. Such systems require validation of multiple functions to ensure compliance with established requirements. The quality of FCSW significantly affects the operational characteristics of automated flight control systems (AFCS). Compliance assessment can be carried out by evaluating (at the testing stage) the overall quality level of AFCS SW, considering that the quality level of each indicator within the unified system quality model [1] must meet the minimum acceptable value specified in the PS requirements.

To comprehensively analyze the compliance of system attributes with the specified requirements, we will develop a formal conceptual model within the subject area. Based on this model, we will technologically form a set of objects, fundamental functions, and methods for their implementation.

Analysis of Recent Research and Publications

In publications [2; 3], using ontology tools, the relevance of studying aspects of subject area for decision-making processes, including in the context of ambiguous problem situations, has been analyzed and confirmed. In publication [4], Burov emphasizes the importance of ontological models for solving tasks in the development of various PS, utilizing formalized knowledge representation and processing for these systems' SA.

Using the focus and results of scientific publications [2; 3; 4], which are presented in a general form, it is important to note that for the class of flight control SW, the primary task is to determine whether the FV (the control object) remains within the "acceptable range" (normal mode) or has exceeded it. This decision is made based on calculations performed on a set of control functions (tasks). Therefore, for the class of automated flight control system (AFCS) SW, the application of ontological research methods within their SD (Subject Domain, in this specific area) can be highly effective.

Studies on the area of flight control have been conducted in publications [5; 6; 7], yielding solutions for the technical diagnostics of aviation equipment and FV being monitored during operation. These studies developed relevant algorithms and offered a new structural safety framework for state aviation, along with appropriate flight control systems. In publication [8], modern results regarding data retrieval and post-flight processing of flight information were obtained, while in publication [9], research was conducted on assessing the quality of parametric data filtering methods and improving the AFCS filtering subsystem.

However, in publications [5–9], ontological studies of the flight control SD were not conducted,

even though this area is defined for each type of FV by corresponding standards and regulatory documents.

Problem definition and its actuality

Flight information (Flight Data, FD), which includes trajectory parameters, as well as information of FV functioning and crew actions to control it, is parametric and should be used to monitor the state of the FV and crew actions. The flight information is processed by FV on-board control algorithms or by ground systems after landing [10]. Processing of information recorded by flight data recorders (FDRs) is an urgent task that allows to ensure flight safety. The ICAO documents highly appreciate the information of FDRs and it is recommended to implement the processing of FD in order to prevent aviation accidents, study crew actions and improve FV maintenance. In Ukraine, the processing of flight data by all FVs operators is obligatory [11].

SW of the automated control systems class solves the following general tasks:

– reproduction of parametric information (visualization of FV flight parameters);

– control of the control object's parameter outputs for limitations (tolerance control);

– control of the FV quality functioning, as an object of control;

– preparation of graded characteristics (depend on the calibration of FV sensors) and filtering of FD.

Depending on the purpose of AFCS SW, these tasks can be solved both in real time and after the control object has been operated by ground-based processing systems of flight data (i.e. parametric digital sensors information). Therefore, the software of this class of systems consists of a subsystem of parametric information playback (parametric information reproduction subsystem), a subsystem of admission control (tolerance control subsystem), a quality control subsystem of the control object functioning (in our case, FV), and a subsystem of the graded characteristics and filtering of FD (Fig. 1).

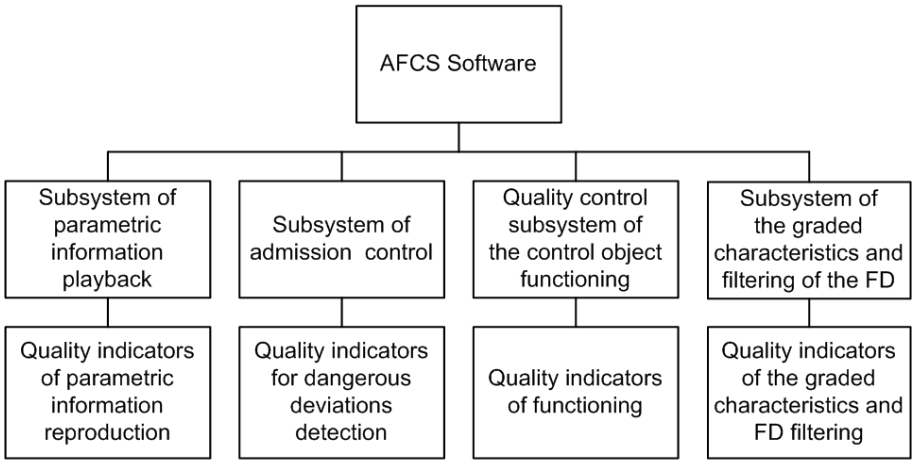

Fig. 1. Subsystems composition and classification of AFCS SW quality indicators

One of the main tasks is to reproduce flight information, which consists in graphical representation of changes in analogue parameters (AP) and one-time commands (OTC) over time. The parametric information display subsystem is present in any AFCS. It provides automated processing of the flight copy with the following output of graphical and tabular information of flight parameters, as well as visualization of the FV flight trajectory on external displays.

The main problem faced by the AFCS SW is solving the tasks of tolerance control. The corresponding subsystem should ensure the implementtation of control algorithms established for each FV type. The subsystem is characterized by making a decision about the FV being in a certain state. For this purpose, a certain set of predicates is calculated and the occurrence of control events is established (or not). A significant problem for the subsystem of admission control is to determine the probability of occurrence of control events and confirmation of the fact that the assessment error is within the tolerance limitations [12;13].

The task of flight quality control is to monitor the performance of flight operation modes and rules by FV crews with the required level of reliability, which is implemented by the corresponding subsystem of the object's operation quality control. This subsystem is based on the results of the FV's tolerance control.

The subsystem of graded characteristics and filtering of FD performs the tasks of analyzing parametric FD, removing emissions, distortions, and noise present in it, and bringing areas of distorted

information from FV instruments and sensors closer to reliable values for the purpose of further processing this information by other subsystems.

The consequences of AFCS SW poor quality and false results of the basic subsystems that assess the state of the control object (Fig. 1) can lead to catastrophic consequences [9; 14]. Indeed, even in the case of ground-based flight information processing systems, in the event of an incorrect assessment of piloting and operation of FV units and equipment, there are risks of allowing further flights by an untrained crew or a FV with equipment defects. These factors may cause an aviation incident that may directly affect the safety of human life and the integrity of the FV. In the case of onboard real-time flight control information systems, an incident or disaster will inevitably occur during the actual flight.

Therefore, the problem of improving the operation of objects safety controlled by means of AFCS is always relevant [12; 13]. One of the ways to achieve this goal is to ensure high-quality and reliable operation of AFCS SW as critical systems, which is especially important when operating the tolerance control subsystem.

The aim of the article is to develop methods and tools for creating an ontological model of AFCS SW components and its effective application during the LC of such systems. The ontological model will be used to build a mathematical model for representing the functions (tasks) of the ontology, which will be used to solve and calculate tasks of the same type in the LC processes, in this case, during the creation of the concept, requirements analysis, programming, and testing of a set of software components that implement the tasks of FV flight control [14].

Domain analysis and ontology model of the flight control SW application domain

At each stage of a FV flight, it is necessary to monitor a significant number of control events (algorithms) simultaneously, which is a non-trivial and rather complex task. Indeed, the object of control can be either in the ‘normal’ mode or not, depending on whether the FV parameters are within the permissible ranges with high reliability. To solve this problem, it is proposed to use an ontological model of the application domain of FC SW, taking into account the objects and functions (tasks) of the ontology with further formalization of these tasks.

The most modern and closest to the practical implementation of the PS in a specified subject domain is the approach that presents the analyzed subject domain (the domain of the PS application) in the form of a structured set of concepts (notions, terms) at the functional level of abstraction. Within the domain, such a general conceptual model of the system is formed, which provides structuring of knowledge, selection of ways of its representation, as well as implementation of the search for functional structural and algorithmic elements of the software, which directly lays the foundation for further technical implementation of a real software system.

Let us begin the domain analysis with a review and analysis of the industry standard for ground-based automated flight information processing systems [10], consisting of software packages (subsystems) shown in Fig. 1. Let's highlight the main objects of the ontology of the SD of AFCS SW (where SD is marked as "Subject Domain"), taking into account the provisions of the standard [10], which establishes general requirements for the characteristics of such systems.

The formal model of the AFCS software ontology is an ordered triple of finite sets:

$$\Omega_{\text{AFCS SW}} = \langle C, R, F \rangle, \quad (1)$$

where C is a finite non-empty set of notions, *concepts* or terms (concepts are based on SD objects), R is a set of *relations* between SD concepts, F is an interpretation *function* defined on the concepts and/or relations of the ontology Ω_{AFCSSW}.

The standard [10] sets requirements for the general characteristics (properties) of components of any AFCS SW implementation, and systems that meet the requirements of this standard and can be certified in accordance with the established procedure and are allowed to be operated by aviation enterprises in Ukraine. The ontology terms define and describe a set of basic objects that are in some relationship with each other. Therefore, the main C terms (concepts) of the SD of AFCS SW ontology will define objects (or a list or a set of objects) that are general descriptive artefacts of the conceptual representation of requirements (during design and programming, they can be implemented as a hierarchy of container-type classes with the corresponding attributes and methods).

In order to create an ontology of the SD of AFCS SW, we will use the IDEF5 ontological research standard [15] and a special schematic graphical language (Schematic Language, SL) designed to represent the system of SD basic data in the form of ontological information. Schemas of any ontology according to IDEF5 should define a system of concepts (a set of basic objects), their classification, composition, relationships and states. Analyzing the standard [10], let us first of all build a diagram of the requirements classification for AFCS components (Fig. 2).

