РОЗДІЛ ІІ. КОМП'ЮТЕРНІ НАУКИ

UDC 004.9.629.7.05
DOI https://doi.org/10.26661/2413-6549-2021-1-12

FUNCTIONAL METHODS OF DEVELOPING INTEGRATED MODULAR AVIONICS SYSTEMS

Kovalenko Yu. B.
Ph.D. in Pedagogy,
Associate Professor at the Department of Information Technology Security,
Doctoral Student in the study program "Computer Science and Information Technology"
National Aviation University
Liubomyra Huzara avenue, 1, Kyiv, Ukraine, 03058
orcid.org/0000-0002-6714-4258
yleejulee22@gmail.com

Kozlyuk I. O.
Doctor of Technical Sciences,
Professor at the Department of Telecommunication Systems
National Aviation University
Liubomyra Huzara avenue, 1, Kyiv, Ukraine, 03058
orcid.org/0000-0001-8239-8937
avia_ira@ukr.net

Key words: *information systems, decision-making support, project in the aviation industry, automated design system, technological process, integrated modular avionics.*

Development of modern avionics systems To rob the design of such systems is unfeasible without victorious automation. Today, the area of such instruments is represented by patented instruments, which are broken by such great authors of lithuanians, such as Boeing and Airbus, as well as by low criticality, but sometimes they are often seen in All the tools are based on the architectural models of the broken system. Avionics systems today are a complex interaction of software and hardware, so the methods and approaches developed in the field of design and analysis of avionics and software systems should enrich each other. At the top of the statistics, the movable materials available for describing the architectures of the avionics systems are displayed, which is shown as the program for the best passing through the texts of the meaning and prompted the concepts, as it is good to go for the presentation. Then the statty presents a set of tools for designing custom avionics systems. A set of tools will provide you with a platform for designing and analyzing architectural models, as well as special solutions for singing avionics systems. Winning the structure, editing and manipulating models in both text and graphical formats. It is important that the architectural models themselves, how to describe the components of the system and interconnection between them, form the basis for the formation of new technologies and tools for automating the design. Smell allow you to describe the specific aspects of architecture in a single form-small-scale model, as you can use different tools for revising the internal narrow-mindedness of the architecture, and the configuration of the automation of the system. The foldability of modern emergency systems and a lot of them to the extent necessary to produce up to the need for local resources. When IMA-systems are opened, the retailers are stuck with low

levels of problems, and the stinks have not been stuck in the past. For the resolution of these problems, additional help comes in the form of automation and the computer-turn to adjust the box. The development of a straight line in the first place is linked to the vicarious models of the new models, including the architectural models of the software and hardware complexes.

ФУНКЦІОНАЛЬНІ МЕТОДИ РОЗРОБКИ ІНТЕГРОВАНИХ МОДУЛЬНИХ СИСТЕМ АВІОНІКИ

Коваленко Ю. Б.
кандидат педагогічних наук,
доцент кафедри безпеки інформаційних технологій,
докторант напряму 'Комп'ютерні науки та інформаційні технології'
Національний авіаційний університет
пр. Любомира Гузара, 1, Київ, Україна
orcid.org/0000-0002-6714-4258
yleejulee22@gmail.com

Козлюк І. О.
доктор технічних наук,
професор кафедри телекомунікаційних систем
Національний авіаційний університет
пр. Любомира Гузара, 1, Київ, Україна
orcid.org/0000-0001-8239-8937
avia_ira@ukr.net

Ключові слова: *інформаційні системи, підтримка прийняття рішень, проекти в авіаційній галузі, автоматизована система проектування, технологічний процес, інтегрована модульна авіоніка*

Розвиток сучасних систем авіоніки робить проектування таких систем неможливим без використання засобів автоматизації. У даний час область таких інструментів представлена запатентованими інструментами, розробленими такими великими виробниками літаків, як Boeing та Airbus, а також низкою відкритих або частково відкритих міжнародних проектів, що відрізняються за термінами дії, наявністю вихідного коду та документації. Есі інструменти базуються на архітектурних моделях розробленої системи. У цій статті розглядаються мови, доступні для опису архітектурних моделей систем авіоніки, та показано, яка мова програмування є найбільш підходящою через її текстові позначення та вбудовані концепції, які добре підходять для представлення більшості елементів вбудованих систем. Потім у статті представлено набір інструментів для проектування сучасних систем авіоніки. Набір інструментів забезпечує як загальну платформу для проектування та аналізу архітектурних моделей, так і спеціалізоване рішення для певної галузі систем авіоніки. Він підтримує створення, редагування та маніпулювання моделями як у текстовому, так і в графічному форматах. Зауважімо, що саме архітектурні моделі, що описують компоненти системи і взаємозв'язок між ними, стають основою для формування нових технологій і інструментів для автоматизації проектування. Вони дозволяють описувати різні аспекти архітектури в єдиній формалізованої моделі, яку можна обробляти різними інструментами для перевірки внутрішньої узгодженості архітектури, відповідності різним вимогам системи, автоматизації проектних рішень, генерації даних і файлів конфігурації, вихідний код і т.д. Складність сучасних авіаційних систем і високі вимоги до їх надійності призводять до необхідності використання загальних ресурсів. Під час створення IMA-систем розробники

стикаються з низкою завдань і проблем, з якими вони раніше не стикалися. Для вирішення цих проблем на допомогу приходять різні засоби автоматизації і комп'ютерна підтримка розробки. Розвиток цього напрямку в першу чергу пов'язано з використанням різних моделей, в тому числі архітектурних моделей програмно-апаратних комплексів.

Introduction. The development of modern avionics systems and other safety-critical control systems requires advanced methodological and instrumental support. There are appropriate tools available, but the development of such high-tech domestic industries as aircraft construction cannot rely on them alone for at least two reasons. First, such tools are quite expensive; secondly, and probably more importantly, they are 'closed' for development and adaptation by domestic researchers and engineers, which leads to an even greater backlog of available technologies in this area.

Tools for the design, development, verification and validation of avionics-type systems traditionally support the model-based approach to model development (Model Driven Engineering – MDE, and Model Driven System Engineering – MDSE), as modeling methods in their various forms: full-scale, semi-natural, mathematical – are always utilized in aircraft construction and related industries [1]. In the last 20 to 30 years, a new type of modeling has appeared in the field of software development, related to research on formal program specifications and the use of so-called formal methods for analysis – in particular, for verification of software systems. Avionics systems today are a complex interaction of software and hardware, so the methods and approaches developed in the field of design and analysis of avionics and software systems should enrich each other. For this reason, the use of formal methods of verification of complex software and hardware systems, such as operating systems and microprocessors, allowed us to quickly master the development of design and integration of avionics systems, as many problems in this new area can be solved based on modeling technologies and verification [2].

