Methodology and Tools for Development and Verification of formal fUML Models of Requirements and Architecture for Complex Software and Hardware Systems

A.V. Samonov <a.samonov@mail.ru>
G.N. Samonova <g.samonova@mail.ru>
Mozhaiskiy Military Space Academy,
13, Zhdanovskaya St., Saint Petersburg, 197088, Russia

Abstract. The article presents models and algorithms to support end-to-end quality control of complex software and hardware systems through the implementation of the software-controlled process of development and verification of formal models of requirements and architecture of such systems, Firstly, we give the analysis of scientific publications and the normative-methodical base in the field of development and application in practice of the model-based approach is given. We establish that least provided by model, algorithmic and software solutions are issues related to the development of a complete and correct set of requirements, as well as the formalization and verification of technical projects of software and hardware systems. To solve the existing problems, we propose to develop a special unified environment for the development, modeling and testing formal models of requirements and architecture of complex software and hardware systems. These models provide an optimal set of interconnected fUML diagrams presented in ALF notation and verified in the fUML virtual machine and using SMT/SAT solvers.

Keywords: activity diagrams; class diagrams; design and implementation; life cycle of automated systems; model of requirements; model of architecture; software and hardware systems; verification and validation

DOI: 10.15514/ISPRAS-2016-30(5)-8

For citation: Samonov A.V., Samonova G.N. Methodology and Tools for Development and Verification of formal fUML models of Requirements and Architecture for Complex Software and Hardware Systems. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018. pp. 123-146. DOI: 10.15514/ISPRAS-2018-30(5)-8

1. Introduction

Now, when the confrontation in the political, economic and military fields is growing, one of the most important activities of the state is to ensure the safe operation of critical information infrastructure (CII). According to the Federal Law of the Russian Federation [1], CII objects are automated control systems (ACS) for production and technological processes of the critical objects of the Russian Federation and information and telecommunication networks providing them, IT systems and communication networks for solving public administration tasks, ensuring defense capability, security and law enforcement. Disruption of the functioning of CII objects can lead to disastrous consequences in the field of defense capability, economy, health care and security of the nation.

Automation means complexes, which form the basis of the CII objects, are complex software and hardware systems (CSHS); their foundation of reliable and safe functioning is laid in the process of their design, development, and verification. The main factors and conditions for achieving the required quality indicators of CSHS are:

1) implementation of a quality management system defined by modern normative-methodical documents (NMD) in the field of system and software engineering at companies developing CSHS;
2) highly qualified designers, developers, and testers of CSHS;
3) use of modern technologies, methods and tools for design, development, and testing of CSHS.

The most important issues relate to the implementation of the third direction, which is being developed in system and software engineering [2] and model-based methodology [3]. The need to improve the technology and development tools of CSHS is due to distressing statistics on the implementation of IT projects both in Russia and abroad. Thus, according to the research of The Standish Group, the analysis of the results of work on the creation of information systems showed that in the United States (over the past 15 years), only 20% of the projects were completed on time and according to the original budget. At the same time, 30% of the projects failed; 50% faced various problems: the total budget exceeded the initial one by 2 times on average; the terms increased by 1.5 times; less than 75% of the required functionality was implemented [4]. The development process of CSHS consists of three main stages: justification of requirements, design, and implementation, each of which, according to the methodology of the model-based approach, includes a verification procedure of the corresponding artifact. As the analysis showed, issues related to the automation of the processes of generating and verifying computer code created at the implementation stage have been solved quite successfully. At the same time, the stages of requirements formation and system architecture design require the participation of specialists in the field of system engineering and information technology and end users.

As the analysis showed, the main limiting factors in achieving qualitative improvements in solving these tasks are:

- absence of a rigorous mathematical model describing the processes of implementation and application of methods and tools of model-based systems engineering in the main stages of the life cycle of CSHS in a uniform model-language environment;

- objective complexity of the task of creating a formal presentation of system requirements based on their original informal representation;
- availability of a wide range of languages and tools proposed for building models of the analysis, architecture, and implementation of a system in the absence of clear and specific rules and recommendations for their application;
- lack adequate tools for automated construction and execution of test scenarios for the verification of requirements and architecture.

The second section provides a brief overview of scientific and technical publications, in which the described issues are considered and solved. The third and fourth sections of the article present the models and algorithms for building a formal specifications requirements. The fifth section describes the models and algorithms for developing and verification the architecture of CSHS. The sixth section presents the methodology for constructing test scenarios to verify models of requirements, architecture, and implementation of CSHS using the SAT/SMT solvers.

2. Overview of the Current Normative-methodical Base and Scientific Publications in the Field of Development and Verification of CSHS

The exceptional relevance of the problems described above has led to the great attention and efforts taken by international and national organizations, scientific and professional communities, development teams and individual researchers to solve them. In the authors’ opinion, the most important ones are methodical documents and specifications developed under the auspices of the OMG (Object Management Group) organization that cooperates with about 800 research organizations (DISA, INCOSE, NIST, etc.) and industrial companies (AT & T, IBM, Oracle, Microsoft, Cisco Systems, NASA, etc.). In Russia, active research in this area is carried out by such organizations as ISP RAS, the Faculty of Computational Mathematics and Cybernetics of Lomonosov Moscow State University, Saint Petersburg State University, Novosibirsk State Technical University, Military Space Academy named after A. F. Mozhaisky, etc.

Currently, more than 230 methodical documents and specifications have been published on the OMG website. Considering the issues described above, the most important specifications are: MOF (Meta Object Facility), UML (Unified Modeling Language), XMI (XML Metadata Interchange), SysML (System Modeling Language), OCL (Object Constraint Language), UTP (UML Testing Profile), ALF (Action Language for Foundational UML), FUML (Semantics of a Foundational Subset for Executable UML Models), ReqIF (Requirements Interchange Format).

These documents are the scientific and methodical base for their application, further improvement, and development. A brief analysis of the most important scientific publications and papers starts with monographs and practical guidelines in the field of industrial development of CSHS.

The fundamental paper written by Dragan Milicev, the Serbian scientist and MBSE expert, Professor of University of Belgrade [5], outlines the principles and methods of applying modern information technologies based on the object-oriented paradigm and model-based approach for the industrial development of CSHS. This is especially valuable in the context of the problems considered in this article. Also, the paper provides recommendations and examples of using the fundamental UML (fUML) language, which is used to create and verify executable formal UML models.

In the monograph [6], the techniques and methods of applying the constructs and mechanisms of the SysML language are described in a summary and illustrated form containing practical examples, the idea and principles of this language are explained. This monograph is written by the group of active developers of many OMG methodical documents and specifications, and those who apply this knowledge in practice at such companies as Lockheed Martin and Raytheon Company: S. Friedenthal, A. Moore, R. Steiner. Useful information on applying the SysML language mechanisms for designing CSHS is presented in the monograph by Lenny Delligatti [7] (Lockheed Martin Corporation).

From among all publications of Russian organizations and researchers, it is worth to mention the papers by the ISP RAS team dealing with both theoretical and practical aspects of these problems. The theoretical foundations of the design and verification of CSHS based on a category-theoretic approach to metaprogramming are described in publications written by S. Kovalev, the leading ISP RAS researcher [8] [9]. They present the ways to apply category theory to solve the problem of representing heterogeneous software engineering technologies in a common format that would be convenient for their integration and coordination in the software system design life cycle. Particular attention is paid to such modern technologies as model checking development and aspect-oriented programming, for which universal category-theoretic semantic models are built.