The class «Requirements for the list of the obligatory tasks» is subordinated to the class «Requirements to the main components of the AFCS SW». It is the most important of all (Fig. 2) and is

closely related to other basic classes of the ontology (1), which directly depend on it.

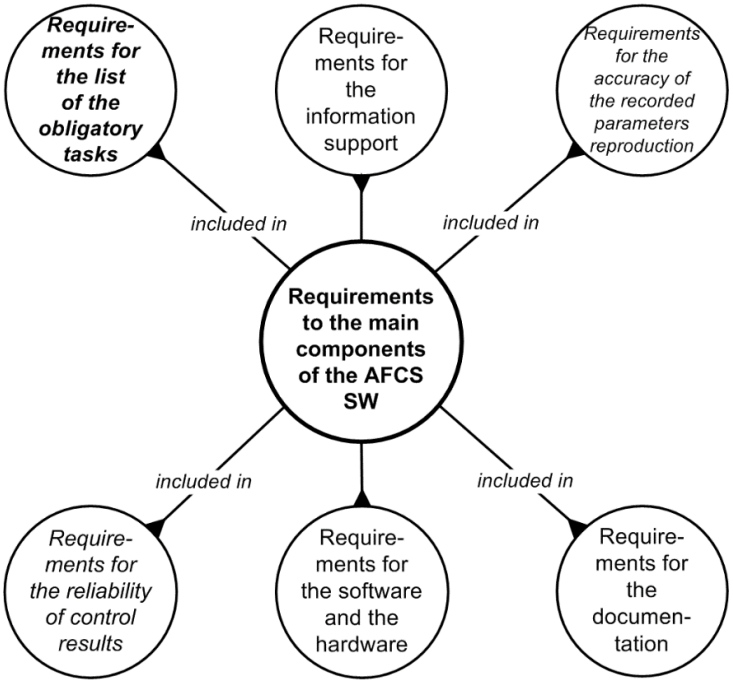

Fig. 2. Ontology of requirements constituents for AFCS SW (conceptual scheme of requirements classification)

Therefore, the compositional scheme of the class «Requirements for the list of the obligatory (mandatory) tasks» (Fig. 3) is essentially a graphical representation of the composition of the ontology classes, where the set of relations between objects *R* consists of classification relations and generalization and aggregation relationships. The compositional scheme of requirements is shown in Fig. 3 [14].

The class «Requirements for the list of the obligatory tasks» is the parent class, including the class «Requirements for the reproduction of a registered FD», which in turn aggregates the «FD file copy» and «Description of FV and airport options» (Fig. 3). The last class contains the «Description of algorithms for detecting dangerous deviations in the actions of the crew and FV systems», where relations and links with the classes of FV control events are placed. Thus, the set of control algorithms becomes known and available for the class «Requirements for dangerous deviations detection in the FV flight».

Let us consider the risks of malfunctioning of the main complexes (subsystems) of the AFCS SW (Fig.1). The complex of FD reproduction programs solves the problem of reproducing flight data, which is a graphical representation of the change in AP and OTC over time. In the complex operation, there are dangers of inaccurate calculation of physical values of the AP and discrepancies (divergences) in the results of repeated FD processing.

The complex of admission control programs solves the tasks of tolerance control. The corresponding software should ensure the implementation of all control algorithms established for a given type of FV and is characterized by making a decision on whether the object of control is in a certain state. A significant problem for this complex in decision-making tasks is the correct determination of the probability of occurrence of control events and confirmation of the fact that the assessment error is within the tolerance that affects decision-making.

The complex of programs for controlling the quality of the facility's operation solves the task of controlling the quality of flight performance, which consists in monitoring the crews' compliance with flight operation modes and rules. The complex should implement control algorithms in the amount not less than that established by the Chief Designer of each FV type and have a procedure for confirming detected deviations. The hazards for this complex, as well as for the complex of tolerance control programs, are the possibility of inaccurate calculation of type I and type II errors by the relevant functions, which also affects the reliability of the decision.

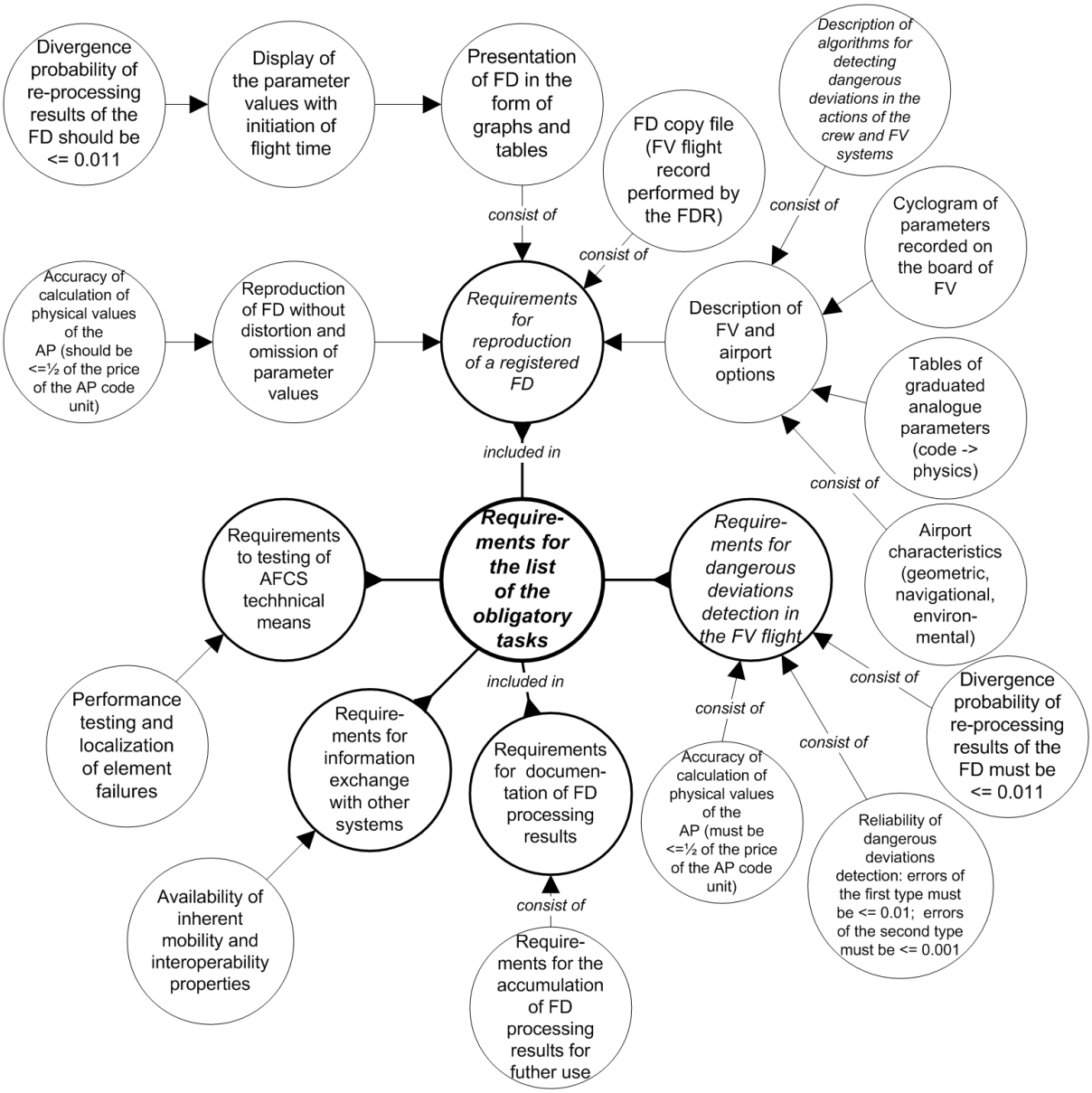

Fig. 3. General requirements ontology for the list of obligatory tasks of the AFCS SW (composite requirements diagram)

Let us introduce the concepts of hazards and risks into the ontology (1) and build a diagram (scheme) of the relationships between requirements and hazards of their non-compliance for the AFCS subsystems (Fig. 4).

The interpretation functions *F*, which are set on the relations of the ontology (1), for the scheme from Fig. 4 are the predicates of each of the FV control algorithms. For example, for the class "Requirements for admission control of FV flight parameters" (Fig. 4), one of the controlled algorithms sets is the logical expression of the control event class, which means "Exceeding the maximum operational flight speed" at the "flight according to the route" stage:

$$S073 = (7120 \le H_{б} \le 10300) \wedge (V_{пр} \ge 588),$$

where the following values are used: $H_{б}$ – barometric altitude, $V_{пр}$ – indicated airspeed. Such interpretation functions (tolerance control near boundary values in predicates) should cover the full set of control algorithms for each FV type.

From the above analysis, it follows (Fig. 4) that in the process of building such classes of critical information systems, it is necessary to apply the specifications of requirements for reliability and safety properties during the operation of SW.