This article focuses on the development of methods for modeling, synthesis and verification of complex aircraft systems, but the scope of potential application of these technologies is much wider.

Integrated modular avionics. Currently, the main approach to the design and development of on-board systems of civil aircraft is the approach of integrated modular avionics. According to this approach, specialized controllers are replaced by general-purpose processor modules, which provide independent operation of different aviation systems. The wires of each aviation subsystem are replaced with virtual connections within a switched network infrastructure based on technologies such as AFDX (Avionics Full DupleX Switched Ethernet) [3; 4; 5] and CAN (Controller Area Network) [6; 7]. This reduces unreasonable duplication of hardware, which leads to unacceptable levels of power consumption and complexity of the on-board equipment system. But, on the other hand, this approach greatly complicates the process of software and hardware development, posing new challenges in the design and integration of software and hardware.

With the introduction of the IMA approach in the complex of on-board equipment of the aircraft, there is a new subsystem that provides a hardware platform for the software of other on-board systems. This subsystem is called the IMA platform and codenamed ATA-42. The team responsible for designing, configuring and verifying the IMA platform is usually called the System Integration Group, as its task is not only to develop a stand-alone subsystem, but also to coordinate the needs of all platform users and ultimately integrate the entire software and hardware components using the IMA platform.

The tasks of the System Integration Group also include:

– clarification/coordination of discrepancies between requirements and needs with software and hardware developers;

– proecting the IMA platform based on the needs of functional applications in hardware resources, including:

1) distribution of functional applications from computing modules (Core Processing Module – CPM) taking into account the needs of applications (amount of CPU time, distribution of CPU time between strictly periodic applications, RAM/ROM memory, network interface bandwidth, etc.);

2) determining the composition of network components (network topology), taking into account the requirements of reliability, delivery time of messages from sender to recipient, etc.

– verification of the developed on-board equipment complex (OEC) for compliance with the requirements set forth in the design documentation for the aircraft, OEC and its individual components;

– preparation of configuration tables for IMA platform components.

To solve these problems requires an accurate understanding of all the details of the developed complex at both high and low levels of detail, as well as the greatest care in the analysis of the consequences in case of changes. Due to the size of the OEC and the number of essential parts of modern aircraft, it is impossible for one person to have complete knowledge of the full systems. In such conditions, the use of traditional development methods by specialists, based

on a careful description of all requirements, architectural solutions, etc. in text documents, becomes excessively time-consuming and error-prone. The ability to utilize software automation to solve these problems encounters problems of heterogeneity and unstructured information. A natural step to overcome this problem is the formalization of information, translating it into a unified machine-readable form, which allows automation of its processing.

In the context of designing complex software and hardware systems such as the IMA platform, the main core is the architecture of the complex, around which the requirements for the system as a whole are designed, including its individual components, design trade-offs, analysis and verification, etc. Therefore, it is not surprising that it is the architectural models that describe the components of the system and the relationship between them become the basis for the formation of new technologies and tools for design automation. They allow different aspects of the architecture to be described in a single formalized model, which can be processed by different tools to check the internal consistency of the architecture, meet the system's various requirements, automate design decisions, generate configuration data and files, source code, etc. Model analysis tools can be applied at different levels of abstraction, including at the earliest stages of the project in the presence of only partial and evaluative information. Among experts, this practice is called 'Early Validation', and associated sets of relevant tools (Early Validation Tools) [8].

The places for application of such tools in the process of designing and developing the IMA platform are shown in Fig. 1.

The use of architectural models in this area allows resolution of the following problems:

1. Checking restrictions/requirements for the components of the developed complex:

– Checking the adequacy of hardware resources; for example, that the needs of all functional applications in CPU time and memory meet the hardware characteristics of the computing module on which these applications will run.

– Checking the temporal characteristics of the interaction of functional applications or computing modules; for example, that the delivery time of a message from one functional application to another does not exceed the specified requirements.

– Checking the possibility of allocating hardware resources in accordance with certain restrictions; for example, the ability to allocate CPU time for a set of strictly periodic tasks, taking into account that each task must be run at certain times according to a given period.

– Safety and failure analysis of individual components of the OEC (safety analysis).

2. Automation of distribution of hardware resources between functional applications, taking into account defined restrictions; for example, distribution of functional applications on computer modules taking into account sufficiency of bandwidth of network interfaces and possibility of scheduled periodic tasks.

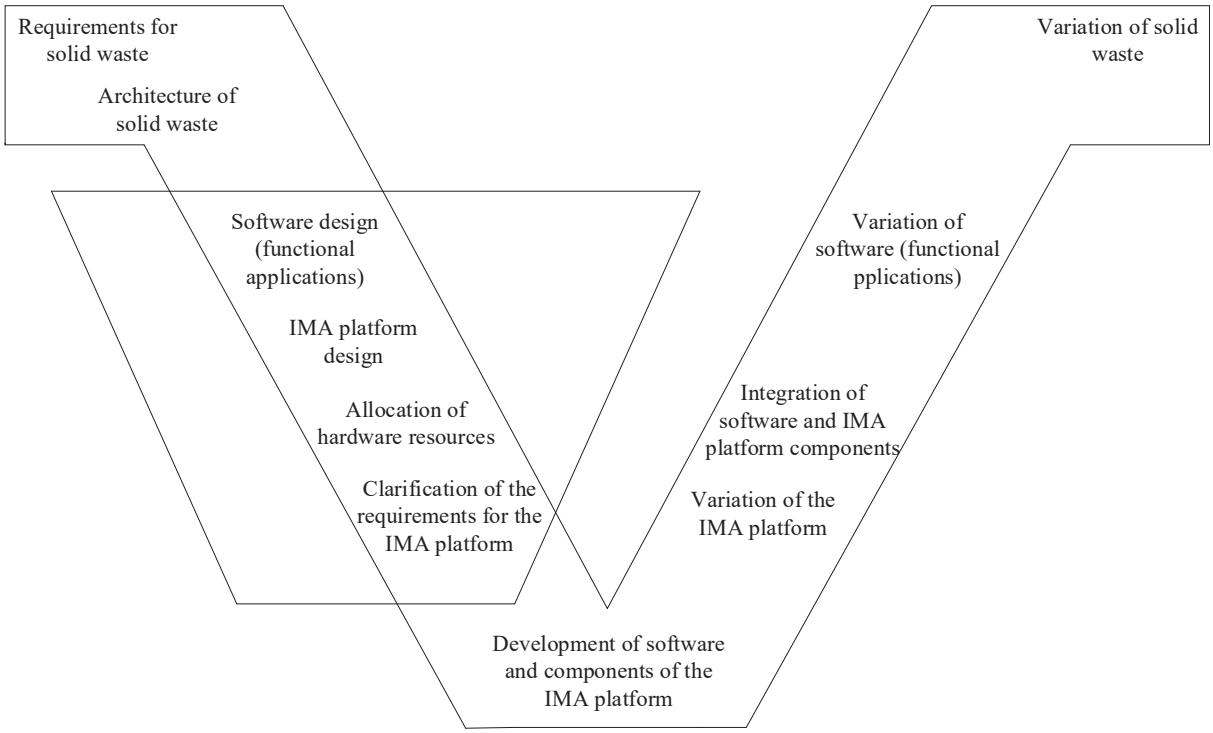

Fig. 1. Validation during design and development IMA platforms

3. Generation of elements of the BWC platform: configuration data / files, source codes of individual components of the platform, etc.