One of the modern means to describe the architecture of software and hardware systems is Architecture Analysis & Design Language (AADL) [10]. On the basis of this language, the system for supporting the design and verification of MASIW onboard aircraft systems developed by ISP RAS together with GosNIIAS as part of the state program for the development of Integrated Modular Avionics (IMA) is being actively used. When developing MASIW, the following libraries and tools were used: Eclipse Modeling Framework, Graphical Editing Framework, Eclipse Team Providing, SVN Team Provider, GIT Team Provider. As noted in the article [11], the MASIW tools allow solving the following tasks:

- creation, editing, and management of models of hardware-software complexes (HSCs) using the AADL language;
- analysis of models for the sufficiency of hardware resources and interface consistency, the evaluation of the characteristics of projected data networks built in accordance with the AFDX standard (Avionics Full-Duplex Switched Ethernet);

- distribution of functional applications over computation modules, taking into account the limitations of the hardware platform resources and the requirements for the reliability and security of HSCs;
- generation of computer code and configuration data for VxWorks653 RT OS and termination units of the AFDX network.

An example of using the special extension of the AADL language – Error Model Annex (EMA) and the MASIW tool for modeling and analyzing the security of the designed HSCs is presented in [12]. The model is created using EMA, in which a finite-state machine (FSM) is developed for each component of HSCs. The states of FSM are normal states and emergencies, including dangerous and failure situations of this component. The effect of system component failures on other components is described by specifying the logical conditions for the propagation of errors between different types of components in different states, taking into account the probabilities of their occurrence. The following algorithms are used for risk analysis: Fault Tree Analysis, Failure Mode and Effects Analysis, Markov Analysis. The implementation of the approach described in this article helps to identify and eliminate the security-critical defects in design solutions at the design stage.

The ISP RAS team has developed the technology called UniTESK (Unified TEsting Specification based toolkit) for testing software interfaces. This is a unified set of testing tools based on specifications. UniTESK is unified due to the fact that the general testing methodology and general architecture can be used to test modules using almost all programming languages. Currently, there are the UniTESK implementations for C (CTESK), C ++ (C ++ TESK), Java (JavaTESK and Summer), Python (PyTESK). The UniTESK technology has two main differences from common testing tools [13]:

- UniTESK helps to describe the specifications of a software contract of modules in the form of pre- and post-conditions using the extensions of programming languages (in case of C ++ TESK, no extension is required);
- instead of manual development of test cases, UniTESK allows describing a generalized scenario – a compact description of test logic that allows the test sequence generator to call each specified interface in all its uses automatically and to verify the correctness of the result for compliance with a specified post-condition.

The next group of publications consists of papers devoted to the solution of particular problems of developing and verifying CSHS. The thesis written by A.V. Markov, the employee of Novosibirsk State Technical University, is devoted to the issues of automation of design and software analysis processes using the UML language and Petri nets [14]. The paper describes the software design methodology using UML sequence diagrams in the .xmi format and presents the method for their automatic convert to the .cpn format used to describe Petri nets. The result of using this method is hierarchical Petri nets being analyzed for verifying the software project, which is represented in the form of UML diagrams. The following solutions presented in this paper are the most valuable in practice:

- algorithm of transforming UML diagrams to Petri network;
- algorithm and rules of implementing inversion in Petri nets to check the reachability of the selected network state;
- algorithms and software for constructing and analyzing Petri nets to identify and eliminate defects in the developed software.

The review of modern methods for automatic test generation presented in [15] is quite useful. The paper describes the following methods:

- structural testing using symbolic execution;
- model-based testing;
- combinatorial testing;
- random testing;
- search-based testing.

The article [16] presents the automated method for making UML sequence diagrams using the description of UML use case diagrams and class diagrams. To implement this method, it is necessary to use the ATL language and metamodels of use case diagrams, class diagrams and sequence diagrams developed by the authors of the article, as well as the rules for obtaining the third diagram from the first and second ones. The result of this transformation is a sequence diagram in the XMI format, which is then converted to the XSLT format to display a sequence diagram in a graphical editor for viewing, analysis, and making changes. The disadvantage of the proposed algorithm is the lack of automatic correction of the original models if any new changes are made to a sequence diagram. This is due to the fact that the transformations using the ATL language are unidirectional – they work with read-only source models and create write-only target models.

In the work [17], experts at Shanghai University have described the approach to verify large-scale web projects by developing and analyzing the executable model of the corresponding software. To build this executable model, the authors have developed the method that uses live sequence charts (LSCs) as input data. A UML model using LSCs diagrams is transformed into a symbolic finite-state machine. Test scenarios are created by traversing a finite-state machine with the Depth-first Search method (DFS).

The paper [18] describes the method of automatic generation of computer code based on the project (architectural model) of a program presented in the ALF language. Of particular interest is the conceptual scheme of the mechanism for generating computer code from the project description in the ALF format using the rules in the extended Backus-Naur (EBNF) notation). The authors point out the following advantages of the tool to transform the model of the architecture of the ATL language: the ability to describe both declarative and imperative language constructs, the presence of means to combine modules that allow creating and reusing sets of transformation rules. The result is a Java code that corresponds to the Modisco Java metamodel.

The article [19] describes two methods for implementing automatic testing of real-time loaded systems using scenarios. In the first, the system is modeled as the network of timed automata (TA). In the second, it is modeled as a set of live sequence charts (LSCs) and requirements in the form of a separate LSC diagram to analyze. The authors of the article have developed temporal extensions for a subset of the core of the LSC language and defined its semantics based on tracing. The analyzed LSC diagram is transformed to its behavioral equivalent in the notation of the TA diagram. The correctness verification of a model is carried out by modeling the TA diagram in real time using Computational Tree Logic (CTL) followed by the comparison of the obtained result with the standard. Both methods are implemented with the tools of UPPAAL.

The paper [20] describes the method for generating unit cases based on the architecture of a model presented in the form of UML activity diagrams. The tests are created with the SMT/SAT solvers, which analyze the control flow graph of a program presented in A Modeling Language for Mathematical Programming (AMPL). This paper proposes test coverage criteria based on control flow analysis. Particular attention is paid to mixed integer nonlinear programming, as well as to the construction of logical formulas for OCL (Object Constraint Language) constraints.

One of the serious disadvantages of modern approaches is the lack of ability to take into account the composition and structure of designed systems, as well as to establish and synchronize the relations between system requirements and design elements. To eliminate these disadvantages, the paper [21] proposes to make a system design based on SysML behavioral diagrams. To verify automatically the project created in this way, it is proposed to use the following methods:

- transformation of SysML activity diagrams to modular Petri nets presented in PNML (Petri Net Markup Language);
- mathematics and such tools as CPN Tools and SPIN for analyzing Petri nets;
- algorithm for verifying the time requirements in SysML activity diagrams, which are pre-converted to formulas of Linear Temporal Logic (LTL) using Active Temporal Requirement Language (AcTRL) developed by the authors.

To create tools for the dynamic verification and validation of project behavioral models, it is proposed to use Executable Domain-specific Modeling Languages (xDSMLs) in [22]. Means based on them make it possible to monitor the states of analyzed models (transitions, events, variable values) during their execution. The new generative approach based on a multidimensional and domain-specific trace metamodel is proposed. This method helps to construct and manage execution traces for models corresponding to a specified xDSML. According to the authors of this paper, this method has higher performance compared to the standard UML metamodel due to the ability to exclude redundant data from processing (for example, analyzed traces) using the mechanisms of the corresponding xDSML.

To conclude the analysis of publications and the solutions presented in them, the following ideas can be summarized:

- main efforts of researchers are aimed at developing methods and tools for the automated generation and verification of software implementations of CSHS [13] [14] [15] [17] [18] [20]; fewer efforts are aimed at automating the development and verification of design solutions [11] [19] [20] [22]; there are practically no solutions for the automated formation and verification of a set of requirements;
- mathematics and analysis of Petri nets, SMT/SAT solvers, such modeling languages as AADL, UML, fUML, SysML and domain-specific languages (xDSMLs) developed on their basis are used as the basic mathematical models and tools for automatic verification based on these models.

In this regard, the main purpose of research and papers, the results of which are presented in this article, was to develop a model, algorithmic and methodical support of the processes of building and verifying formal models of requirements and the architecture of CSHS used in state CII objects.