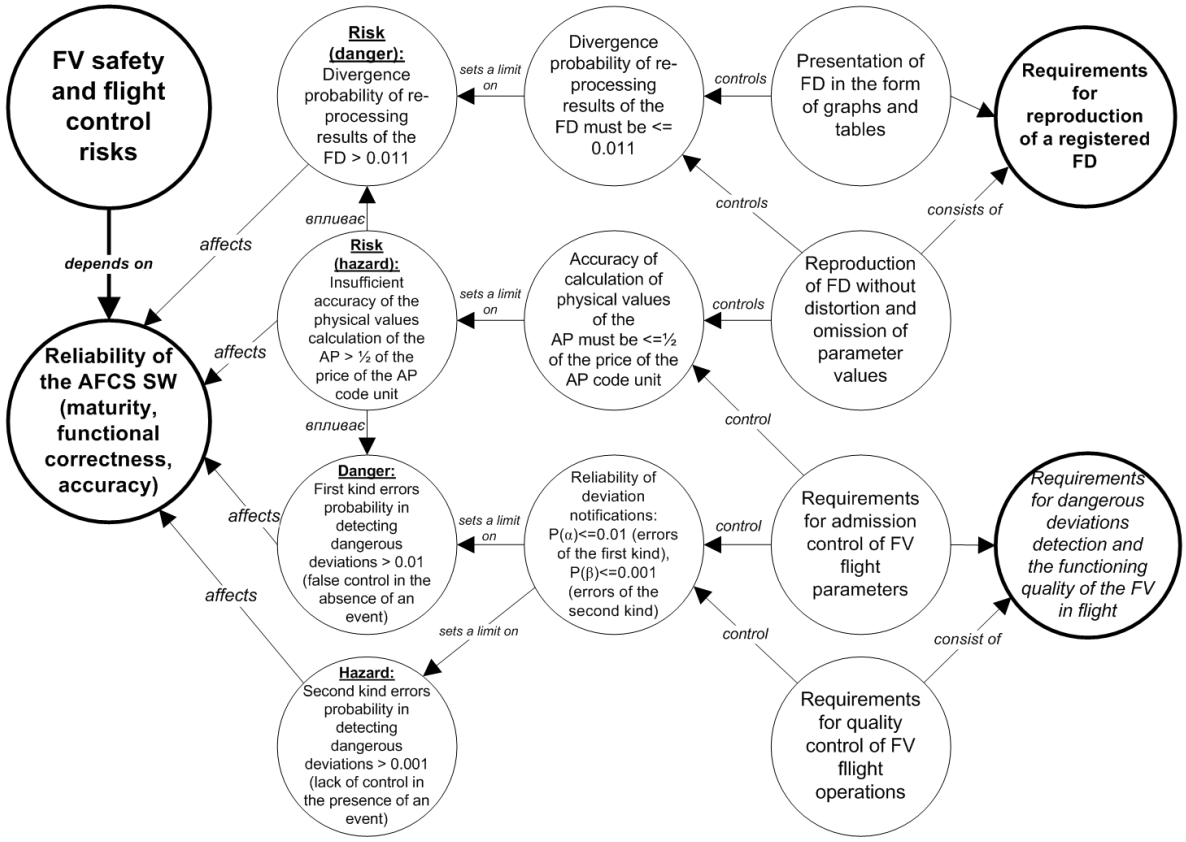

Fig. 4. Ontology of FV flight control (diagram of interconnections between requirements and hazards and risks)

Formation of criteria specifications and constraints for decision-making tasks in critical systems

Verification of full compliance of software with the requirements is necessary for critical systems, the non-compliance of which leads to dangerous situations. SW security requirements for critical systems are determined by analyzing the potential hazards and risks that may arise during system operation. Based on this analysis, we specify software requirements in such a way as to either eliminate these hazards and risks or reduce their impact. This procedure can be performed based on the concept of a «safe» SW life cycle as specified in the IEC 61508 standard [16]. The paradigm of the safe SW life cycle of critical systems is based on the following sequential rules: 1) analysis of the SD system's properties and determination of the PS functionality; 2) analysis of hazards, failures and risks for the SW; 3) distribution of requirements by subsystem functions and specification of requirements for non-functional quality characteristics of safety (freedom from risks) and reliability of operation [1].

The hazard and risk analysis consists of a comprehensive analysis of the system and its domain of use, which was carried out above. As a result, potential hazards were identified and a classification of hazards and risks was performed (Fig. 4). After that, those hazards are selected that are potentially the most dangerous and likely to occur. For the hazards resulting from the selection, the possible causes of their occurrence are analyzed, followed by the construction of a tree of failures and dangerous errors [17]. After that, the risks associated with the identified hazards are assessed, which affect the decision-making on the state of the controlled object.

Correct decision-making about the state of the FV depends on the correct functioning of the control SW. For the decision-making task, we will analyze the mechanisms for preventing the occurrence of hazards and reducing the probability of risks. During the functioning of flight control SW, the set of hazards and risks can be described in the form of incorrect determination of the state of the FV in controlled situations. Based on the system analysis, we determine that a potentially serious hazard that may arise due to poor-quality SW that can arise by false determination of the FV state as an object of control. In this case, there are two possible false decisions:

$$\alpha = S_i\left(\overline{x_i} \in X \,/\, x_i \in \overline{X}\right),$$

and

$$\beta = S_i\left(\overline{x_i} \in \overline{X} \,/\, x_i \in X\right).$$

Here $\overline{x}_i$, x_i – is the actual and calculated value of the vector of controlled parameters at the moment of time t_i, X and $\overline{X}$ – is the acceptable and

unacceptable value range, S_i is the algorithm of the *i*-th flight control event.

Since these solutions can be implemented with a certain probability, the following *constraints criteria* are imposed due to technical and economic factors: $P(\alpha) \le \alpha_d$ and $P(\beta) \le \beta_d$. Here α_d and β_d – are rather small boundary values of event probabilities. The described solutions will be interpreted as criteria with constraints for the FC SW. Note that the probabilities $P(\alpha)$ and $P(\beta)$ will determine the errors of the I and II kind, which can be used as a metric of the attribute «control reliability», which can be attributed to the characteristic of functional suitability, subcharacteristics of functional correctness, accuracy (system and SW quality model [1]).

Taking into account the flight control ontology (1), as well as the scheme of interrelationships between requirements and hazards and risks (Fig. 4), we will summarize the significant hazards and risks for the AFCS as components of the operational hazards for any critical FV type.

When analyzing multi-criteria situations of object control, we will assume that the complete input domain of the program data *D* is composed of a set of complete input domains of parameters D_h included in the formulas of control algorithms and are independent. Any event control algorithm is a general logical expression *P* (aggregate predicate) that can be represented in a perfect conjunctive normal form (PCNF) (usually abbreviated as CNF, omitting the word «perfect») in the form:

$$P = P_1 \cdot P_2 \cdot \ldots \cdot P_h \cdot \ldots \cdot P_H,$$

where *H* – is the number of predicates in the control algorithm $(h = \overline{1, H})$, and P_h – is an arbitrary control predicate.

The criteria for making a decision on an event must comply with the *constraints* and are based on the calculated value of the general logical expression (aggregate predicate) of the control algorithm (*true* – the control event occurred, *false* – the control event did not occur). At the same time, for each variable of each predicate included in the PCNF of the control algorithm, during its calculation, an analysis and verification of the presence of sufficiently small values of errors of the I and II kinds within the permissible limits (0.01 and 0.001) is additionally carried out, especially taking into account the presence of each predicate variable near the boundaries of its equivalence classes, which are formed on the basis of the conditions of inequalities or equalities in the predicate.

It is the magnitudes of errors of the I and II types (a sign of reliability) that we will take as constraint *L* for the decision-making criteria and consider *L* as a sign of *constraint* for each *decision-making criterion*, since each criterion is based on the value of the calculated predicate taking into account these constraints, and the values of the constraints calculated by the program components directly affect the final decision.

Taking into account the results obtained, as well as the conclusions of publications [18; 19; 20], we introduce criteria with constraints *L* into the ntology model. Then formula (1) takes the form:

$$\Omega_{\text{AFCS SW}} = \langle C, R, F, L \rangle. \quad (2)$$

Thus, *C* concepts are considered as objects that form classes of *control events* at the lower level of the hierarchy. Control events are indivisible lower-level objects that contain an event name, an event description (in natural language), and a control algorithm, and each of these algorithms represents one interpretation function from the set *F*. In addition, for each interpretation function there is *a decision-making criterion,* that is $\forall F_i \in F \exists L_i \in L$. The restriction in the criterion determines whether the FV is in the «normal mode» (the control event did not occur) or «not in the normal mode» (the control event occurred: failures in the operation of FV units and systems, or piloting violations by the crew). The main task of the control PS is to accurately diagnose flight control events. Particularly important consequences of these systems are the complete and reliable detection of dangerous deviations in the operation of FV systems and crew actions, taking into account the probability of errors of the first and second types, which should be within the tolerances. It is criteria for compliance with the constraints in the decision-making tasks of ontology (2).

For example, the class «Description of algorithms for detecting dangerous deviations in the actions of the crew and FV systems» (Fig. 3) contains links to many classes that use FV control events, including the class «Requirements for dangerous deviations detection and the functioning quality of the FV in flight» (Fig.4). Control events, in turn, have an association relationship with all ontology classes «interested» in them (including the class «Requirements for admission control of FV flight parameters») and are indivisible objects of the lower level of the hierarchy of ontology classes (1) and (2). The constraints for the criteria of the decision-making task for each of the predicates of the interpretation function can be: «Reliability of deviation notifications: $P(\alpha) <= 0.01$ (errors of the first kind), $P(\beta) <= 0.001$ (errors of the second kind)», «Accuracy of calculation of physical values of the AP must be <=½ of the price of the AP code unit», etc.

Studies carried out in this area have shown that any of the predicates of the FV control algorithms,

i.e. the interpretation function from (2), can be formalized using a system of canonical equations of a finite state machine [21]. Based on this formalization, we will construct logic diagrams of automated models in a unified manner, which will serve to identify controlled dangerous deviations for a given type of FV. Taking into account the limitations of the decision-making criteria of ontology (2), we will use the results of studies of the reliability of detecting control events, which were considered in publications [22, 23].