Description languages of architectural models. During the research in the field of design of software and hardware systems on the basis of models, several approaches to the description of architectural models were formed (table 1) The most widespread approaches are based on AADL [9], EAST-ADL [10] and UML [11]. The EAST-ADL language is not considered in this paper because its scope is limited to automotive systems based on AUTOSAR architectural solutions. AADL inherited the main features from the Meta-H language, developed to describe on-board avionics systems in the late 1990s, and is now the most common language for describing architectural models of software and hardware systems in various application areas. UML is most often used to describe software and hardware systems in the form of one of its profiles, the most popular of which are SysML [12] and MARTE [13]. Below are the main features of these languages [14].

Based on the above, it can be concluded that both UML (in the form of SysML and MARTE profiles) and AADL provide approximately the same capabilities to describe the software and hardware model of the OEC. At the same time, AADL has a number of advantages:

– In addition to graphical notation, AADL has a text representation that will allow a specialist to create and edit models, as well as analyze the semantics of existing models without specialized editors, while 'reading' UML-based models without special chart editors can be an intractable task;

– AADL limits the developer to a specific set of declaration types (model element types) that have specific semantics that the developer can use to describe the firmware model, allowing you to reuse

Table 1

Languages for describing architectural models

UML	AADL
Notations	
Provides a set of charts to represent the structure of the software; in this case, individual diagrams that describe certain components of the software and hardware complex that cannot be fully related to each other, i.e. combining models developed by different groups of developers is extremely difficult. Developed more in the tradition of programming languages than descriptions of diagrams; it operates with declarations of types and implementations of model components that can be reused in declarations of other components.	developed more in the tradition of programming languages than descriptions of diagrams; it operates with declarations of types and implementations of model components that can be reused in declarations of other components.
Extending	
Can be extended by using the following mechanisms: stereotypes, which allow to expand the UML dictionary to create new modeling elements; tags of identifiers and values (tagged values); redefinition of model elements with additional constraints. These mechanisms are usually used by one or another profile, which is a dialect of model description (for example, SysML and MARTE).	Can be extended by defining: user-defined property sets that can add new property types and definitions or extend existing types and properties; annex-specifications, which allow to describe additional characteristics of model elements in arbitrary syntax and with arbitrary semantics, which are processed by specialized tools.
Aspects of modeling	
used mainly to describe the structure of the software; it is based on three aspects: data, interaction and state; data is described by class diagrams, interaction is described by connection diagrams or sequence diagrams, states are described by state diagrams. The most used SysML and MARTE profiles extend UML as follows: SysML adds two types of charts – the requirements chart and the parametric chart; the requirements diagram is used to describe the requirements and link the requirements to the elements of the model; parametric diagram is used to describe the relationships of software model components with hardware model components. MARTE expands UML by introducing the following stereotypes: software model, hardware model, the relation between software and hardware models.	used to describe the 'execution architecture'. 'Execution architecture' is implicitly divided into two parts: a set of software components and interaction between them, a set of hardware components and interaction between them; also describes the relationship between software components and hardware components.

existing models developed by independent teams at no additional cost. At the same time, UML, due to its versatility, does not impose strict restrictions on the types and semantics of the elements used, which complicates the understanding of models developed by third-party experts.

MASIW – a system integrator workstation. Given the above features, the AADL language was chosen as a formalism to describe architectural models in research in the field of automation of software and hardware systems.

The research pursues a dual goal, consisting of a research component – the development of methods for modeling and verification of complex software and hardware systems, and an engineering component – the development of working tools for designers and integrators of avionics systems.

The basic principles on which research and tools are built are as follows:

– openness – as a necessary condition for cooperation with the international research community;

– reliance on international standards;

– a combination of mathematical rigor in the choice of proposed solutions and ensuring the availability of these solutions for engineers;

– focus on support and integration of various processes of the life cycle of systems: definition and analysis of requirements, design, integration and verification of software and software and hardware systems.

Currently developed MASIW tools allow to solve such tasks.

1. Creating, editing and managing models in AADL:

1) creating / editing models using a text or graphic editor;

2) support for team development with the ability to track and make changes to individual elements of the model;

3) support for the re-use of third-party AADL models.

2. Analysis of models:

1) analysis of the structure of the software and hardware complex – the sufficiency of hardware resources, consistency of interfaces, etc.;

2) analysis of data transmission characteristics in the AFDX network – time of delivery of messages from sender to recipient, depth of queues of transmitting ports, etc.;

3) simulation of a model of software and hardware with the generation of user-defined reports on the results of the simulator.

3. Synthesis of models:

1) the distribution of functional applications from computing modules, taking into account the resource constraints of the hardware platform and taking into account additional constraints on the reliability and security of software and hardware;

2) generation of CPU computing time allocation between functional applications (application launch schedule cyclogram for ARINC-653 compatible real-time operating systems).

4. Generation of source code / configuration data:

1) development of specialized code / configuration data generation tools, based on the provided software interface (API);

2) generation of configuration files for VxWorks653 RV and AFDX network end devices.

Model creation, editing and management, as well as code and configuration data generation are implemented using common Eclipse environment extensions, such as Eclipse Modeling Framework, Graphical Editing Framework, Eclipse Team Providing, SVN Team Provider, GIT Team Provider. When implementing these capabilities, we mainly had to solve engineering problems, so in the following sections we will focus in more detail on the implementation of support for analysis and synthesis of models, where the main research tasks were concentrated.

Analysis of models. When it comes to the analysis of models, it means the derivation of new properties of the model as a result of considerations about its already known properties. For example, the result of the analysis may be an estimate of the maximum time between sending a message and its delivery based on an analysis of the path of the message and the characteristics of the components encountered in this path. The most important type of model analysis is its verification, ie verification of the model's compliance with the requirements for it. Other types of analysis are usually used as an intermediate step in the verification process.

Requirements for OEC architecture arise from a variety of sources.