For create unified conceptual, language and instrumental environment for the development and verification of analysis models and the architecture, it is proposed to use:

- UML, OCL, fUML and ALF modeling languages;
- VM fUML, SPIN (Promela), Rodin (Event-B), SMT-Lib, Z3, CVC-4, Alt-ERGO;
- environment, libraries and software products implemented within the Eclipse project: Eclipse Modeling Framework, Graphical Editing Framework, Papyrus, Moka.

The choice of these models, languages and tools is conditioned by the following circumstances. First, their development is actively supported by leading development enterprises and consumer organizations of CSHS. The second is that both the technologies and means based on them are open and available for study, application, and improvement.

3. Models and Algorithms of Formal Description of the Requirements for a System Based on the Original Informal Representation

To solve the problem of building a formal description of the requirements for automated systems and software, you must perform the following operations and procedures:

1. First, additional content control elements are developed and installed in a text editor (MS Word or Writer). These elements are XML schemas (*tz_as.xsd*, *tz_sw.xsd.*) based on a universal Requiment Interchage Format (ReqIF). XML schemas describe the composition and structure of requirements for automated systems and software defined in the relevant normative-methodical documents.
2. Then in the environment of a text editor in accordance with the established in the previous step xml schemas (tz_as.xsd and tz_sw.xsd) structured text documents are developed containing requirements to the system.
3. The next step is the automatic generation of the first version of the formal model of the set of requirements. to implement this procedure, use the metamodel shown in the Fig. 1. This metamodel is a conceptual and logical union of a use case diagram and a class diagram.

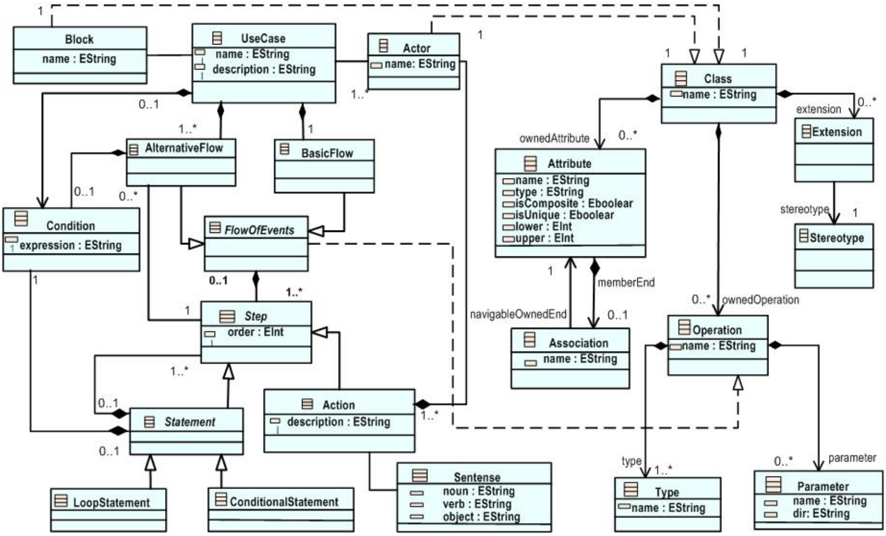


Fig.1. Comprehensive model of the use case diagram and class diagram

To develop this metamodel, the official specifications of these diagrams on the OMG website and the models proposed in publications [16] [22] [23] were used. In addition to the explicit establishment of relationships between diagram elements of these diagrams, the proposed model includes the new class – “Sentence” and excludes two classes – “Subject” and “Agent”. The program implementing the generation procedure uses the *xmi* representation of this metamodel and developed before structured text documents containing requirements to the system.

Each *i*-th use case is a functional requirement and is described as follows:

$$uc_i = (nameuc_i, actor_k, function_i, block_j),$$

where $nameuc_i$ – use case name uc_i;

$actor_k$ – user or external system that initiates uc_i;

$function_i$ – system function that implements uc_i;

$block_j$ – system component that implements $function_i(input, operation, result)$,

where $input$input – input data;

$operation$operation – algorithm that implements $function_i$;

$result$result – result of the implementation of $function_i$.

$function_i = (basicflow, alterflow)$,

where

$basicflow$ – algorithm that implements the main flow of the function;

$altersflow$ – algorithms that implements alternative flows of the function.

The *class* construction is developed for each functional block (module) and information object. Its attributes, operations (methods), restrictions and semantics are specified. The sets of interacting classes are combined into class diagrams – d_class. Formally, a class diagram can be described as follows:

$d\_class = (Classes, Relations)$,

where $Classes = \{ class_i\}\ i = 1, \dots, I$ – diagram classes;

$Relations$ – class relationships;

$class = (nameclass, attributes, operations)$,

$Relations = \{R_{as}, R_{in}, R_{ag}, R_{de}, R_{sp}, R_{re}\}$ – relations between classes of the following six types;

R_{as} – associations;

R_{in} – inheritances;

R_{ag} – aggregations;

R_{de} – dependences;

R_{sp} – specializations;

R_{re} – realizations.

The next step in building a requirements model is to develop non-functional requirements specifications for each system function:

$d\_reqs = (r_1^{f_i}, r_2^{f_i}, r_3^{f_i}, r_4^{f_i} \dots)$,

where $r_1^{f_i}$ – requirements for the efficiency of execution of f_i;

$r_2^{f_i}$ $\mathrm{r}_2^{\mathrm{f_i}}$ – performance requirements (for example, the amount of data stored, processed and transmitted, the number of users, the number and size of requests per unit of time, etc.);

$r_3^{f_i}$ $\mathrm{r}_3^{\mathrm{f_i}}$ – requirements for reliability (availability rate, uptime, recovery time, etc.);

$r_4^{f_i}$ – security requirements.

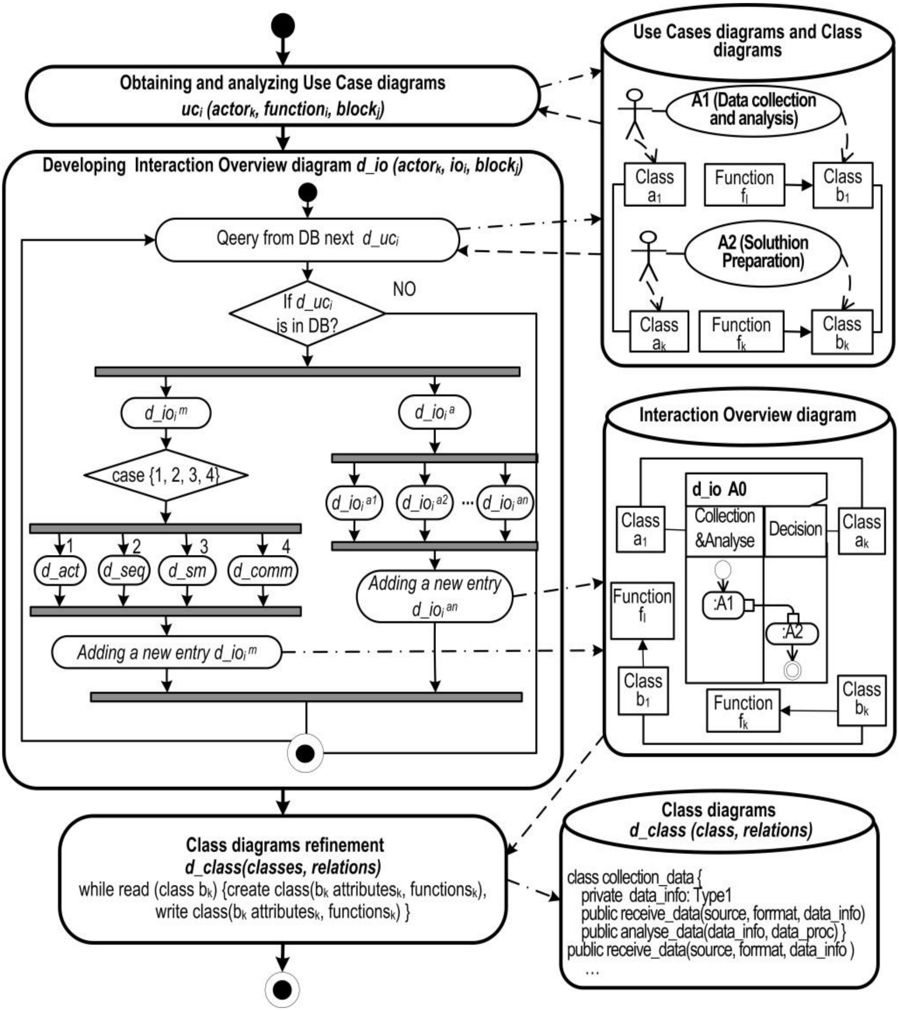
The model built in this way is preliminary, and it is used as input data for the algorithm for building a model formal requirements in the fUML language which described in the next section.