Application of the automaton theory to the control algorithms programming and data sets tests construction

During the tests for certification of automated flight control systems, there is a problem of creating test sets of data that would cover all possible routes of implementation of programs that implement control algorithms [12, 13]. Indeed, in addition to the control algorithms, which contain dozens of predicates and logical functions, algorithms for finding these controlled situations, which are no less complex logical expressions, are also implemented in the AFCS SW. Considering the fact that there are several hundreds of them, the creation of reliable and efficient control software is a difficult task. PSs are generally created using a direct coding of control algorithms, and this fact creates a high probability of non-detected errors [24] and complicates programs verification. A high degree of nesting of branching operators leads to difficulty in tracing and constructing a comprehensive set of tests, and it causes errors in programs that implement algorithms from control events.

Subsequently, the technology of creating TDS (meaning Test Data Sets) generation programs, which is based on the construction of automatic models, is proposed. The TDS obtained allows you to automate the detection of control events, which makes it possible to significantly increase the efficiency of tests.

One of the ways to solve the problem of covering the graph of control transfers between AFCS SW modules during its testing is the use of formal methods that make it possible to carry out tracing and comprehensive analysis of the logic of control programs and to apply the means of automation of programming and progressive programming technologies (object-oriented, functional, automatic, etc.) [25; 26]. Since the input information for control algorithms is discrete and the output is a finite set of discrete events, the mathematical apparatus of discrete finite machines can be used to model such algorithms [21]. Due to the absence of the control algorithms of existence quantifiers and generality, the use of finite machines theory for the analysis and synthesis of relevant automatic models is much easier. Predicates in control algorithms take the form of standard inequalities and ultimately reduce to single-place predicates.

For example, there exist such predicates: $V_{\text{пр}} \geq 410, \alpha_{\text{руд}} < 110, H_{\text{Г}} \geq 15, H_{\text{Б}} \geq 7120$. Here, $V_{\text{пр}}$ – indicated airspeed, $\alpha_{\text{руд}}$ – throttle lever deviation angle, $H_{\text{Г}}$ – geometric altitude, $H_{\text{Б}}$ – barometric altitude. Automata that implement predicates of this type can be represented as follows:

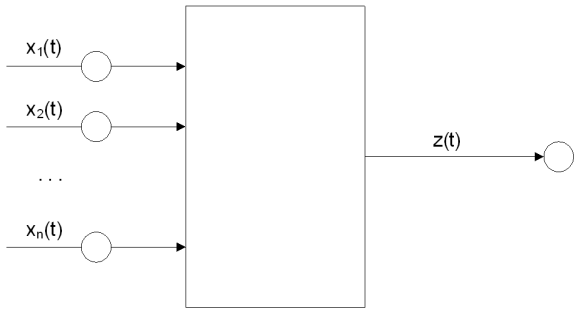

Fig. 5. Functionality scheme of an automaton with one output

In Fig. 5, $x_1(t), x_2(t), \ldots, x_n(t)$ are functions that take values of 0 or 1 and are defined through the corresponding predicates of the given control algorithms. For example, $x_1(t) = V_{\text{пр}} \geq 410, x_2(t) = \alpha_{\text{руд}} < 110, \ldots, x_n(t) = H_{\text{Б}} \geq 7120$ ($V_{\text{пр}}$ – indicated airspeed, $\alpha_{\text{руд}}$ – throttle lever deviation angle, $H_{\text{Б}}$ – barometric altitude).

By simplifying the logical functions and predicates describing the control algorithm and transforming the algorithm into the PCNF (previously deciphered as Perfect Conjunctive Normal Form), we construct a system of canonical equations of a finite automaton [21], which can generally be represented by the following equations:

$$\left.\begin{aligned} z(t) &= \Phi\left[x_1(t), x_2(t), \ldots, x_n(t), q(t)\right] \\ q(t+1) &= \Psi\left[x_1(t), x_2(t), \ldots, x_n(t), q(t)\right] \\ q(0) &= 0, t = 0, 1, 2, \ldots \end{aligned}\right\}. \quad (3)$$

Equation (3) and the functions Φ and Ψ are canonical equations and functions over the alphabets X, Z, Q (X – input alphabet, Z – output alphabet, Q – states alphabet, $q(t)$ – state function of the automaton). Since the output has a single pole where either 1 or 0 appears (indicating whether the event has occurred or not), there is a single resulting function – $z(t)$.

To automate the modelling of control algorithms and the generation of test data sets (TDS), a unified mathematical representation of control algorithms has been developed. For this purpose, a classification

of these algorithms was carried out, and, using the example of AFCS SW, flight control algorithms for aircraft were studied and implemented. This made it possible to synthesize automaton models, through which the problem of automatic detection of control events based on parametric information was formulated and solved, as shown in [21]. According to this classification, first-class control algorithms are identified as memoryless automata, while second-class control algorithms are automata with memory, where the automaton state accumulates the event duration.

The development of automaton models is based on the methodology for synthesizing finite automata, which includes constructing canonical equations and logical schemes of control algorithms. Ultimately, by further programming of automaton models, the problem of software-based detection of control events can be solved, along with the development of an optimal testing methodology for a set of control algorithms. This will allow for the automation of TDS generation, ensuring coverage of execution paths of control algorithms.

The application of the obtained automaton models for programming control algorithms significantly simplifies the design of TDS generation programs for certification testing of AFCSS SW. To achieve this, it is necessary to develop a procedure for building programs that automate the detection of monitored events and hazardous deviations in the operation of the controlled object, while also enabling the generation of TDS that simulate such events. The construction of such programs relies on automaton-based programming, which is based on the use of finite automata to describe program behavior using the C++ language with explicit state representation.

Construction of programs for detecting control situations and creating TDS using automaton models

Since the number of situations during the control of any object is usually significant (several hundred), automating their testing (in-depth testing and analysis of the structural elements of control algorithms) will increase the efficiency of AFCS SW testing as a whole, enhance its reliability, and improve the safety of the controlled object. For test automation, we use automaton models based on finite automata. From the canonical equations of automata, it is possible to develop logical schemes and construct programs that generate TDS and detect control situations, including for assessing relevant quality indicators.

It should be noted that when directly programming complex control algorithms that contain dozens of predicates and logical conditions, as well as several time intervals (durations), the likelihood of errors significantly increases. These errors can be avoided by applying the principles outlined below for constructing correctly functioning programs based on logical schemes of automaton models and by using efficient automaton programming.

We propose constructing program algorithms according to the obtained canonical equations and logical schemes of automaton models that represent the corresponding control algorithms (interpretation functions of ontology (2)). The correctness of the constructed programs is confirmed by the fact that automaton models, as synchronous action devices, accurately interpret control algorithms due to the way they are constructed [21]. The proposed method is based on programming the canonical equations of the automaton.

Programming an automaton model for a control event means constructing the canonical equations and logical scheme of the automaton for the given event and then encoding these equations (the automaton algorithm) in a programming language. In other words, it is necessary to program the automaton's output function, its state function, and describe the structure of input and output data.

It should be noted that the proposed method of programming automaton models differs from the traditional automaton programming methodology in that it is based on pre-constructed canonical equations and logical schemes. This significantly simplifies the coding of automaton models compared to automaton programming, where it is first necessary to construct state tables, develop diagrams, and design the automaton's operational algorithms. The formalism of canonical equations eliminates the need for these steps, as the automaton algorithm and its states are predefined in the equations and the logical schemes derived from them. However, it should be noted that it is not always possible to construct canonical equations for every class of problems [21].

The coding of control algorithms follows a standard scheme for any object-oriented programming language using a developed method that employs standard templates for first- and second-class control algorithms, greatly simplifying program development. The resulting programs allow for the detection of control events and the insertion of events into a copy of the parametric flight information, which reflects the operation of the controlled object, thereby enabling the construction of TDS, as discussed below.

For programming automaton models, we use an object-oriented approach and the C++ programming language. Control events are described using classes, whose elements include the type of algorithm, predicates, automaton states, logical conditions and constraints, input and output signals, as well as tolerance and confidence intervals. The confidence intervals are derived from algorithms developed in publications [22, 23]. Class member functions describe the automaton's function and state function and serve to detect control events, while friend functions of the classes are used to generate TDS for verifying the detection of these events by AFCS modules.

Using automaton models, we have developed software for detecting control events in parametric information and for creating minimal-sufficient TDS.

The concepts of minimality and sufficiency are understood in a practical sense – developed programs iterate over combinations of logical conditions and time durations that lead to the occurrence (or absence) of controlled events, covering the structures of predefined flight control algorithms.

This software can be built based on automaton models programmed in the form of:

1. Programs designed using SDL or UML languages;
2. Programs written in object-oriented languages (C++, C#, Java);
3. Programs based on R-technology.