– These may be design requirements for the aircraft and the OEC architecture – these requirements in the process of analysis are clarified and decomposed into requirements for individual components of the system.

– Project often regulates the requirements for the design and organization of architectural models, which are described in the so-called model design standard.

– Another source of requirements is the restriction on the area of permissible use or on the permissible configurations of the simulated components (usage domain rules).

– The author of a library model component may impose requirements on the consistent use of this component.

– There are also requirements imposed by model analysis tools or tools that are necessary to be able to perform the relevant analysis.

Since when modeling the system there is a need to detect errors as early as possible, the task is to ana-

lyze the model, which has unspecified components or components with a still unknown structure. Sometimes in such cases for some kind of analysis enough assumptions about the raw components. For example, the system has a process A with an unknown implementation. However, it is assumed or known that on average every 100 ms it generates a data packet with an average size of 100 bytes, intended for process B. In this case, the components that provide network interaction are described in detail in the model. Then such an incomplete model can be analyzed in terms of network interaction, process delays, buffer occupancy of network components, and so on.

Types of model analysis. Types of analysis differ in the method of its implementation (static or dynamic) and aspects of the object under study (the rough division is aspects of the structure or architecture of the system and aspects of behavior, functioning of the system).

The dynamic analysis implies that some patterns are clearly set, according to which there is a change in the model (the internal state of the components, the relationships between them) and interaction with the environment. During such analysis, actions are performed according to the given regularities, obtaining new states of components, new connections, ensuring interaction with the environment. Further (depending on the checked requirements) there is an analysis of the received condition or sequence of states and, for example, an estimation of their correctness.

The static analysis uses a mathematical description of the components of the model, which is compared with the description of the requirements for them. In the course of the analysis the comparison of requirements, calculation of characteristics of components on which further conclusions about correctness or incorrectness of the analyzed model are made is carried out.

The analysis of model behavior considers the characteristics that arise only when considering how components behave over time, how they interact with the world around them, what events and how they react, what events and data they generate, how they change their internal state. The structure analysis considers the characteristics of how the components are connected, what properties these connections have, what are the capabilities and directions of data transmission and events, which components have access to certain resources and so on.

Methods and aspects of modeling can be combined in any way, then, in sections 5.3-5.6, all four possible combinations are considered.

Input data for analysis. The input data for the analysis is a model of software and hardware complex, which describes the structure and characteristics / properties of the elements of the complex. The previously considered model description languages (UML and AADL) allow to describe in detail the structure of the developed complex, up to the description of each sensor, button, etc. As practice has shown, such a model is redundant for most types of analysis. Much of the model is ignored by specialized analysis tools, and as a result the tool has to do extra work to sample essential data from the model of the complex. In addition, model description languages give the model developer some freedom in choosing which entities will describe certain components of the developed complex (UML to a greater extent, AADL – to a lesser extent). At the same time, when developing an analysis tool, it is possible to accurately determine the structure of the input data, which does not depend on what specific entities describe the model of the software and hardware complex. Therefore, in the development of the MASIW tool, the concept of so-called representations was proposed. View is a specialized model of the entire software and hardware complex or a certain part of it, which represents a set of essential data in a form convenient for processing, as is the case with representations in database management systems. To create a specialized representation, a set of adapters is used – the rules by which the original model is transformed into a specialized and, if necessary, vice versa. This approach allows you to abstract from how the developer will describe the model of the complex or part of it (what entities will be used and even what model description language will be used). Due to this, the developed analysis tools can be reused in other modeling tools.

Static structural analysis. The structure of the model can be understood as a graph. the nodes of which are the components of the model, and the arcs are the connections between the components. Types of connections may differ in different modeling languages. For example, the relationship between two components may mean that one component is part of another, or that one component is a hardware resource on which another, software component is running.

The structure of the model may contain information about the composition of the components of the model (of which components-parts it consists), the location of the components, the degree of connectivity of the components, etc. Most often, modeling languages allow you to compare the components of the model attributes, ie some scalar values. In this case, they also become part of the structure of the model and can be used in structural analysis. For example, you can analyze a model to see if the values of an attribute with some specified name in all the components in which it is defined belong to some specified set of valid values.

One of the possible attributes of the components may be the component type. Examples of the use of type in structural analysis can be the following tasks:

to find out whether all components of some type contain an attribute with some given name. Or another example: find out if all components of type A are part of any of the components of type B.

In modeling languages that aim to describe the structure of the model (as opposed to behavioral targets), structural analysis of the model is more convenient than behavioral analysis. The reason for this is that when using such modeling languages, the structure of the model may already be present, while the behavior is not yet fully defined or described. As a result, a number of analyzes can be performed in advance, before a more time-consuming operation to determine the behavioral component of the model.

To organize and automate static structural analysis it is necessary to solve the following tasks:

1) how to specify what needs to be analyzed (in particular, what condition of correctness of structure of model should be checked and what language should be chosen for the description of this condition);

2) how to set the context of the analysis (for which part of the model structure the analysis should be performed) – most often, the analysis should be performed on the whole model, although it is possible, for example, that the analysis should be performed only on components that are part of this model only some specified type;

3) how to perform a given analysis for a given part of the model;

4) in what form to present to the developer the result of the analysis.

All these tasks arose during the development of a tool for static verification of the correct structure of models in the MASIW environment. In this environment, AADL is used as the simulation language. The structure of the model in this language is hierarchical with respect to the inclusion of some components in others. The AADL model also contains a hierarchy of types that describe component classes. The MASIW environment instantiates the model by generating instances of all components and combining these components as required by the semantics of the AADL language, taking into account all inherited and predefined attributes. As a result, the analysis tools work with an already prepared instance of the model and they do not need to know about the difficulties of transforming the declarative AADL model into an instance of the architectural model of the software and hardware complex.

According to the context of the analysis, there are usually global conditions of correctness and component conditions of correctness. Global correctness conditions represent limitations on the model as a whole. Component correctness conditions describe restrictions on certain types of components. Component conditions are sometimes referred to as ‘invariant properties of components’.

To be able to automatically analyze the model to meet the requirements, the requirements must be formalized and described in machine-readable form. To do this, you must have a language for writing formalized requirements. Currently, MASIW supports the description and verification of the correctness of the architectural model in REAL (Requirements Enforcement Analysis Language) [15].

REAL was proposed in 2010 at Telecom ParisTech (France) and has been supported by several research laboratories around the world. This language is based on the apparatus of set theory. The author of the language tried to make the language as convenient as possible for engineers engaged in modeling and with basic skills of imperative programming. However, this language proved to be unsuitable for practical use due to its limited functionality, and high-quality documentation on this language did not appear in the open press [16].