4. Algorithm for Building a Formal Model Requirements

The scheme of the algorithm that implements the second stage of the procedure of building a formal requirements model using the fUML language is shown in Fig. 2. Use case diagrams (UCDs) – *d_uc*, class diagram (CDs) – *d_class* and requirements diagrams (RDs) – *d_reqs* are used as initial data. The analyst and future system user develop an interaction overview diagram (IOD) – *d_io* for each UCD (*d_uc*). In this diagram, the functions implemented by the system are described from the user's point of view in more detail using activity diagrams, sequence diagrams, and statechart diagrams. Formally, an interaction overview diagram can be represented as follows: $d\_io_i = \left(actor_k, io_i^{fi}, block_j\right)$, where $io_i^{fi} =$ ($\mathrm{io}_\mathrm{i}^\mathrm{fi} = \left(\mathrm{a}_\mathrm{k}, \mathrm{f}_\mathrm{i}, \mathrm{b}_\mathrm{j}\right)$ io_i^{fim}, io_i^{fia}) describes the algorithm for implementing the function by the *j*-th block (class) of the designed software, which includes the description of main (io_i^{fim}) and alternative (io_i^{fia}) flows.

Alternate flows describe the operation of programs in case of abnormal situations, such as erroneous user actions, unexpected influences from the external environment, etc. The main and alternative flows can have subordinate flows, which are described in IOD using frames with “ref”. The subordinate IOD flows show the work of a program from the user's point of view and can be represented with activity diagrams, sequence diagrams or statechart diagrams depending on the features of the functioning of CSHS and ways of the interaction with the user and environment. To describe the procedure and possibility of realization of those or other threads are used pre - and post- condition.

the model of requirements constructed in this way should be subjected to validation and verification procedures. The validation procedure is to assess the completeness and correctness of the set of requirements. It is carried out both by software tools and by the informal expertise of specialists in a particular subject area. Such properties of a model as consistency, systematicity, non-redundancy, security, liveliness, absence of deadlocks, impossible operations, looping are checked during verification. The verification of the requirements model is carried out through its execution and testing in the *fUML* virtual machine environment and analysis using *SAT/SMT solvers*.



Obtaining and analyzing Use Case diagrams
uc_i (actor_k, function_i, block_j)
Developing Interaction Overview diagram d_io (actor_k, io_i, block_j)
Qeery from DB next d_uc_i
If d_uc_i is in DB?
NO
d_io_i^m
d_io_i^a
case {1, 2, 3, 4}
1
2
3
4
d_act
d_seq
d_sm
d_comm
d_io_i^a1
d_io_i^a2
d_io_i^an
Adding a new entry d_io_i^an
Adding a new entry d_io_i^m
Class diagrams refinement
d_class(classes, relations)
while read (class b_k) {create class(b_k attributes_k, functions_k), write class(b_k attributes_k, functions_k) }
Use Cases diagrams and Class diagrams
A1 (Data collection and analysis)
Class a_1
Function f_l
Class b_1
A2 (Soluthion Preparation)
Class a_k
Function f_k
Class b_k
Interaction Overview diagram
d_io A0
Collection &Analyse
Decision
Class a_1
Class a_k
Function f_l
:A1
:A2
Class b_1
Function f_k
Class b_k
Class diagrams
d_class (class, relations)
class collection_data {
private data_info: Type1
public receive_data(source, format, data_info)
public analyse_data(data_info, data_proc) }
public receive_data(source, formmat, data_info)
…

Fig.2. Construction algorithm of the technical project model

The description of these methods and tools is provided in sections 5 and 6.

5. Algorithm for Building a formal model of architecture of CSHS

The architecture development of CSHS is implemented in accordance with the algorithm shown in Fig. 3

The initial data are the interaction overview diagram – *d_io*, diagrams of quality requirements for the implementation of functions – *d_reqs*, class diagrams – *d_class* and the requirements for development technologies and operating environment.

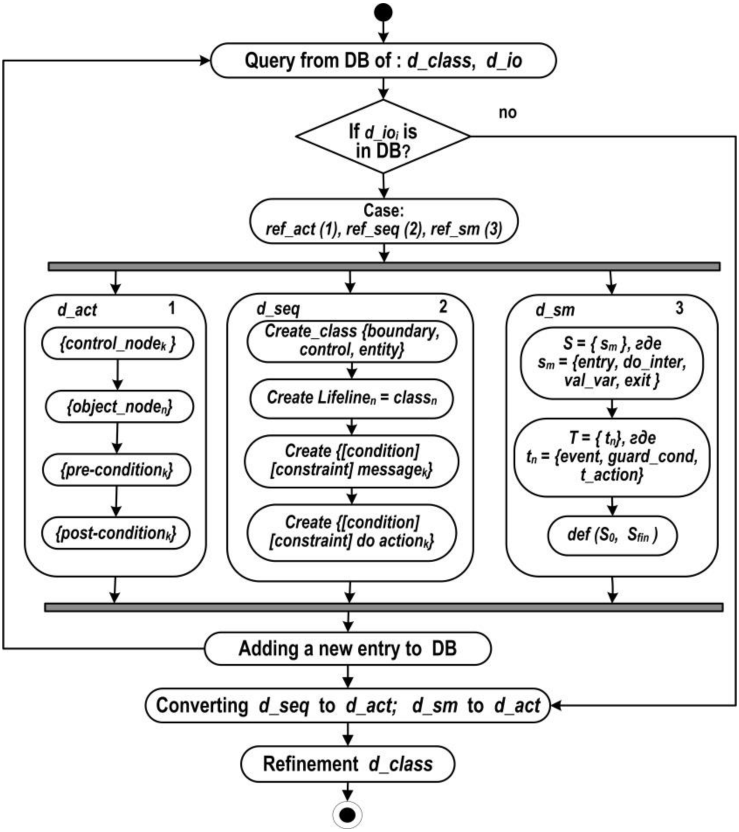

Query from DB of : d_class, d_io
If d_io_i is in DB?
no
Case: ref_act (1), ref_seq (2), ref_sm (3)
d_act
1
{control_node_k }
{object_node_n}
{pre-condition_k}
{post-condition_k}
d_seq
2
Create_class {boundary, control, entity}
Create Lifeline_n = class_n
Create {[condition] [constraint] message_k}
Create {[condition] [constraint] do action_k}
d_sm
3
S = { s_m }, где s_m = {entry, do_inter, val_var, exit }
T = { t_n}, где t_n = {event, guard_cond, t_action}
def (S_0, S_fin)
Adding a new entry to DB
Converting d_seq to d_act; d_sm to d_act
Refinement d_class

Fig.3. Algorithm of the architecture model development

In each interaction overview diagram (*d_io*) searches for a reference to activity diagrams (*ref_act*), sequence diagrams (*ref_seq*) and statechart diagram (*ref_sm*). If such references are found, the architect is asked to build or modify the corresponding diagrams. Activity diagrams are described using *control nodes* (*control_node: decision node, merge node, fork node, join node, interaction, interaction use); object nodes (object_node*); *pre-condition*s and *post-conditions*.