Let us consider an example of constructing a program based on a typical control algorithm for piloting S012 «Exceeding the continuous heating time of the Constant Pressure Device (CPD) before takeoff», which is recorded as:

$$S012 = \Pi_{\text{рул.}} \wedge i_{\text{пкспит}} \wedge (\Delta\tau_2 \geq 615)\ ,$$

here $\Delta\tau_2$ – a time interval greater than or equal to 615^{th} of a second of registration (the flight data recorder records 2 data points per second); $i_{\text{пкспит}}$ – OTC "The anti-icing system of the CPD is turned on"; $\Pi_{\text{рул.}}$ – a taxiing indicator recorded in this way:

$$\Pi_{\text{рул.}} = \overline{\Gamma T}_{\alpha\text{руд1-3}} \wedge \left[(n_{\text{д1}} < 85) \wedge (n_{\text{д2}} < 85) \wedge (n_{\text{д3}} < 85)\right],$$

where $\Gamma T_{\alpha\text{руд1-3}}$ meaning "readiness of a throttle lever"; $\alpha_{\text{руд}}$ – throttle lever deviation angle; $n_{\text{д1}}, n_{\text{д2}}, n_{\text{д3}}$ – rotor RPM (Revolutions Per Minute) of the 1*st*, 2*nd,* and 3*rd* engines.

$$\Gamma T_{\alpha\text{руд1-3}} = (\alpha_{\text{руд1}} \geq 100) \wedge (\alpha_{\text{руд2}} \geq 100) \wedge (\alpha_{\text{руд3}} \geq 100) \wedge (n_{\text{д1}} \geq 85) \wedge (n_{\text{д2}} \geq 85) \wedge (n_{\text{д3}} \geq 85) \wedge (\Delta\tau_1 \geq 4).$$

Here and after $\alpha_{\text{руд1}}, \alpha_{\text{руд2}}, \alpha_{\text{руд3}}$ – throttle lever position of the 1*st*, 2*nd*, and 3*rd* engines, and $\Delta\tau_1$ – time period greater than or equal to 4 seconds. Then:

$$S012 = \overline{(\alpha_{\text{руд1}} \geq 100) \wedge (\alpha_{\text{руд2}} \geq 100) \wedge (\alpha_{\text{руд3}} \geq 100) \wedge (n_{\text{д1}} \geq 85) \wedge (n_{\text{д2}} \geq 85) \wedge}$$

$$\overline{\wedge (n_{\text{д3}} \geq 85) \wedge (\Delta\tau_1 \geq 4)} \wedge (n_{\text{д1}} < 85) \wedge (n_{\text{д2}} < 85) \wedge (n_{\text{д3}} < 85) \wedge i_{\text{пкспит}} \wedge (\Delta\tau_2 \geq 615).$$

For this control event, we define the following set of predicates:

$$\alpha_1 = \alpha_{\text{руд1}} \geq 100,\ \alpha_2 = \alpha_{\text{руд2}} \geq 100,\ \alpha_3 = \alpha_{\text{руд3}} \geq 100,\ n_1 = n_{\text{д1}} \geq 85,\ n_2 = n_{\text{д2}} \geq 85,$$
$$n_3 = n_{\text{д3}} \geq 85,\ n_{11} = n_{\text{д1}} < 85,\ n_{22} = n_{\text{д2}} < 85,\ n_{33} = n_{\text{д3}} < 85,\ i = i_{\text{пкспит}}.$$

Let's denote $z0\_gt\_a\_rud(t) = \alpha_1(t) \wedge \alpha_2(t) \wedge \alpha_3(t) \wedge n_1(t) \wedge n_2(t) \wedge n_3(t)$. Then the canonical equations for $\Gamma T_{\alpha\text{руд1-3}}$ as the 2nd class algorithm, taking into account (3), will have the form:

$$\left.\begin{array}{l} z\_gt\_a\_rud(t) = z0\_gt\_a\_rud(t) \wedge (q\_gt\_a\_rud(t) \geq 8) \\ q\_gt\_a\_rud(t+1) = z0\_gt\_a\_rud(t) \wedge (q\_gt\_a\_rud(t) + z0\_gt\_a\_rud(t)) \\ q\_gt\_a\_rud(0) = 0, t = 0,1,2,... \end{array}\right\}. \quad (4)$$

To calculate the sum of the automaton system state (4), the well-known operator of the sequential action adder is used, but this is only necessary for constructing the basic electrical circuit of the automaton, and in the case of performing actions to program the automaton to accumulate the duration of the event in time, it is sufficient to use a conventional counter.

Let's denote

$$z0\_p\_rul(t) = \neg z\_gt\_a\_rud(t) \wedge n_{11}(t) \wedge n_{22}(t) \wedge n_{33}(t).$$

Then the canonical equations for the taxiing indicator $\Pi_{\text{рул.}}$ sign as the 1st class algorithm, will be written in the form of system (5):

$$\left.\begin{array}{l} z\_p\_rul(t) = z0\_p\_rul(t) \\ q\_p\_rul(t+1) = z0\_p\_rul(t) \\ q\_p\_rul(0) = 0, t = 0,1,2,... \end{array}\right\}. \quad (5)$$

Finally, let's denote

$$z0\_S\_012(t) = z\_p\_rul(t) \wedge i.$$

Then the canonical equations for the 2nd class control algorithm S012 will be written in the form of system (6):

$$\left.\begin{array}{l} z\_S\_012(t) = z0\_S\_012(t) \wedge (q\_S\_012(t) \geq 1230) \\ q\_S\_012(t+1) = z0\_S\_012(t) \wedge (q\_S\_012(t) + z0\_S\_012(t)) \\ q\_S\_012(0) = 0, t = 0,1,2,... \end{array}\right\} \quad (6)$$

The analysis of the control event (6) is facilitated by the logic diagram shown in Fig. 6. In the diagram, the symbol $\oplus$ denotes the adders of the states of the automata, which can be programmed, for example, using a counter (the state of the automaton stores the duration of the controlled event). Such complex and rather incomprehensible events are difficult to analyze even for an expert. The resulting logic diagram contains a composition of two automata of the 2nd type ($\Gamma T_{\alpha\text{руд1-3}}$ and S012)

and automata of the 1st type ($\Pi_{рул.}$, which includes $ГТ_{\alpha руд1\text{-}3}$), the canonical equations of which are obtained according to the rules developed in [21]. Having the canonical equations and the logic diagram of the automata, it is possible to construct the correct control program according to the rules set in [21]. Let us consider this process in details. Note that in this automaton model the feature of the stage $\Pi_{рул}$ and readiness of $ГТ_{\alpha руд1\text{-}3}$ are implemented, according to which, having built automaton models and programs once, they can be used repeatedly in various control algorithms where they participate (Fig. 6).

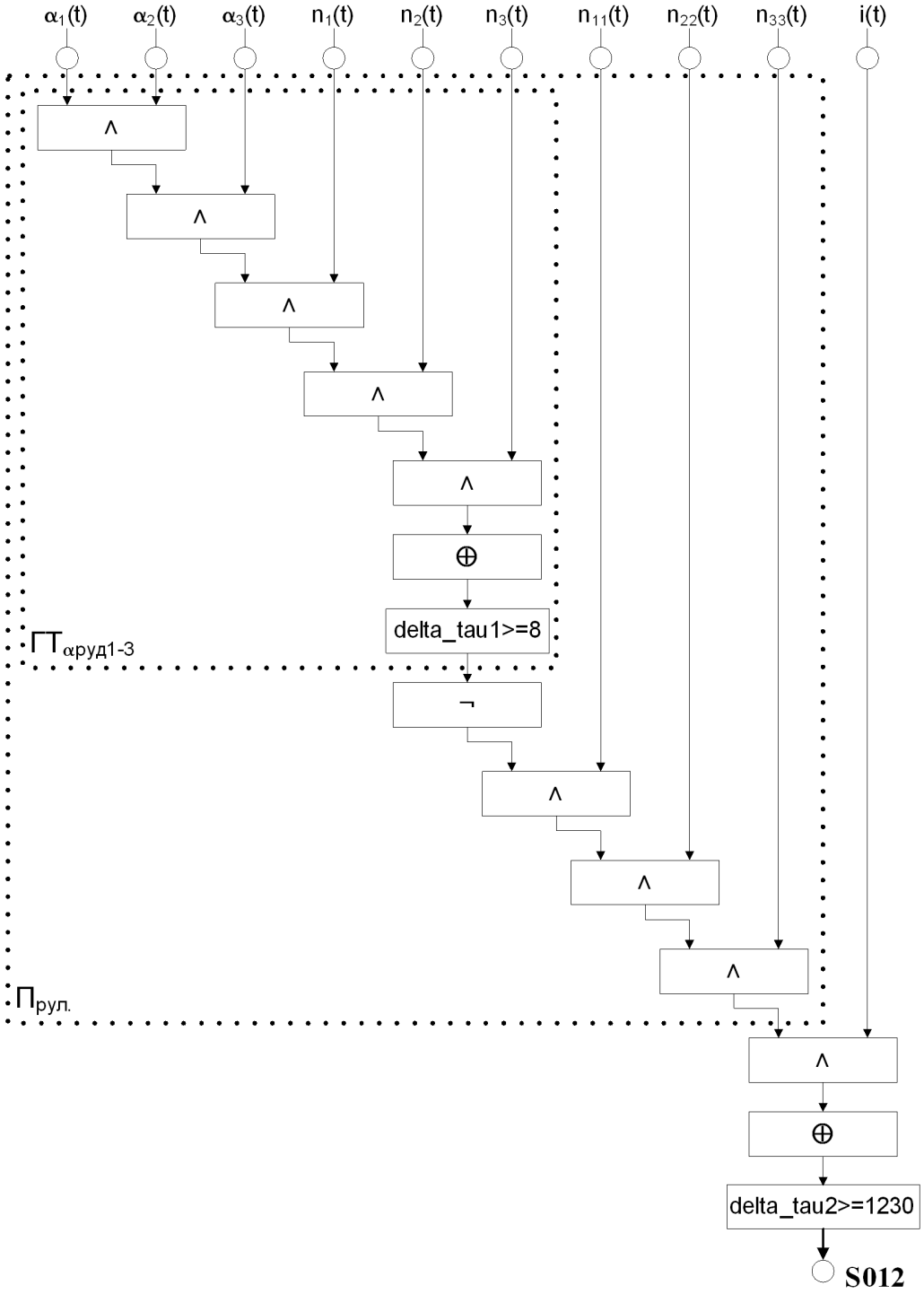

Fig. 6. Logical scheme of the automaton model for the control event S012

If you program an automaton model of any control event, it provides a tool and algorithm for searching and detecting this event. In addition, in order to enter some control events into the copy of the parametric information of the functioning of the control object (i.e., create various TDSs), it is necessary to introduce into the class that describes this event a function (for example, a friendly function of the class), which will, during the program's passage through the file of the copy of the

parametric information, increase (or decrease) the necessary parameters (for example, altitude, speed, engine speed, etc.), changing their values until they reach and exceed the controlled limit value from the confidence interval. In this way, you can build various TDSs, in which the necessary combinations of predicates, logical conditions and durations in time, including readiness and characteristics of flight stages, will be sorted.