As practice has shown, the REAL language has a number of serious shortcomings. First, it does not support all data types and components of the AADL language (in particular, it does not support values with units of measure). Second, the viona does not support component correctness conditions. Third, the language does not have the means to reuse some of the conditions in others (which is especially true if the verification of several conditions requires the same calculations). In addition, not all conditions of correctness are convenient to represent in the imperative form, and the REAL language does not contain means for the non-imperative description of conditions of correctness.

To partially solve these problems, we have made a number of changes to the REAL language. Moreover, we aimed to make static analysis not so much a means of demonstrating the correctness of the model on AADL, as a means of finding errors in the model. To do this, we, without formally changing the language, proposed a way to annotate the text of the statement in REAL in order to document its semantics. This documentation is used by our tool when constructing a verification report to demonstrate which validity conditions were verified, on which components the validity conditions were verified, what the verification status was on the components (which components were found to be in violation of the validity conditions and which were not), the reason violation (if it was specified in the comments-annotations).

Static behavioral analysis. The purpose of static analysis of the behavior of the system model is to obtain mathematical methods for estimating the limit values of various characteristics of the behavior and interaction of system components.

One of the most important behavioral characteristics is the reaction time of the system to external events. The reaction time is affected by both the speed

of event processing and the time of delivery of information between components. At the heart of the IMA architecture is the idea of dividing hardware resources between many aviation functions while ensuring the absence of unintentional influence of one function on another. The first step in this direction was the separation of computing resources. The next step was the virtualization of data buses, which since Airbus-380 is based on AFDX technology [5].

AFDX is built on the basis of ordinary Ethternet, but has been modified to provide the determinacy, stability, security, reliability required to meet the certification requirements. A key element of AFDX in this regard is the concept of a virtual channel (Virtual Link), which is essentially a virtual wire equivalent to a physical wire between the sender and recipient of messages. The virtuality of the wires reduces the weight, power consumption, complexity of the laying and, most importantly, the cost of maintenance and network development, as the laying of the physical cable is replaced by a modification of the configuration tables in the switches.

In fact, the components that generate and receive messages may be outside the AFDX network. These are either specialized control functions contained in sensors and actuators, or programs that run on computing modules. In all cases, these components are connected to the AFDX network through one or more intermediate gateways, which can communicate with several different data protocols.

Analysis of data transmission in the AFDX network

AFDX technology is based on full-bandwidth switched Ethernet, so conflicts and delays on the lines during data transmission do not occur. The total packet delivery time is equal to the sum of the packet transmission times on the lines plus the delays in the switches [16].

Queues are installed on the output ports of the switches. Thus, the delay in switches can be very variable due to the merger of different virtual channels competing for each output port. Therefore, to determine the upper limit of the total packet transmission time, it is necessary to analyze the delays in each output port of the switch.

Another important behavioral feature of the AFDX network is the guarantee that there is enough space in each queue to store all incoming packets. In practice, for each queue in the network, the upper limit of the amount of data in this queue is estimated.

There are several analytical methods for calculating the estimate of the upper limits of packet delivery time and queue sizes: Model Checking, Trajectory, Network Calculus. Everyone has their advantages and disadvantages. The main disadvantage of most methods is the so-called pessimism – that is, obtaining a deliberately higher estimate due to some assumptions or rough calculations. In addition, a significant disadvantage of the Model Checking method is that when the size of the network increases, it very quickly leads to a combinatorial explosion. Only Trajectory and Network Calculus methods are suitable for industrial use.

The Trajectory method

The Trajectory method [17] is based on the analysis of the worst case scenario that can occur with a packet on its trajectory. The occupancy interval for packet f in the output port p is the time interval during which f can be processed in p. The Trajectory method assumes the longest employment interval in each port. For each competing channel, the maximum number of packets that can delay the sending of packet f from port p during the busy interval is estimated.

When calculating the upper limit of packet delivery time for a given virtual channel, it is assumed that the packet f is on the output port p in the queue, which is already the maximum number of packets of all other virtual channels that can delay sending packet f.

Estimation of the upper limits of queue sizes by the Trajectory method is performed as follows. It is necessary for each source port to find the maximum of the queue sizes among the busy intervals of all virtual channels sent through this port. This value will be the upper limit of the queue size of this port.

The Network Calculus method

In the Network Calculus method, the flow of information through a specific network node is a function of the flow. The function of the flow R (t) from time is a function whose value at time t is the total number of bits that entered this node from the moment $t0 = 0$.

Since the nature of the flow function in this node, in general, is influenced by many different factors, its exact definition is quite difficult. To analyze information flows, the Network Calculus method uses so-called arrival curves, which majorize the flow function 'evenly' from any point in time: the incoming flow curve α (t) R (t) is such a non-decreasing function that for any $0 \le s \le t$ the inequality $R(t) - R(s) \le \alpha(t-s)$ is true.

If the information flow pretends to be a periodic sequence of packets of limited length, the function R (t) is a 'step' function. The input curve for a function of this kind is the function $\gamma_{r,b}(t) = rt + b$, where r is the average flow rate, b is the maximum packet size.

In addition to the input curve, which is a 'representative' of the flow function, the Network Calculus method in each network node considers a service curve (service curve), which describes the amount of processed information in the node to a given point in time.

Consider a situation in which a node processes information at a constant speed R (usually this speed corresponds to the bandwidth of the communication line at the output of the node), but before process-

ing introduces some delay limited by time T (usually this time corresponds to the maximum technological delays in the delivery of information within the node from input to output). In this case, the serving curve at this node is equal to $\beta_{R, T}(t) = \max \{0, R (t-T)\}$.

The maximum delay that the information from the stream with the input curve α receives in the node that provides the service curve σ, is estimated from above by the value of the maximum horizontal (ie on the axis 'time') deviation from α to σ. For the case $\alpha = \gamma_{r, b}$ and $\sigma = \beta_{R, T}$ the size of the maximum delay is equal to T + b / R.

The maximum amount of raw information from the stream with the input curve α in the node that provides the service curve σ is estimated from above by the value of the maximum vertical (ie along the axis 'information') deviation from α to σ. For the case $\alpha = \gamma_{r, b}$ and $\sigma = \beta_{R, T}$ the maximum amount of raw information is equal to $\gamma_{r, b}(T) = b + rT$.

The use of coarsening in the form of incoming curves instead of flow functions leads to the fact that as the packet passes through the nodes of the network, the input curve becomes more 'rough'. In particular, for the case where the node input curve for the input stream is $\alpha = \gamma_{r, b}$, and the serving curve is equal to $\sigma = \beta_{R, T}$, the input curve for the output stream (it will be the input curve for the input flow of the next node on the path of this packet) is equal to $\gamma_{r,b+rT}$, ie as the packet of network nodes passes, the second parameter of the function γ increases, so that on subsequent nodes, respectively, estimates of delay time and the amount of raw information increase.