When constructing sequence diagrams, the additional *boundary*, *control*, and *entity* classes are first created, which perform the functions of intermediate (boundary) classes, control, and information objects, respectively. Then the *lifelines* are defined corresponding to classes that exchange *messages*. Messages are defined by the conditions and limitations of their transmission and reception, and the actions that are performed (*do action*). When developing state diagrams, the $S = \{sm\}$ states and $T = \{tn\}$ transitions between them are defined. Each sm state consists of a description of the attributes - *val_var*, as well as the actions performed: *entry* – at the entrance, *do_inter* – internal, *exit* – at the exit.

The tn transitions include descriptions of the event initiating this transition – *event*, the pre- and post- implementation conditions – *guard_cond* and actions that must be performed before the actions of a new state – *t_action*).

The constructed diagrams are added to the database. To obtain a consistent and bound set of CSHS technical project (architecture) diagrams, class diagrams (*d_class*) and requirements diagrams (*d_regs*) are refined by establishing relations with new activity, sequences and statechart diagrams that were developed or modified. Fig. 4 shows the diagram illustrating the relationships between class and activity diagrams. Each d_act_i has a relationship with a specific class by describing the algorithm for implementing the corresponding class *method*.

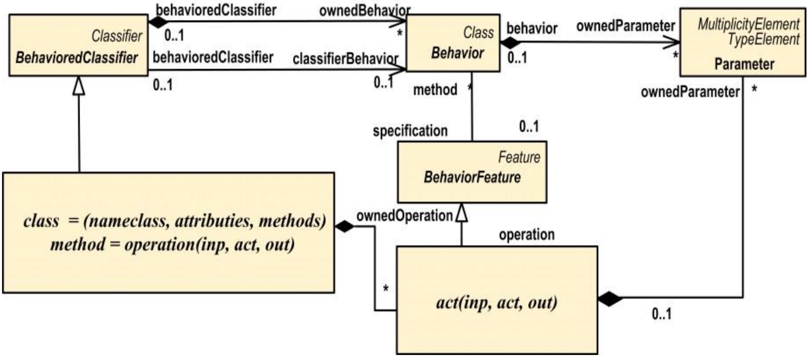


Fig. 4. Relationships and dependencies between the main components of the technical project model: class and activity

To implement the architecture model verification procedure in the virtual machine environment, *fUML* sequence diagrams (*d_seq*) and statechart diagrams (*d_sm*) are transformed to activity diagrams (*d_act*), which are then described in the language *ALF (Action Language for Foundational UML).*

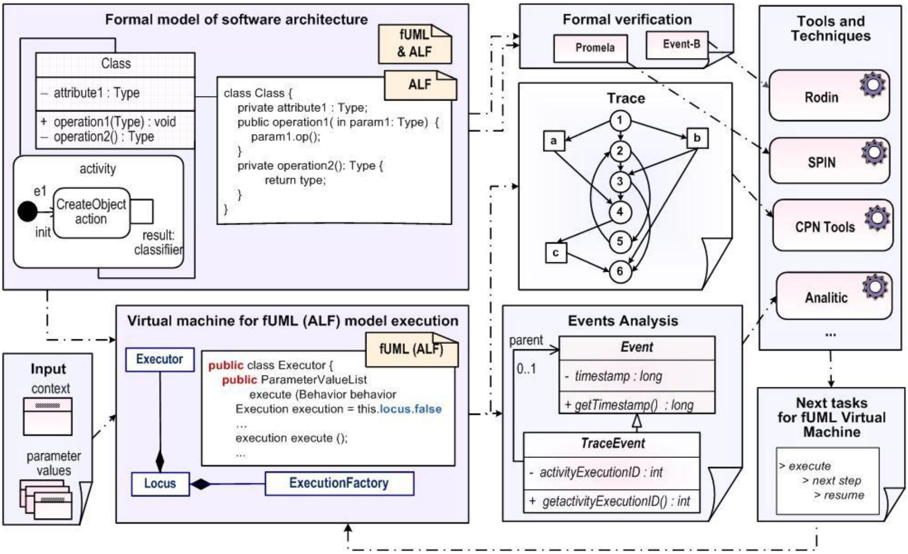
Fig. 5 presents the diagram illustrating the verification procedure of the formal *fUML* model of the CSHS architecture in a virtual machine environment consisting of three components: *ExecutionFactory, Executor* and *Locus.*

ExecutionFactory is used to create instances of the *visitor* semantic classes corresponding to the executable elements of the *fUML* model. The *Executor* class is a top-level abstraction for the executable *fUML* model and provides three operations:

- *evaluate* – evaluate a value specification, returning the specified value

- *execute* – synchronously execute a behavior, given values for its input parameters and returning values for its output behaviors;
- *start* – asynchronously start the execution of a stand-alone or classifier behavior, returning a reference to the instance of the executing behavior or of the behaviored classifier.

Each execution is performed on a specific VM (*Locus*), which is the abstraction of a physical or virtual computer capable of executing and verifying *fUML* models.



```
class Class {
   private attribute1 : Type;
   public operation1( in param1: Type) {
      param1.op();
   }
   private operation2(): Type {
         return type;
   }
}
```

```
public class Executor {
   public ParameterValueList
        execute (Behavior behavior
     Execution execution = this.locus.false
     ...
     execution execute ();
     ...
```

Fig.5. Scheme of executable fUML-model verification

The following basic requirements are imposed on the software architecture of CSHS:

- completeness of the implementation of functional requirements defined in the interaction overview diagrams – *d_io*;
- completeness and correctness of the implementation of non-functional requirements defined in requirements diagrams – *d_reqs*;
- coherence and consistency of all model diagrams;
- lack of redundancy.

Testing the architecture model in the *fUML* virtual machine environment also makes it possible to detect defects that can lead to security and liveliness violations, the occurrence of deadlocks, impracticable operations, and loops. In addition, it is advisable to use SAT/SMT solvers to verify the architecture model. The description of their application is presented in the next section of the article.

6. Methods for Constructing Test Scenarios to Verify Models of Requirements, Architecture and Implementation of CSHS using SAT/SMT solvers

The main stages of the process of constructing test scenarios to verify models of requirements, architecture, and implementation of CSHS are presented in Fig. 6:

– building a control flow graph (CFG);

– description of CFG in language ALF;

– generation of test scenarios (TSs) for verification of a set of requirements and technical project (architecture);

– generation of TSs for implementation verification;

– adding test scenarios to database (DB).