At the request of the expert, a given logic coverage from a set of control algorithms can be carried out, that is, the inclusion of the necessary control events in the test data set, as well as the indication of predicates and logical conditions in the composition of each event for which a test needs to be created.

Thus, instead of control programs that may contain logical errors due to the fact that they are built using the direct programming method, control programs that are a priori correct (according to the construction method) are coded by using automatic programming.

Method of programming automatic control models with an example of typical program design and use of design templates

A feature of the of critical systems operation is that during this process, parametric information is recorded, which allows you to reproduce the state of the control object and control effects. Regulatory documents on operation provide restrictions on the parameters of the operation of the control object at certain stages (modes) and in certain situations. Control software, as a rule, is intended to search for events of exceeding the limits of controlled parameters according to the registered information when investigating abnormal situations, accidents and disasters. Therefore, to verify the correct functioning of the AFCS SW at the testing stage, it is necessary to create a TDS, which can be performed by making appropriate adjustments to the registered flight information.

Search and processing of controlled events is performed according to information recorded in accordance with a discrete time series. In the case of finding an event, the function of the automaton becomes equal to one, and at the next point in time becomes equal to zero.As an illustration, we will demonstrate the process of building a program using the example of the S012 control event. We will define an elementary base class for $ГТ_{\alpha руд}$ (gt_a_rud), as well as classes for $П_{рул}$ (p_rul) and S012 (s_012). The p_rul class uses data structures and calls the base function from gt_a_rud, and the s_012 class uses the function of the p_rul class and data from both subordinate classes. Therefore, the last two classes can be considered derived. In contrast, elementary (indivisible) classes do not contain elements of higher classes and do not refer to elements of functions of higher classes.

Predicate is a friendly function of all base classes. It calculates a predicate on a set of input data of the automaton model, as a device for synchronous information processing. We will call this function from the main function of the class *fz*_(), which searches for a control event and returns the main data of the class – *z*. An additional function of the class *cz*_() is also introduced, which will automatically enter the controlled event into the test copy of parametric information, thereby constructing a TDS at a given interval. For this function, a time point and the maximum (or minimum) value of the parameter that must be reached at this point and which exceeds (or is lower than) the limit value is specified. The construction of the event is performed by drawing a smooth upward (or downward) convex cubic parabola in the case of spline interpolation, or a polynomial of degree not greater than (*n*-1), where *n* is the number of interpolation points.

An example of a constructed controlled event is shown in Fig. 7. The technological scheme of searching for control events is implemented in the program algorithm, which is presented below. In a similar way, any programs for determining events based on parametric information are built.

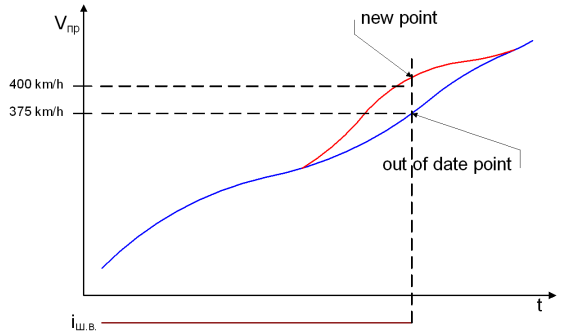

Fig. 7. Entering the control event S068: “Exceeding speed during landing gear retraction on takeoff (400 km/h)” into a copy of parametric information for the purpose of creating TDS ($V_{пр}$ – indicated airspeed; $i_{ш.в.}$ – initial condition – OTC “Landing gear deployed”)

The method of building programs based on automaton models makes it possible to use templates. There are two templates: 1) a program template based on the 1st class control algorithm; 2) a program template based on the 2nd class control algorithm.

To build new classes in the program that represent any other control algorithms, we use these templates in the following way:

1) replace the event, readiness and feature identifiers in the selected template with new required identifiers;

2) change the predicates: when calling the predicate() function, change the constraint type, constraint value, duration, dimension (depending on the type of predicate);

3) change the general form of the automaton function to the required one (the formula that depends on the predicates in the automaton equation is calculated by applying the rules from clause 4).

The proposed method allows you to quite easily, in a uniform manner, build a set of classes that cover a set of specified object control algorithms. The built classes are designed to automatically search for control events in a parametric copy of the flight. Therefore, writing and debugging control programs, as well as programs for building TDS containing control events (entering events into a parametric copy), is significantly simplified. A program for searching for the S012 control event in C++ looks like this:

```
#include  <owl.h>
unsigned int  tic=0;  // current time in ticks from the start of the copy (1 tick = 0.5 sec.)
unsigned short  cadr[128], *pcadr=&cadr[0];  // copy frame (synced with ticks (tic) )
…. // declarations
main ( int argc, char *argv[] )
{ …. // of declaration
 class  gt_a_rud   // class ГТαруд – “readiness of a throttle lever” – control algorithm of the 2nd class
 { private:
   unsigned short  z_gt_a_rud; // the result of the automaton readiness model (0 or 1)
   unsigned short  alpha1,alpha2,alpha3,n1,n2,n3; // input signals (predicate values)
   unsigned short  q1_gt_a_rud; // state q(t+1), counter
   unsigned short  q_gt_a_rud=0; // internal state of the automaton q(t)
   unsigned short  z0_gt_a_rud; // result of calling a logical function ГТαруд
   unsigned int  ctic;  // tick (dot) of the last call
   unsigned int  d_tau; // Δτ (дельта тау, delta tau) – operating time interval
  public:
  void  gt_a_rud() {z_gt_a_rud=0;q1_gt_a_rud=0;z0_gt_a_rud=0;  ctic=0;}   void ~gt_a_rud(); // constructor and destructor
  unsigned short  fz_gt_a_rud ( unsigned int  p_tic, unsigned short  *p_cadr);
  { alpha1 = predicate(1,19,2,100,1,-1,p_tic,p_cadr);    // The predicate function must be described in the general part of the system and must be accessible; its parameters have the following values: 1) type (0 - OTC, 1 - AP, 2 - Δτ); 2) registration channel number; 3) restriction type (0 :> , 1 :<, 2 :≥ , 3 :≤); 4) physical value of the restriction (number); 5) control probability (0 - do not determine, 1 - determine, and if the probability is low, the function returns -1 ); 6) if the OTC type, then sets the bit number in the channel, otherwise the field = -1; 7) tick number; 8) frame address in the copy; predicate calculates a simple single-place predicate and returns 1 when the predicate=true and 0 if predicate=false.
      alpha2 = predicate(1, 23, 2, 100, 1, -1, p_tic, p_cadr);  alpha3 = predicate(1 ,31, 2, 100, 1, -1, p_tic, p_cadr);
      n1 = predicate(1, 25, 2, 85, 1, -1, p_tic, p_cadr);   n2 = predicate(1, 35, 2, 85, 1, -1, p_tic, p_cadr);
      n3 = predicate(1, 43, 2, 85,1, -1, p_tic, p_cadr);  z0_gt_a_rud = fz0_(); // definition of the logic function of the automaton
      if(ctic!=0)
         if ( (p_tic-ctic>1) || (p_tic-ctic<=0) ) { d_tau=0; q_gt_a_rud=0; q1_gt_a_rud=0;}
      d_tau = fq_gt_a_rud(); // the state of the machine accumulates the operating interval
      z_gt_a_rud = z0_gt_a_rud · predicate(2, d_tau, 2, 8, 0, -1, p_tic, p_cadr);
// automaton function according to the canonical equation  z(t) = z0(t) · (q(t) ≥ 8)
    return( z_gt_a_rud);  }
 unsigned short  fz0_( void)   // logical function definition ГТαруд
 { return( alpha1&&alpha2&&alpha3&&n1&&n2&&n3);  }
 unsigned short  fq_gt_a_rud ( void)  // automaton state function q(t)
 {  if (z0_gt_a_rud  = = 1)  {
        q1_gt_a_rud++; q_gt_a_rud = z0_gt_a_rud · (q1_gt_a_rud - 1); }
    else { q1_gt_a_rud=0; q_gt_a_rud=0; }
    return(q_gt_a_rud);   }
friend unsigned short predicate (unsigned short, unsigned int, unsigned short, unsigned int, unsigned short, unsigned short, unsigned int, unsigned short *);
}   // end of class description ГТαруд
class p_rul { // “taxiing feature” – 1st class control algorithm
 private:
    unsigned short  z_p_rul;  // automaton function
    unsigned short  n11, n22, n33; // input signals of engine rotor revolutions
    unsigned short  gt_a; // result of calling the function gt_a_rud (“readiness of a throttle lever”)
    unsigned short  q_p_rul, q1_p_rul; // internal states of the automaton
    unsigned short  z0_p_rul; // logic function of the automaton p_rul
public:
  p_rul() {q_p_rul=q1_p_rul=0; z0_p_rul=z_prul=0;};  ~p_rul(){};  // constructor and destructor
```

```
friend unsigned short predicate (unsigned short, unsigned int, unsigned short, unsigned int, unsigned short,
unsigned short, unsigned int, unsigned short *);
  unsigned short int fz_p_rul ( unsigned int p_tic, unsigned short * p_cadr)
{  g_ta=fz_gt_a_rud(p_tic, p_cadr); // call “readiness of a throttle lever”
    if( gt_a == 0) g_ta =1; else g_ta = 0; // denial of readiness
    n11=predicate(1, 21, 1, 85, 1, -1, p_tic, p_cadr); // 21 ,29, 33– engine rotor revolutions
    n22=predicate(1, 29, 1, 85, 1, -1, p_tic, p_cadr);  n33=predicate(1, 33, 1, 85, 1, -1, p_tic, p_cadr);
    z0_p_rul = fz0_p_rul(); q_p_rul = fq_p_rul(); q1_p_rul = q_p_rul;   z_p_rul = z0_p_rul;
    return( z_p_rul);   }
 unsigned short fz0_p_rul( void)  { z0_p_rul = g_ta&&n11&&n22&&n33; return( z0_p_rul); }
 unsigned short fq_p_rul( void) { q_p_rul = z0_p_rul; return( q_p_rul); }
}  // end of class description p_rul
class  s_012  {  // control event S012 - 2nd class control algorithm
private:

  unsigned short  z_s_012; // result of the automaton model operation of the control event
  unsigned short  i; // OTC “The anti-icing system of the CPD is turned on”
  unsigned short  q1_s_012; // state q(t+1), counter
  unsigned short  q_s_012 = 0; // the internal state of the automaton q(t)
  unsigned short  z0_s_012; // the result of a logical function call S012
  unsigned int  ctic;  // last call tick (point)
  unsigned int  d_tau; // Δτ (delta tau) – functioning time interval
public:
 void  s_012() {z_s_012=0; q1_s_012=0; z0_s_012=0; ctic=0;} ~p_rul(){};  // constructor and destructor
 unsigned short  fz_s_012( unsigned int p_tic, unsigned short * p_cadr)  // event function S012
 {  i = predicate(0, 15, 0, -1, -1, 4, p_tic, p_cadr); z0_s_012 = fz0_s_012();
    if(ctic!=0)
      if ( (p_tic-ctic>1) || (p_tic-ctic<=0) ) { d_tau=0; q_gt_a_rud=0; q1_gt_a_rud=0;}
    d_tau=fq_s_012();  // operating interval
    z_s_012 = z0_s_012 · predicate(2, d_tau, 2, 1230, 0, -1, p_tic, p_cadr);
   return( z_s_012);   }
 unsigned short  fz0_s_012( void)  {
    z0_s_012 = fz_p_rul(p_tic, p_cadr) · i;    return( z0_s_012);  }
 unsigned short fq_s_012( void)  // automaton state function q(t)
 { if (z0_s_012  = = 1)  {
      q1_s_012++; q_s_012 = z0_s_012 · (q1_s_012 - 1); }
   else { q1_s_012 = 0; q_s_012 = 0; }
   return(q_s_012);   }
}   // end of class description s_012
....
 gt_a_rud  g;  // readiness class instance
 p_rul  p;       // an instance of a taxiing class
 s_012  s12;  // an instance of the control algorithm S012
....// operators

 while ( (cadr=read_cadr()) != EOF ) // until the parametric copy of flight is finished
{   tic++; ….  // operators
  if( s12::fz_s_012( tic, cadr) = = 1)  {
     printf(“controlled event found *****”);
     break;  // exit from the cycle
   }
}   //  end of the cycle while
.... // operators
}   // end of the program main()
```