If we compare Network Calculus and Trajectory with each other, we can not say about the clear advantage of one of the methods. Although in many cases Trajectory gives more accurate estimates of the worst time than Network Calculus, there are network configurations where the opposite is true.

Dynamic behavioral analysis. Dynamic analysis of behavior allows to obtain less pessimistic estimates of behavioral characteristics, compared with static analysis. In addition, sometimes the use of dynamic analysis allows the analysis in cases where it is statically impossible to perform or it requires too many resources (too much complexity of the model is difficult and difficult to analyze the mathematical description of components, combinatorial explosion, etc.). However, it should be borne in mind that dynamic analysis is performed in a specific performance, not in the worst case, and requires a specific context of work: input data and impacts [18].

Dynamic behavioral analysis requires that the behavior of the components be specified in the model in some executive way. For the correct use of feasible models, it is important to organize work with model time, with the transfer of information and events within the model.

This problem is well solved using a discrete-effective approach to behavior modeling. In this approach, the work of the component is presented as a set of 'events' – acts of action to change the state of objects and interact with other components and the outside world at certain points in time.

This approach has proven to be the most suitable for modeling the behavior of computer systems, which are complex of onboard equipment IMA systems. On the one hand, it is quite powerful for modeling such systems, on the other hand, this approach is much easier to apply (in contrast to the even more powerful approach of continuous simulation, which faces the solution of nonlinear differential equations and used in modeling physical processes).

To support the discrete-event paradigm of behavior description, the MASIW tool has implemented a library that supports stimulation time and provides processing of events arising from the simulation system, and synchronous and asynchronous data transmission. To ensure these possibilities, a continuations programming approach was used, supported by the Matthias Mann's continuations library.

The AADL language has its own means of describing behavioral semantics for some elements of the model. However, these tools are not enough to describe the behavior of applications, devices and other complex components of the model. There are standardized extensions of AADL – Behavioral Model Annex [19] and BLESS – that allow you to describe the behavior of model components based on finite state machines that work with time and events.

At the moment in the MASIW tool the feasible model of behavior is set in Java language. This allowed us to quickly implement support for dynamic behavioral analysis of AADL models, and at the moment the tool can already be used for such analysis. However it is necessary to put behavior of components in a non-standard way. On the other hand, the use of Java will allow in the future to implement translators from standard behavior task languages without the need to rework the part responsible for the actual dynamic analysis.

As mentioned earlier, it is often advisable to conduct an early analysis of the system on incomplete models. In such models, some components do not yet have a detailed description, and often there are only some assumptions about how they behave in this system. In this case, the analysis can be performed by recording the assumptions in the form of a particular behavior of the components. The MASIW tool supports a special method of parameterizing the model, which simplifies the description of different assumptions about the components for different variants of model analysis.

Dynamic structural analysis. This type of analysis is required for reconfigured systems, ie for sys-

tems whose structure may change during operation. In the general case, there may be a dependence of the model structure on the input influences or the environment of the modeled system, so it is not always possible to statically analyze the implementation of all necessary structural constraints.

In its pure form, dynamic structural analysis involves verifying that all achievable states of the architectural model are structurally correct, ie meet the requirements for the correctness of the structure of the model. To check the properties of the model structure obtained at any time during the execution of the model, you need to get the changed structure and run checks on the new model. This check is similar to a static structural check, the input of which is fed to the model obtained in the dynamics. The difficulty here is that you do not always need to check all the properties in all states, but you need to specify in some way which checks to perform at what time.

Dynamic structural analysis can also check the properties that are set not on one state of the model structure, but on the sequence of such states, although such analysis is rather an analysis of the behavioral aspect of the functionality for reconfiguring the system.

Traditionally, among the properties of objects over time, there are safety properties and survivability properties. The former require that something never happen, while the latter require that something ever happen. An example of a security property is the requirement that immediately after the occurrence of event X, a given component A will have a subcomponent B, and the survivability properties – the requirement that after the occurrence of event X, a given component A will sooner or later have a subcomponent B.

The already considered requirement of structural correctness of all achievable states is the simplest example of a safety property. Dynamic verification of more complex security properties can be implemented on similar principles, given that part of the information used in the static structural analysis of the fixed state of the architectural model should be calculated based on the properties of previous states of the model.

Checking the survivability properties is a significantly different task. The main feature of these properties is that they are violated only at infinity. Therefore, the task of verifying such properties cannot be solved by pure dynamic analysis and requires the development of special tools.

Automatic synthesis of models. The designer of the IMA system has a task to build an architecture that must meet the requirements of different types: the adequacy of hardware resources, fault tolerance, reliability, security of the system as a whole, limiting the maximum allowable time for delivery of messages between components, requirements for timely functions etc.

To a certain extent, the art of experienced specialists, armed in addition with the tools of verification of the constructed architectural model, allows to solve such a problem. However, this approach has limited scalability and high subjectivity. System design automation tools that meet a set of requirements and constraints can make designers work much more efficiently.

In many cases, individual parts of the model can be automatically synthesized based on the information contained in another (‘source’) part of the model, which describes the basic logical relationships between the components and the requirements for the resulting architecture. In this case, the development of the original part of the model is much easier than the development of the corresponding synthesized part. In addition, the source part in any case must be described in the design process. For example, based on the source information about the available set of applications and their hardware requirements, as well as information about the architecture and capabilities of computing modules, it is possible to automatically synthesize the binding of applications to these modules to meet all resource adequacy and scheduling requirements.

The MASIW design environment offers the following work scenario for developing a model of the designed system. The designer develops the necessary source part of the model, then launches an automatic synthesis algorithm, which based on the available information contained in the source part of the model, completes the architecture model with new parts, which can be adjusted manually or regenerated if the original part of the model is updated.

Automatic synthesis of schedules for strictly periodic tasks. When dividing hardware resources between several applications, one of the most important aspects is the timely provision of CPU resources for all tasks in the system. This aspect is usually dealt with by a special operating system task scheduling subsystem, which allocates CPU time to functional applications based on a pre-prepared schedule.

As initial data in the task of construction of the schedule for each of periodic tasks are set:

1) task start period;

2) task execution time on one start-up period.

Classical algorithms for scheduling periodic tasks work only when the start time of the task within the period is allowed to vary at different periods of its execution. However, there is currently a need to compile schedules in which the time between adjacent launches of one periodic task would be fixed and equal to the length of the period. This additional requirement of strict periodicity does not allow the use of classical planning algorithms in the scheduler.

The main difficulty of the algorithm for planning strictly periodic tasks is the search for starting points for all tasks, so that it was possible to build the actual schedule. This search is an NP-complete task.