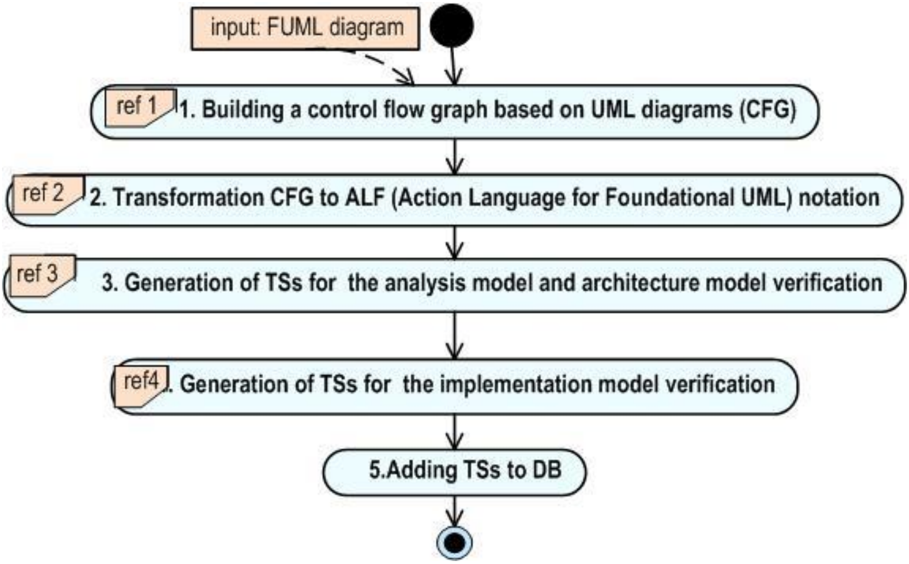

input: FUML diagram
ref 1
1. Building a control flow graph based on UML diagrams (CFG)
ref 2
2. Transformation CFG to ALF (Action Language for Foundational UML) notation
ref 3
3. Generation of TSs for the analysis model and architecture model verification
ref4
. Generation of TSs for the implementation model verification
5.Adding TSs to DB

Fig.6. Generalized algorithm of test scenarios development for verification of requirement, architecture and implementation models

With the help of this algorithm, the requirements model and the architecture model can be verified. The original format for representing these models is *.xmi*. Based on these descriptions, the corresponding verifiable CFG model is built, in which both functional and non-functional requirements for the system being developed are taken into account. To represent non-functional requirements, *Object Constraint*

Language (OCL) is used. A SMT/SAT solver checks CFG for defects and, if they are found, creates counterexamples. Using them, the developer determines the causes of defects and makes the necessary corrections to the analyzed artifact. To implement this approach, it is proposed to use the ALT-ERGO, CVC4 and Z3 solvers, integrated into the Frama-C framework [24] [25].

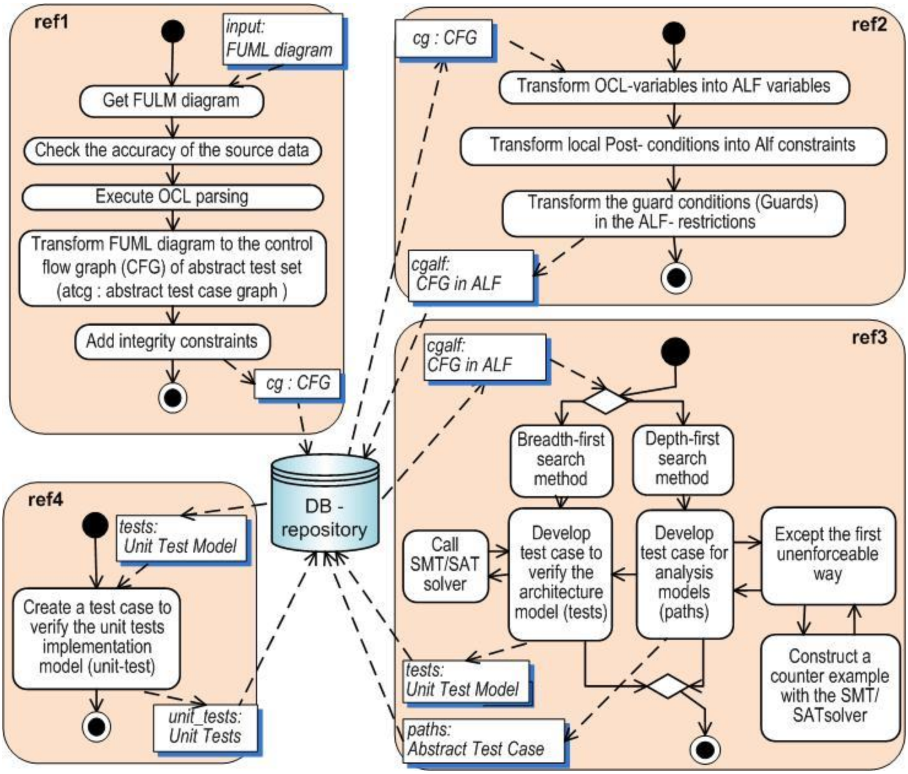

ref1
input:
FUML diagram
Get FULM diagram
Check the accuracy of the source data
Execute OCL parsing
Transform FUML diagram to the control flow graph (CFG) of abstract test set (atcg : abstract test case graph)
Add integrity constraints
cg : CFG
ref2
cg : CFG
Transform OCL-variables into ALF variables
Transform local Post- conditions into Alf constraints
Transform the guard conditions (Guards) in the ALF- restrictions
cgalf:
CFG in ALF
ref3
cgalf:
CFG in ALF
Breadth-first search method
Depth-first search method
Call SMT/SAT solver
Develop test case to verify the architecture model (tests)
Develop test case for analysis models (paths)
Except the first unenforceable way
Construct a counter example with the SMT/ SATsolver
tests:
Unit Test Model
paths:
Abstract Test Case
DB - repository
ref4
tests:
Unit Test Model
Create a test case to verify the unit tests implementation model (unit-test)
unit_tests:
Unit Tests

Fig. 7. Detailed algorithm of test scenarios development for verification models of requirement, architecture and implementation

Concluding the presentation of the developed models and algorithms, let us present a generalized scheme for the implementation of software-controlled process of development and verification of formal models of requirements and architecture of CSHS, which provides end-to-end quality control of all artifacts of the life cycle of CSHS (Fig. 8).