Conclusions

By analyzing the SD of use, an ontology of the SW of automated flight control systems was built. As a result of creating the ontology, the following results were obtained:

1. The created ontology provides the opportunity to:

– systematize and structure knowledge about the concepts and objects of the SD of the FC SW;

– to determine the relationship and interrelationships between the objects of the subject area;

– identify interpretation functions for ontology objects;

– formulate and solve the problem of making a decision about the state of the control object;

– determine the criteria and restrictions for making a decision for interpretation functions;

– build a formal mathematical apparatus of automata models as a basis for interpretation functions;

– cover all stages of the LC development of the FC SW and apply ontology (2) for formalization,

unification and automation of the LC processes both during development (requirements analysis, programming, testing) and at subsequent stages of operation and maintenance of the AFCS SW.

2. The use of the constructed ontology has the following advantages:

– the ontology takes into account all notions, concepts and objects of the SD of use;

– all control events and risks for the control object (FV) are taken into account;

– correct "as built" programs are created for the full set of control events;

– the necessary program control is carried out based on taking into account risks with an emphasis on critical tolerances;

– verification of critical safety conditions of the control object (FV) is supported;

– based on the correct solution of the problem of making a decision on the state of the control object (FV) and due to the reduction of risks, high indicators of reliability and safety of the functioning of flight control systems are achieved.

3. The constructed ontological model and its interpretation functions can be used to develop control systems of any type, since control information systems are characterized by the presence of a control object, which is controlled by a set of parameters that are measured using measuring instruments. Such systems always work with control tasks and events that are formulated similarly to those presented in the article.

As a result of creating automated control models, the following results were obtained:

1. Using an object-oriented approach with explicit selection of states, according to the canonical equations and logic diagrams of automatic control models, a method for designing properly functioning programs for searching for controlled events was developed. The obtained programs allow to automate event detection and generate TDS, which provides a tool for determining the actual quality indicators of detection of dangerous deviations by software components of automated flight control systems.

2. On the basis of automated models, a method for testing control algorithms has been developed, which consists in the automatic generation of TDSs, which are minimum-sufficient coverage of the routes of execution of programs that implement control events. Recommendations for practical tracking of control events in the programming of flight control modules have been developed, which reduces the number of errors in writing and debugging programs.

3. Automatic programming of control events makes it possible to program control algorithms for the full set of control events of an object in the same way ‘according to the scheme’. It follows that testing of control programs built in this way can be performed in accordance with a formalized strategy based on this scheme.

4. On the basis of the obtained method, a strategy for testing the logic of control algorithms is built using test data sets that cover the full set of control algorithms for any type of FV [24]. The Mathematical apparatus from publication [21] can be used to specify requirements for any critical control PS.

Further Research Prospects

1. The methods and tools developed in this paper can be easily modified, adapted and used in other related applications, especially in the SD, where critical PS operate, for which the state of the control object is assessed on the basis of parametric information from sensors or measuring devices (aviation, energy, medicine, environment).

2. In order to improve and increase the efficiency of flight control SW implementation, it is proposed to automatically create software components for processing parametric flight information. For this purpose, the intellectual capabilities of artificial intelligence should be used. The automatic models of flight control algorithms (ontology (2) interpretation functions) built in the article can be effectively implemented with the involvement of artificial intelligence, which can be easily ‘taught’ the typical methods of designing and programming control algorithms considered in the article.

REFERENCES

[1] ISO/IEC 25010. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software Quality models. 2011. 34 p. URL: http://www.iso.org, http://www.iec.ch (access data 20.02.2025)

[2] Буров Є. В., Карпов І. А. Особливості застосування онтологій в системах підтримки прийняття рішень : теоретичний та практичний аспект. *Grail of Science.* 2022. № 12–13. С. 307–310. URL: https://archive.journal-grail.science/index.php/2710-3056/article/view/57.

[3] Карпов І. А., Буров Є. В. Використання онтологічних мереж у системах підтримки прийняття рішень в умовах неоднозначності. *Інформаційні системи та мережі.* 2020. № 7. С. 8–15.

[4] Буров Є. В., Пасічник В. В. Програмні системи на базі онтологічних моделей задач. *Вісник Національного університету «Львівська політехніка».* 2015. № 829. С. 36–57.

[5] Кузьміч О. Є., Аркушенко П. Л., Андрушко М. В., Гайдак І. Г., Пащенко С. В. Розгляд алгоритму експлуатації авіаційної техніки державної авіації України «за станом» з використанням наземних засобів технічного контролю та систем бортових вимірювань. *Зб. наук. пр. Державного*

НДІ випробувань і сертифікації озброєння та військової техніки. 2021. Вип. № 3(9). С. 73–78.

[6] Шейн І. В., Аркушенко П. Л., Андрушко М. В., Кузьміч О. Є., Сокоринська Н. В. Вивчення можливостей вдосконалення та обґрунтування варіанту структури системи контролю безпеки польотів державної авіації. *Зб. наук. пр. Державного НДІ випробувань і сертифікації озброєння та військової техніки.* 2022. Вип. № 2(12). С. 152–159.

[7] Мішарін І. В. Обговорення питань створення системи контролю польотів державної авіації України. Створення та модернізація озброєння і військової техніки в сучасних умовах: XIX, Чернігів, 5–6 вересня 2019 р. : Тези доповіді. Чернігів: ДНДІ ВС ОВТ, 2019. С. 21–22.

[8] Савченко А. С., Матросов А. В., Кравченко М. О. Модернізація засобів зчитування і обробки польотної інформації //*Проблеми інформати зації та управління.* 2024. № 3(79). С.41–49.

[9] Райчев І. Е., Федченко С. В., Харченко О. Г., Савченко А. С. Оцінювання якості програмного забезпечення фільтрації цифрового сигналу в реальному часі для систем критичного призна чення. *Наукоємні технології.* 2021. № 1(49). С. 23–32.

[10] ДСТУ 3275–95. Системи автоматизованого оброблення польотної інформації наземні. Загальні вимоги. [Чинний від 1996–07–01]. К.: Держстандарт України, 1996. 7 с.

[11] Програма перевірки відповідності програмного забезпечення контролю польоту вимогам ДСТУ 3275–95 і «Порядку збирання та практичного використання інформації бортових систем реєстрації на підприємствах цивільної авіації України». К.: Держ. Департамент авіац. транс. України, 1997. 11с.

[12] Райчев І.Е. Проблеми сертифікації програмного забезпечення автоматизованих систем контролю. *Вісник НАУ*. 2004. № 1. С. 23–28.

[13] Райчев І. Е., Харченко О. Г. Концепція побудови сертифікаційної моделі якості програмних систем. Проблеми програмування. 2006. № 2–3. С. 275–281.