In addition, we use the strategy of finding starting points implies a search in the first place of such options that provide the longest possible continuous execution of the first ticks after starting each task.

In general, this approach allows you to quickly get a solution to the problem of scheduling for strictly periodic problems [20].

Automatic synthesis of IMA system architecture. As initial data in the problem of synthesis of architecture of IMA the following are set:

– functional applications and logical data flows between them, as well as between applications and sensors / actuators;

– a set of needs for functional applications to hardware resources (memory, computing power, etc.);

– a set of requirements for the maximum time of delivery / processing of messages in logical data streams;

– a set of available hardware components (computing modules, switches, etc.) in conjunction with a description of their capabilities and limitations on the scope of their permissible use (usage domainrules).

You need to automatically build the architecture of the IMA system, which includes:

– composition and communication of hardware components;

– placement of functions on computing modules;

– details of the organization of connections in the AFDX-network;

– work schedule of application and system partitions ARINC-653 compatible operating systems.

The system architecture must meet all safety and performance requirements.

The synthesis task is divided into two major subtasks:

1) placement of applications from computing modules so that it was possible to build a schedule on each module;

2) assignment of virtual channels between computing modules and distribution of switches on virtual channels so as to meet the requirements for message delivery time.

The solution of the first problem is based on the consideration of the set of periods of application launch and on the application of numerical reasoning, which allow to divide the set of periods into such subsets that for each obtained subset there are guaranteed starting points of the corresponding applications [21; 22].

The solution of the second problem is based on the use of genetic algorithms, at each step of the genetic algorithm is built a population consisting of N correct topologies of the AFDX-network. Each topology of the new population is obtained either as a result of a small modification (mutation) of some topology of the previous population, or as a result of crossing some two topologies of the previous population. When crossing, the resulting topology receives the maximum number of common properties (in the sense of connecting components together), which are in both source topologies.

After the next population is constructed, the incoming topologies are ranked in such a way that N topologies that best meet the requirements for message delivery time are selected for further construction. Static methods (Trajectory, Network Calculus) are used to estimate the delivery times obtained in this topology, and the main component of the ranking function looks like this:

$$\Sigma\, e^{T-\tau},$$

where the summation is performed on all channels for which the delivery time limit is set, T is the delivery time for this channel in this topology, τ is the specified maximum delivery time for this channel.

Development prospects. At the moment, the MASIW tool allows to perform only part of the tasks assigned to the system integration group and further plans to expand the functionality of its functionality in many areas.

In the context of static structural analysis of models, the main direction of development is the development of a full-featured language for describing constraints on the structure of an architectural model convenient for a compact description of both global and component constraints. In our opinion, this language should be based on one of the well-known existing programming languages in order to be able to reuse ready-made libraries with a variety of functionality and simplify the task of training engineers. A good contender for the role of such a language is the Python language, which due to the concept of decorators provides an opportunity to form a specialized language based on standard syntax, which means the ability to use the existing interpreter and other tools unchanged for a new language. Other promising areas are the development of libraries of ready-made parts of the code for their reuse in checking the conditions of correctness and the implementation of static structural analysis of reconfigured systems [23; 24].

In the context of static behavioral analysis of models, a promising area of development of supported analysis methods is the analysis of data transmission in the system as a whole, and not only within the AFDX network. The main difficulty here is to take into account the behavior of all components of the gateways located between the sender / recipient of the message and the AFDX network.

In the context of dynamic behavioral analysis of models, the main direction of development is to support standard ways of setting behavior for the

components of the model (Behavioral Model Annex, BLESS). Another very important area of development of this type of analysis is the implementation of the possibility of using the simulator in combination with a stand of semi-natural modeling and in combination with external emulators of hardware platforms. This will save time on developing detailed models for existing system components that are available for use on the stand or in a virtual environment, which reduces the total time and cost of preparation for testing the model.

In the direction of dynamic static analysis of models, only research work has been carried out, so the implementation and conduct of experiments with this method of analysis is another task for the future development of the functionality of the tool.

In the context of automatic synthesis of models promising areas of development are the support of new types of constraints on the synthesized model, research methods of incremental synthesis of architecture and automatic updating of the model when changing the initial requirements taking into account manual modifications of previous synthesized models. the degree of criticality of each function ensured the smooth operation of the entire system, provided the possibility of failure of individual components.

Another area for the development of MASIW tools is the generation of documentation describing the architecture of the BWW system, as well as the generation of project templates and source code of functional applications that would already include typical functions such as message processing whose structure is already described in the architectural model.

Conclusions. The complexity of modern aviation systems and high requirements for their reliability lead to the need to use shared resources (IMA architecture). When creating IMA systems, developers (in particular, system integrators) face a number of tasks and problems that they have not encountered before. To solve these problems come to the aid of various automation tools and computer development support. The development of this area is primarily associated with the use of various models, including architectural models of software and hardware systems. The corresponding group of technologies is called Model Driven System Engineering (MDSE).

The implementation of MDSE technologies requires serious research and well-thought-out engineering solutions. One of the sources of complexity in the development and implementation of MDSE is the need to take into account the needs and preferences of different groups of professionals, as models are used both as input for synthesis and verification, as a design tool and as a means of communication and cooperation. This article is devoted to the methods and tools for solving these problems. The article pays special attention to the issues of integration of methods of formal specification and formal analysis of avionics models with methods of design, implementation and integration of avionics systems, which were developed in this field earlier.

The MASIW tool simplifies the solution of a number of tasks related to the development of aviation systems. It allows you to conveniently and clearly create and edit models of such systems in AADL, as well as analyze such models for compliance with various requirements related to both the structure and behavior of the model (calculate various temporal characteristics, predict the behavior of the simulated system in different situations, including non-standard behavior of components and failures within the system).

In addition, MASIW facilitates architecture design through the implementation of a number of model synthesis algorithms. This allows, in particular, to distribute the tasks on the computing units so that each task was allocated enough CPU time, and to generate an on-board network model and network resource allocation scheme according to the needs of system components.

The MASIW tool is constantly evolving. This development is based on close cooperation with customers, potential users and with the international community of developers of open standards and open tools to support the development, integration and verification of responsible systems based on the use of modeling tools.