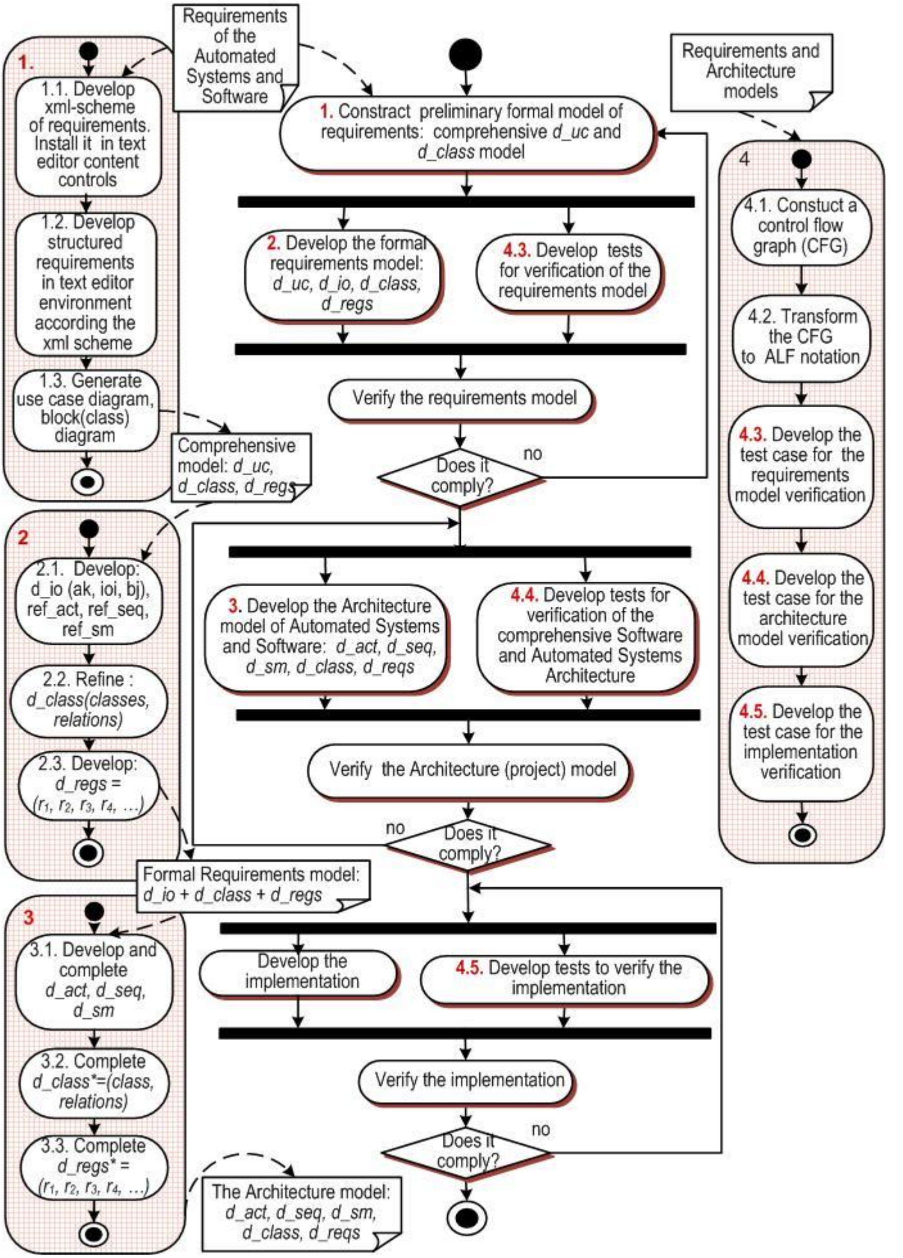

Requirements of the Automated Systems and Software
Requirements and Architecture models
1.
1.1. Develop xml-scheme of requirements. Install it in text editor content controls
1.2. Develop structured requirements in text editor environment according the xml scheme
1.3. Generate use case diagram, block(class) diagram
Comprehensive model: d_uc, d_class, d_regs
1. Constract preliminary formal model of requirements: comprehensive d_uc and d_class model
2. Develop the formal requirements model: d_uc, d_io, d_class, d_regs
4.3. Develop tests for verification of the requirements model
Verify the requirements model
Does it comply?
no
4
4.1. Constuct a control flow graph (CFG)
4.2. Transform the CFG to ALF notation
4.3. Develop the test case for the requirements model verification
4.4. Develop the test case for the architecture model verification
4.5. Develop the test case for the implementation verification
2
2.1. Develop: d_io (ak, ioi, bj), ref_act, ref_seq, ref_sm
2.2. Refine : d_class(classes, relations)
2.3. Develop: d_regs = (r₁, r₂, r₃, r₄, …)
3. Develop the Architecture model of Automated Systems and Software: d_act, d_seq, d_sm, d_class, d_reqs
4.4. Develop tests for verification of the comprehensive Software and Automated Systems Architecture
Verify the Architecture (project) model
no
Does it comply?
Formal Requirements model: d_io + d_class + d_regs
3
3.1. Develop and complete d_act, d_seq, d_sm
3.2. Complete d_class*=(class, relations)
3.3. Complete d_regs* = (r₁, r₂, r₃, r₄, …)
Develop the implementation
4.5. Develop tests to verify the implementation
Verify the implementation
Does it comply?
no
The Architecture model: d_act, d_seq, d_sm, d_class, d_reqs

Fig.8. Stages of implementation of software-controlled of the process of development and verification of software and hardware systems

The main stages of the implementation of this approach are:

1. Construction of a preliminary formal model of requirements for CSHS in the form of a set of interrelated use case diagrams, class diagrams, and requirements diagrams.
2. Development of a formal requirements model in the form of a set of interrelated use case diagrams, overview interaction diagrams, class diagrams, and requirements diagrams.
3. Development and verification of the formal model of the architecture of CSHS through testing in the *fUML* virtual machine environment and analysis using SAT/SMT solvers – ALT-ERGO, CVC4 and Z3.
4. Development and verification of the software implementation.

7. Conclusion

One of the most important directions of improving the development processes and achieving the required quality indicators of complex software and hardware systems is the creation and implementation in practice of their industrial development of model-based technologies for justifying requirements, design, and implementation followed by the procedures of their formal verification and semi-formal validation. Currently, the most problematic issues are related to the verification of requirements and the CSHS architecture. To solve these problems, it is proposed to implement the approach described in this article. The distinctive features of this approach are:

- formation and use of a single model-language and information-software environment for the development and verification of formal models of requirements, architecture and software implementation based on the necessary and sufficient set of interrelated fUML diagrams and the model of internal and inter-model relations developed for them;
- implementation of the software-controlled development process of CSHS in accordance with the developed algorithm that performs sequential-iterative operations of generating and transforming formal models of requirements and architecture presented in *fUML, XMI, ALF*, and that also performs their verification in the fUML virtual machine environment and SMT/SAT solvers.

To implement the proposed approach, the following models, algorithms, and methods were developed:

- algorithm of a formal description of the requirements for the developed system based on the initial informal representation;
- fUML diagram models that are necessary and sufficient to develop complete, correct and consistent formal models of requirements and architecture;
- models, algorithms and guidelines for the development of formal models requirements and the architecture in languages fUML, XMI and ALF;

- verification algorithms for models of requirements and the architecture of CSHS in the environment of fUML virtual machine;
- verification of the formal model of the architecture and program implementation through the analysis using the SAT/SMT solvers.

Currently, work is underway to create a set of software tools to ensure the implementation of this approach. The development tools, libraries, and applications implemented in the Eclipse project (EMF, GMP, RCP, Papyrus, Moka, Titan) are used as a development environment and prototypes. The implementation of this software package in the relevant technological processes at companies will ensure the most complete accounting and correct implementation of requirements for functional and operational characteristics, environment and conditions for the use of CSHS. It will also significantly reduce the cost of finding and eliminating the most critical and resource-intensive defects made at the stages of the formation of requirements and design of their architecture.

References

[1]. Federal law "On security of critical information infrastructure of the Russian Federation". 12.07.2017 (in Russian)

[2]. Systems Engineering and Software Engineering, https://www.sebokwiki.org/wiki/Systems_Engineering_and_Software_Engineering. (accessed 25.07.2018).

[3]. Laura. Introduction To Model-Based System Engineering (MBSE) and SysML.https://www.incose.org/docs/default-source/delaware-valley/mbse-overview-incose-30-july-2015.pdf. (accessed 21.06.2018).

[4]. The Standish Group Report CHAOS. https://www.projectsmart.co.uk/white-papers/chaos-report.pdf. (accessed 25.08.2018).

[5]. Dragan Milicev. Model-Driven Development with Executable UML. John Wiley & Sons, 2009, 720 p.

[6]. Sanford Friedenthal, Alan Moore, Rick Steiner. A Practical Guide to SysML: The Systems Modeling Language. Morgan Kaufmann, 3 edition, 2014, 630 p.

[7]. Lenny Delligatti. SysML Distilled: A Brief Guide to the Systems Modeling Language. Addison-Wesley Professional, 2013, 304 p.

[8]. Kovalev S.P. Theoretical and categorical approach to metaprogramming. M., IPU Russian Academy of Sciences, 2014, 112 p. (in Russian)

[9]. Kovalev S.P. Category-Theoretic Approach to Software Systems Design. Journal of Mathematical Sciences, vol. 214, issue 6, 2016, pp. 814–853.

[10]. Peter H. Feiler, David P. Gluch. Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. Addison-Wesley Professional, 2012, 480 p.

[11]. D.V., Buzdalov, S.V. Zelenov, E.V. Kornykhin, A.K. Petrenko, V.A. Fear, A.A. Ognenko, A.V. Khoroshilov. Design tools for integrated modular avionics systems. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 1, 2014, pp. 201-230. DOI: 10.15514/ISPRAS-2014-26(1)-6 (in Russian)

[12]. S.V. Zelenov, S.A. Zelenova, Modeling of hardware and software systems and analyze their security. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 5, 2017, pp. 257-282. DOI: 10.15514/ISPRAS-2017-29(5)-13 (in Russian)

[13]. http://www.ispras.ru/technologies/unitesk (accessed 17.10.2018) (in Russian)

[14]. Markov, A.V., automation of design and analysis software using UML and Petri nets. PhD Thesis, NSTU, Novosibirsk, 2015 (in Russian).