[14] Райчев І. Е., Харченко О. Г., Василенко В. А. Визначення вимог до програмних систем критичного призначення з використанням засобів доменного аналізу. *Моделювання та інформаційні технології: Зб. наук. пр.* К.: Інститут проблем моделювання в енергетиці. 2019. Вип. 87. С. 41–48.

[15] Ontology Description Capture Method. [Online]. Available: http://www.idef.com/idef5-ontology-description-capture-method/ (access data 26.02.2025).

[16] IEC 61508. Functional Safety of electrical / electronic / progammable electronic safety-related systems. International Electrotechnical Commission. Geneva. 1998. URL: http://www.iec.ch (access data 26.02.2025)

[17] Sommerville Ian. Software Engineering. Pearson India Education Services, 10th Edition. 2017. 808 p.

[18] Карпов І. А., Буров Є. В. Онтології у процесі прийняття рішень. *Вісник Хмельницького національного університету.* 2023. Том 1, № 2(319), С. 149–153.

[19] Іохов О. Ю. Застосування онтології задачі вибору для опису процесів взаємодії суб'єктів управління. 2021. URL: http://repository.kpi.kharkov.ua/handle/KhPI-Press/52520. (дата звернення 23.02.2025).

[20] Буров Є.В. Ефективність застосування онтологічних моделей для побудови програмних систем. *Математичні машини і системи*. № 1. 2013. С. 44–55.

[21] Райчев І. Е. Синтез автоматних моделей контролю. *Вісник НАУ*. 2002. № 2. С. 43–52.

[22] Райчев І. Е. Технологія визначення показників вірогідності одновимірних та багатовимірних об'єктів контролю за інформацією параметричних бортових реєстраторів. *Вісник НАУ*. 2002. № 1. С. 17–24.

[23] Райчев І. Е. та ін. Автоматизація визначення показників вірогідності об’єктів контролю при наявності разової команди. *Вісник НАУ*. 2002. № 3. С. 73–78.

[24] Raichev I. at oth. The features of testing critical program systems while their certification. *Journal of Automation and Information Sciences.* Begel House Inc. 2011. Vol. 43, Issue 4. Pp. 35–44.

[25] Парадигма програмування. URL: https://uk.wikipedia.org/ wiki/ (дата звернення 23.02.2025)

[26] Автоматне програмування. URL: https://uk.wikipedia.org/wiki/ (дата звернення 23.02.2025)

Райчев І. Е., Денисенко Н. Г.

МЕТОДИ ТА ЗАСОБИ ЗАСТОСУВАННЯ ОНТОЛОГІЧНИХ МОДЕЛЕЙ В ЖИТТЄВОМУ ЦИКЛІ ПРОГРАМНИХ СИСТЕМ КОНТРОЛЮ ПОЛЬОТІВ

Стаття присвячена розробленню методів і засобів використання онтології в процесах життєвого циклу програмних систем контролю польотів, які є системами критичного призначення. Для множини класів систем критичного призначення актуальною є задача формування методів виконання перевірки цих систем на відповідність висуненим вимогам в межах предметної області застосування та аналіз ризиків і небезпек в їх життєвому циклі. Оцінювання відповідності вимогам, як правило, виконується шляхом оцінювання на етапі тестування досягнутого загального рівня якості компонентів програмного забезпечення контролю польотів. З метою всебічного аналізу відповідності атрибутів системи висуненим вимогам необхідно дослідити і

структурувати значну кількість об'єктів та функцій. Для вирішення задачі систематизації в межах домену застосування запропоновано побудувати формальну концептуальну модель системи, на базі якої технологічно сформувати множину об'єктів та базових функцій і методи їх реалізації.

Проблема підвищення безпеки функціонування об'єктів, контрольованих за допомогою автоматизованих систем контролю польотів, є повсякчас актуальною. Одним із шляхів досягнення цієї мети є забезпечення якісного й надійного функціонування програмних систем контролю польотів, що особливо важливо під час експлуатації підсистеми допускового контролю, яка має забезпечувати реалізацію алгоритмів контролю, встановлених для кожного типу літального апарату. У випадку невірної оцінки пілотування і роботи агрегатів та обладнання, існують ризики дозволити подальші польоти непідготовленому екіпажу чи борту літака, який має дефекти устаткування. Означені фактори можуть спричинити авіаційний інцидент, який може безпосередньо вплинути на безпеку життю людей, а також цілісність літального апарату.

Шляхом вирішення проблеми є ефективне застосування онтологічної моделі компонентів програмних систем контролю польотів на протязі життєвого циклу цих систем. Створену онтологічну модель використаємо для побудови математичного апарату подання функцій (задач) онтології у виді автоматних моделей, які застосуємо для вирішення та обчислення задач одного типу в процесах життєвого циклу: під час створення концепції, аналізу вимог, програмування і вичерпного тестування множини програмних компонентів, які реалізують задачі контролю польотів.

У статті проведено аналіз останніх досліджень і публікацій, де показана ефективність застосування онтологій для реалізації процесу прийняття рішень в контексті проблемних ситуацій неоднозначності. Для систем ПЗ контролю польотів можна ефективно використати апарат онтологічних досліджень, бо основною задачею ПЗ є прийняття рішення про надходження об'єкта контролю під час польоту в стані «в межах допуску» (штатний режим), або «за межами допуску».

Проведені дослідження включають у себе: 1) виконання доменного аналізу області застосування; 2) створення онтології програмного забезпечення контролю польотів, яка систематизує і структурує знання щодо понять і об'єктів предметної області; 3) формування функцій інтерпретації онтології, а також специфікацію критеріїв та обмежень для задач прийняття рішень в системах критичного призначення. Подальші дослідження проведені з метою застосування теорії автоматів для програмування алгоритмів контролю та побудови тестових наборів даних для виконання тестування реалізованих програмних компонентів. На основі цих досліджень був побудований метод програмування автоматних моделей контролю з прикладом типового конструювання програм на основі шаблонів програмування.

У висновках зазначено, що побудована онтологічна модель та її функції інтерпретації можуть бути застосовані у життєвому циклі різних класів програмних систем контролю, оскільки будь-які інформаційні системи контролю характеризуються наявністю об'єкта контролю, що контролюється сукупністю параметрів, які вимірюються за допомогою вимірювальних приладів.

Подальші дослідження пропонується проводити у напрямку автоматизації створення програмних компонентів обробки параметричної польотної інформації, для чого необхідно задіяти інтелектуальні можливості штучного інтелекту.

Ключові слова: *критичні програмні системи, онтологія, програмне забезпечення, контроль польотів, концепти, функції інтерпретації, задача прийняття рішення, критерії та обмеження, скінченні автомати, автоматні моделі, вимоги до якості програм, вимоги до безпеки.*

Raichev I., Denysenko N.

METHODS AND MEANS OF APPLYING ONTOLOGICAL MODELS IN THE PROGRAM SYSTEMS LIFECYCLE OF FLIGHT CONTROL

The article is devoted to the development of methods and tools for using ontology in the life cycle processes of flight control program systems, which are mission-critical systems. For a variety of mission-critical system classes, the task of developing methods for verifying these systems for compliance with the specified requirements within the subject domain and analyzing risks and hazards throughout their life cycle is relevant. Compliance assessment is typically performed by evaluating the overall quality level of flight control software components during the testing phase. To comprehensively analyze the compliance of system attributes with the specified requirements, it is necessary to examine and structure a significant number of objects and functions. To address the problem of systematization within the subject domain, it is proposed to build a formal conceptual model of the system, based on which a set of objects, basic functions, and methods for their implementation can be technologically formed.

The problem of enhancing the safety of objects controlled by automated flight control systems remains constantly relevant. One way to achieve this goal is to ensure the high-quality and reliable operation of flight control program systems. This is particularly important during the operation of the admission control subsystem, which must implement control algorithms established for each type of aircraft. In case of incorrect assessment of piloting performance and equipment operation, there is a risk of allowing further flights for an unprepared crew or an aircraft with defective equipment. These factors may lead to an aviation incident, directly affecting human safety and the integrity of the aircraft.

A solution to this problem is the effective application of an ontological model of flight control program system components throughout the life cycle of these systems. The developed ontological model will be used to construct a mathematical framework for representing ontology functions (tasks) in the form of automaton models. These models will be applied to solving and computing tasks of a similar type during life cycle processes, including concept creation, requirements analysis, programming, and exhaustive testing of multiple program components that implement flight control tasks.

The article analyzes recent research and publications demonstrating the effectiveness of using ontologies for decision-making processes in the context of ambiguous problem situations. For flight control SW, the apparatus of ontological research can be effectively utilized, as the primary task of the SW is to determine whether the controlled object remains "within tolerance" (normal mode) or "beyond tolerance."

The conducted research includes: 1) performing a domain analysis of the application area; 2) developing an ontology for flight control software, which systematizes and structures knowledge regarding concepts and objects of the subject domain; 3) defining ontology interpretation functions, as well as specifying criteria and constraints for decision-making tasks in mission-critical systems. Further research has been carried out to apply automata theory for programming control algorithms and constructing test data sets for testing the implemented software components. Based on this research, a method for programming automaton-based control models was developed, along with an example of typical program construction using programming templates.

The conclusions state that the developed ontological model and its interpretation functions can be applied throughout the lifecycle of various classes of control program systems, as any control information system is characterized by the presence of a controlled object monitored by a set of parameters measured using measuring instruments.

Further research is proposed in the direction of automating the creation of program components for processing parametric flight information, which requires leveraging the intelligent capabilities of artificial intelligence.

Keywords: *critical program systems, ontology, software, flight control, concepts, interpretation functions, decision-making task, criteria and constraints, finite automata, automated models, program quality requirements, safety requirements.*

Стаття надійшла до редакції 01.03.2025 р.
Прийнято до друку 16.04.2025 р.