BIBLIOGRAPHY

1. Hayley J., Reynolds R., Lokhande K., Kuffner M. and Yenson S. (2012) Human-Systems Integration and Air Traffic, *Control Lincoln laboratory journal,* no. 19(1), pp. 34-49.
2. Parkinson P. and Kinnan L. (2015) Safety-Critical Software Development for Integrated Modular Avionics, *Wind River*, vol. 11, no. 2.
3. Tiedeman H. and Parkinson P. (2019) Experiences of Civil Certification of Multi-Core Processing, *Systems in Commercial and Military Avionics Integration Activities*, vol. 1(2) pp. 419–428. doi: https://doi.org/10.3182/20110828-6- it-1002.01501.
4. Ghannem A., Hamdi M., Kessentini M. and Ammar H. (2017) Search-based requirements traceability recovery: A multi-objective approach, *Proc. IEEE Congress on Evolutionary Computation (CEC)*, pp. 1183–1190. doi: https://ieeexplore.ieee.org/document/7969440.
5. Neretin E. (2019) *J. Phys.: Conf. Ser. 1353 012005.* doi: https://iopscience.iop.org/article/10.1088/1742-6596/1353/1/012005.

6. Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signaling.
7. Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit.
8. Murphy B. and Wakefield A. (2009) *Early verification and validation using model-based design The MathWorks.*
9. SAE International Architecture Analysis & Design Language (AADL).
10. The ATESST Consortium 2010 East-adl 2.0 specification. URL : http://www.atesst.org.
11. ISO/IEC 19505-1:2012 Object Management Group Unified Modeling Language (OMG UML).
12. Object Management Group (OMG) Systems Modeling Language SysML, Version 1.3.
13. Object Management Group (OMG) UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Version 1.1.
14. De Niz D (2007) *Diagrams and Languages for Model-Based Software Engineering of Embedded Systems: UML and AADL, SEI.*
15. Gilles O. and Hugues J. (2010) Expressing and Enforcing User-Defined Constraints of AADL Models, *Engineering of Complex Computer Systems (ICECCS).*
16. URL : https://wiki.sei.cmu.edu/aadl/index.php/Osate_2_Lute.
17. Martin S. and Minet P. (2006) Schedulability analysis of flows scheduled with FIFO: application to the expedited forwarding class, *Parallel and Distributed Processing Symposium*.
18. URL : http://www.matthiasmann.de/content/view/24/26/.
19. AS5506/2 SAE Architecture Analysis and Design Language (AADL) Annex Volume 2.
20. Zelenov S. (2011) Планирование строго периодических задач в режиме реального времени, *Труды ИСП РАН,* Т. 20.
21. Konakhovych H., Kozlyuk I., Kovalenko Y. (2020) Specificity of optimization of performance indicators of technical operation and updating of radio electronic systems of aircraft, *System research and information technologies*, no. 3, pp. 41-54.
22. Kovalenko Y., Konakhovych H., Kozlyuk I. (2020) Specificity of optimization of performance indicators of technical operation and updating of radio electronic systems of aircraft. *International Journal of Engineering Research and Applications (IJERA)*, vol. 10 (09), pp. 48-58.
23. Kozlyuk I., Kovalenko Y. (2020) Functional bases of the software development and operation in avionics. *Problems of Informatization and Management*, no. 63, pp. 49-63.
24. Коваленко Ю.Б., Козлюк І.О. Реалізація програмного комплексу розроблення додатка інтегрованої модульної авіоніки за стандартом ARINC653, *Вісник Запорізького національного університету. Фізико-математичні науки.* 2020. № 2. С. 27–35.

REFERENCES

1. Hayley J., Reynolds R., Lokhande K., Kuffner M. and Yenson S. (2012) Human-Systems Integration and Air Traffic, *Control Lincoln laboratory journal,* no. 19(1), pp. 34-49.
2. Parkinson P. and Kinnan L. (2015) Safety-Critical Software Development for Integrated Modular Avionics, *Wind River*, vol. 11, no. 2.
3. Tiedeman H. and Parkinson P. (2019) Experiences of Civil Certification of Multi-Core Processing, *Systems in Commercial and Military Avionics Integration Activities*, vol. 1(2) pp. 419-428. doi: https:// doi.org/10.3182/20110828-6- it-1002.01501.
4. Ghannem A., Hamdi M., Kessentini M. and Ammar H. (2017) Search-based requirements traceability recovery: A multi-objective approach, *Proc. IEEE Congress on Evolutionary Computation (CEC)*, pp. 1183-1190. doi: https://ieeexplore.ieee.org/document/7969440.
5. Neretin E. (2019) *J. Phys.: Conf. Ser. 1353 012005.* doi: https: //iopscience. iop. org/ article/ 10. 1088/ 1742-6596/1353/1/012005.
6. Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signaling.
7. Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit.
8. Murphy B. and Wakefield A. (2009) *Early verification and validation using model-based design The MathWorks.*
9. SAE International Architecture Analysis & Design Language (AADL).
10. The ATESST Consortium 2010 East-adl 2.0 specification. [Electronic resource] Online: http://www.atesst.org.
11. ISO/IEC 19505-1:2012 Object Management Group Unified Modeling Language (OMG UML).
12. Object Management Group (OMG) Systems Modeling Language SysML, Version 1.3.
13. Object Management Group (OMG) UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Version 1.1.

14. De Niz D (2007) *Diagrams and Languages for Model-Based Software Engineering of Embedded Systems: UML and AADL, SEI.*
15. Gilles O. and Hugues J. (2010) Expressing and Enforcing User-Defined Constraints of AADL Models, *Engineering of Complex Computer Systems (ICECCS).*
16. [Electronic resource] Online: https://wiki.sei.cmu.edu/aadl/index.php/Osate_2_Lute.
17. Martin S. and Minet P. (2006) Schedulability analysis of flows scheduled with FIFO: application to the expedited forwarding class, *Parallel and Distributed Processing Symposium*.
18. [Electronic resource] Online: http://www.matthiasmann.de/content/view/24/26/.
19. AS5506/2 SAE Architecture Analysis and Design Language (AADL) Annex Volume 2.
20. Zelenov S. (2011) Planirovanie strogo periodicheskih zadach v sistemah real'nogo vremeni, *Trudy ISP RAN* vol. 20 (in Russ.).
21. Konakhovych H., Kozlyuk I., Kovalenko Y. (2020) Specificity of optimization of performance indicators of technical operation and updating of radio electronic systems of aircraft, *System research and information technologies*, no. 3, pp. 41-54.
22. Kovalenko Y., Konakhovych H., Kozlyuk I. (2020) Specificity of optimization of performance indicators of technical operation and updating of radio electronic systems of aircraft. *International Journal of Engineering Research and Applications (IJERA)*, vol. 10 (09), pp. 48-58.
23. Kozlyuk I., Kovalenko Y. (2020) Functional bases of the software development and operation in avionics. *Problems of Informatization and Management*, no. 63, pp. 49-63.
24. Kovalenko Y., Kozlyuk I. (2020) Implementation of the integrated modular avionics application development complex according to the ARINC653 standard, *The Bulletin of Zaporizhzhia National University. Physical and mathematical Sciences,* no. 2, pp. 27-35.