METHODS OF DEVELOPING INTEGRATED MODULAR AVIONICS SYSTEMS

Yuliia Kovalenko
Candidate of Pedagogical Sciences, Associate Professor,
National Aviation University, Ukraine
e-mail: yleejulee22@gmail.com, orcid.org/0000-0002-6714-4258

Summary

The development of modern avionics systems makes the design of such systems impossible without the use of automation tools. Currently, the area of such tools is represented by patented tools developed by major aircraft manufacturers such as Boeing and Airbus, as well as a number of open or partially open international projects, differing in terms of validity, availability of source code and documentation. All tools are based on architectural models of the developed system. This article discusses the languages available for describing architectural models of avionics systems and shows which programming language is most appropriate due to its textual notation and embedded concepts that are well suited to represent most of the elements of embedded systems. The article then presents a set of tools for designing modern avionics systems. The toolbox provides both a general platform for designing and analyzing architectural models and a specialized solution for a specific area of avionics systems. It supports creating, editing and manipulating models in both text and graphic formats.

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

DOI https://doi.org/10.23856/4341

1. 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 *(Hayley and other, 2012)*. 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 *(Parkinson and other, 2015)*.

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.

2. 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) *(Tiedeman and other, 2019)* and CAN (Controller Area Network) *(Ghannem and other, 2017)*. This reduces unreasonable duplication of hardware, which leads to unacceptable levels of power consumption and complexity of the on-board equipment system *(Neretin, 2019)*. 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 *(Murphy and other, 2009)*.

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 *(De Niz D, 2007)*.

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:

a. 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.);

b. 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) *(Gilles and other, 2010)*.

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

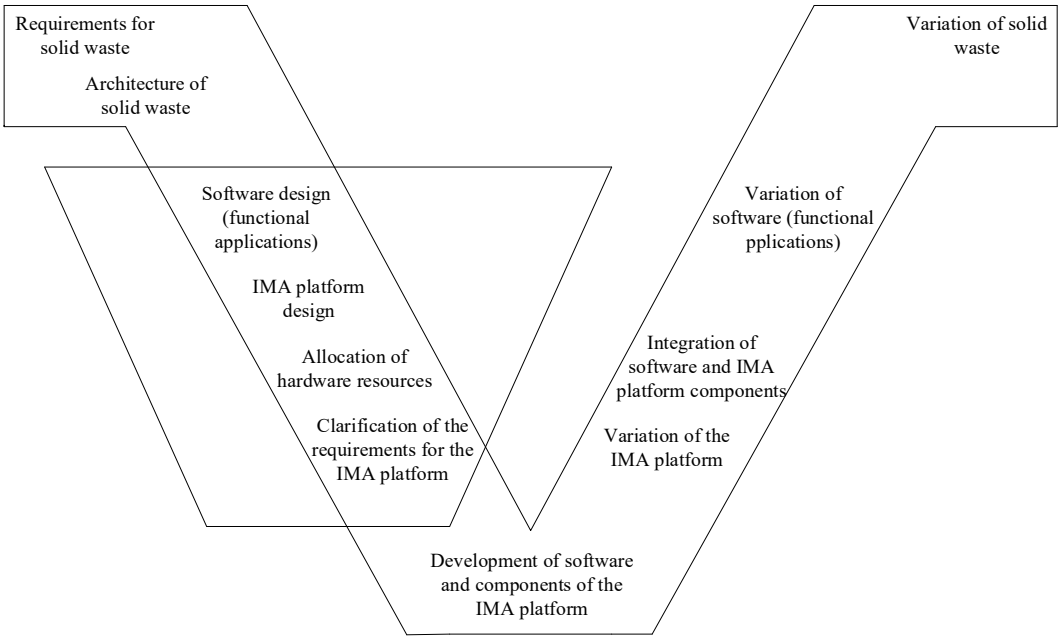

Figure 1. Validation during design and development IMA platforms

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.

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

3. 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).

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.

Table 1 (Continuation)

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.

The most widespread approaches are based on AADL *(Martin and other, 2006)*, EAST-ADL *(Zelenov, 2011)* and UML *(Konakhovych and other, 2020)*. 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 *(Kovalenko and other, 2020)* and MARTE *(Kozlyuk and other, 2020)*. Below are the main features of these languages *(Kovalenko and other, 2020)*.

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 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.

4. 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:

a. creating / editing models using a text or graphic editor;

b. suport for team development with the ability to track and make changes to individual elements of the model;

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

2. Analysis of models:

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

b. 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.;

c. 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:

a. 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;

b. 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:

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

b. 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.

5. 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 analyze 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.

6. 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.

6.1. 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:

- task start period;
- 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.

6.2. 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.

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.

7. Conclusions and suggestions

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.

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.

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.

References

Hayley J., Reynolds R., Lokhande K., Kuffner M. and Yenson S. (2012) Human-Systems Integration and Air Traffic. Control Lincoln laboratory journal, vol. 19(1), pp. 34-49.
Parkinson P. and Kinnan L. (2015) Safety-Critical Software Development for Integrated Modular Avionics. Wind River, vol. 11, no. 2.
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.
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.
Neretin E. (2019) J. Phys.: Conf. Ser. 1353 012005. doi: https://iopscience.iop.org/article/10.1088/1742-6596/1353/1/012005.
Murphy B. and Wakefield A. (2009) Early verification and validation using model-based design The MathWorks.
De Niz D. (2007) Diagrams and Languages for Model-Based Software Engineering of Embedded Systems: UML and AADL, SEI.
Gilles O. and Hugues J. (2010) Expressing and Enforcing User-Defined Constraints of AADL Models. Engineering of Complex Computer Systems (ICECCS).
Martin S. and Minet P. (2006) Schedulability analysis of flows scheduled with FIFO: application to the expedited forwarding class. Parallel and Distributed Processing Symposium.
Zelenov S. (2011) Planirovanie strogo periodicheskih zadach v sistemah real'nogo vremeni [Scheduling of Strictly Periodic Tasks in Real-Time Systems]. Trudy ISP RAN [The Proceedings of ISP RAS].
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.
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.
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.
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.