[15]. Saswat Anand et al. An Orchestrated Survey on Automated Software Test Case Generation. Journal of Systems and Software, vol. 86, Issue 8, 2013, pp. 1978-2001.

[16]. Yachai Limpiyakorn, Photchana Sawprakhon. Sequence Diagram Generation with Model Transformation Technology. In Proc. of the International MultiConference of Engineers and Computer Scientists, IMECS 2014, vol I

[17]. Liping Li, Honghao Gao, Tang Shan. An Executable Model and Testing for Web Software based on Live Sequence Charts. In Proc. of the 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS).

[18]. Thomas Buchmann and Alexander Rimer. Unifying Modeling and Programming with ALF. The Second International Conference on Advances and Trends in Software Engineering, vol I, IARIA, 2016. pp .10-15.

[19]. Shuhao Li, Sandie Balaguer et al. Scenario-based verification of real-time systems using Uppaal. Formal Methods in System Design, vol. 37, Issue 2–3, 2010, pp 200–264

[20]. Felix Kurth. Automated Generation of Unit Tests from UML Activity Diagrams using the AMPL Interface for Constraint Solvers. Master Thesis, Hamburg University of Technology (TUHH), 2014.

[21]. Messaoud Rahim, Malika Boukala-Ioualalen, Ahmed Hammad. Petri Nets Based Approach for Modular Verification of SysML Requirements on Activity Diagrams. PNSE'14, a satellite event of Petri Nets 2014 and ACSD 2014, Tunis, Tunisia, pp 233-248.

[22]. Erwan Bousse, Tanja Mayerhofer, Benoit Combemale, Benoit Baudry. Advanced and efficient execution trace management for executable domain-specific modeling languages. Software & Systems Modeling, 2017, https://link.springer.com/article/10.1007/s10270-017-0598-5 (accessed 20.07.2018)

[23]. D. Savic, S. Vlajic, S. Lazarevic. Use Case specification using the SilabReq domain specific language. Computing and Informatics, vol. 34, 2015, 877–910.

[24]. Yu Feng, Ruben Martins, Osbert Bastani, and Isil Dillig. 2018. Program Synthesis using Conflict-Driven Learning. In Proc.b of 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI’18). ACM, New York, NY, USA, 16 p.

[25]. Efremov D. V., Mandrykin M. U. Formal verification of Linux kernel library functions. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 6, 2017, pp. 49-76. DOI: 10.15514/ISPRAS-2017-29(6)-3 (in Russian)

Методика и средства разработки и верификации формальных fUML моделей требований и архитектуры сложных программно-технических систем

А.В.Самонов <a.samonov@mail.ru>
Г.Н.Самонова <g.samonova@mail.ru>
Военно-космическая академия имени А.Ф. Можайского,
197088, Россия, Санкт-Петербург, ул. Ждановская, д.13

Аннотация. В статье представлены модели и алгоритмы обеспечения сквозного контроля качества сложных программно-технических систем (СПТС) посредством реализации программно-управляемого процесса разработки и верификации формальных моделей требований и архитектуры СПТС. Дан анализ научных публикаций и нормативно-методической базы в области разработки и применения на практике модельно-ориентированного подхода. Установлено, что наименее обеспеченными модельными, алгоритмическими и программными решениями являются вопросы, связанные с разработкой полного и корректного набора требований, а также с формализацией и верификацией технических проектов СПТС. Предложены способы решения существующих проблем посредством формирования единой модельно-языковой и информационно-программной среды разработки и верификации формальных моделей требований и архитектуры СПТС, построенных на основе оптимального набора взаимосвязанных fUML диаграмм, представленных в нотации языка ALF и верифицируемых в среде виртуальной машины fUML и с помощью SAT/SMT решателей.

Ключевые слова: верификация и валидация; диаграммы активности; диаграммы классов; жизненный цикл автоматизированных систем; модели архитектуры; модели требований; проектирование и реализация; программно-технические системы.

DOI: 10.15514/ISPRAS-2018-30(5)-8

Для цитирования: Самонов А.В., Самонова Г.Н. Методика и средства разработки и верификации формальных fUML моделей требований и архитектуры сложных программно-технических систем. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 123-146 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(5)-8

Список литературы

[1]. Федеральный закон «О безопасности критической информационной инфраструктуры Российской Федерации». 12.07.2017 г.

[2]. Systems Engineering and Software Engineering https://www.sebokwiki.org/wiki/Systems_Engineering_and_Software_Engineering. (дата обращения 25.07.2018).

[3]. Laura E. Hart. Introduction to Model-Based System Engineering (MBSE) and SysML https://www.incose.org/docs/default-source/delaware-valley/mbse-overview-incose-30-july-2015.pdf. (дата обращения 21.06.2018).

[4]. The Standish Group Report CHAOS. https://www.projectsmart.co.uk/white-papers/chaos-report.pdf. (дата обращения 25.08.2018).

[5]. Dragan Milicev. Model-Driven Development with Ex ecutable UML. John Wiley & Sons, 2009, 720 p.

[6]. Sanford Friedenthal, Alan Moore, Rick Steiner. A Practical Guide to SysML: The Systems Modeling Language. Morgan Kaufmann, 3 edition, 2014, 630 p.

[7]. Lenny Delligatti. SysML Distilled: A Brief Guide to the Systems Modeling Language. Addison-Wesley Professional, 2013, 304 p.

[8]. Ковалёв С.П. Теоретико-категорный подход к метапрограммированию. М., ИПУ РАН, 2014, 112 стр.

[9]. Ковалев С. П. Теоретико-категорный подход к проектированию программных систем. Фундаментальная и прикладная. математика, том 19, вып. 3, 2014, стр. 111–170.

[10]. Peter H. Feiler, David P. Gluch. Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. Addison-Wesley Professional, 2012, 480 p.

[11]. Д.В. Буздалов, С.В. Зеленов, Е.В. Корныхин, А.К. Петренко, А.В. Страх, А.А. Угненко, А.В. Хорошилов. Инструментальные средства проектирования систем интегрированной модульной авионики. Труды ИСП РАН, том 26, вып. 1, 2014, стр. 201-230. DOI: 10.15514/ISPRAS-2014-26(1)-6

[12]. Зеленов С.В., Зеленова С.А. Моделирование программно-аппаратных систем и анализ их безопасности. Труды ИСП РАН, том 29, вып. 5, 2017 г., стр. 257-282. DOI: 10.15514/ISPRAS-2017-29(5)-13

[13]. http://www.ispras.ru/technologies/unitesk (дата обращения 17.10.2018)

[14]. Марков А.В. Автоматизация проектирования и анализа программного обеспечения с использованием языка UML и сетей Петри. Канд. дис., Новосибирск, НГТУ, 2015.

[15]. Saswat Anand et al. An Orchestrated Survey on Automated Software Test Case Generation. Journal of Systems and Software, vol. 86, Issue 8, 2013, pp. 1978-2001.

[16]. Yachai Limpiyakorn, Photchana Sawprakhon. Sequence Diagram Generation with Model Transformation Technology. In Proc. of the International MultiConference of Engineers and Computer Scientists, IMECS 2014, vol I,

[17]. Liping Li, Honghao Gao, Tang Shan. An Executable Model and Testing for Web Software based on Live Sequence Charts. In Proc. of the 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS).

[18]. Thomas Buchmann and Alexander Rimer. Unifying Modeling and Programming with ALF. The Second International Conference on Advances and Trends in Software Engineering, vol I, IARIA, 2016. pp .10-15.

[19]. Shuhao Li, Sandie Balaguer et al. Scenario-based verification of real-time systems using Uppaal. Formal Methods in System Design, vol. 37, Issue 2–3, 2010, pp 200–264

[20]. Felix Kurth. Automated Generation of Unit Tests from UML Activity Diagrams using the AMPL Interface for Constraint Solvers. Master Thesis, Hamburg University of Technology (TUHH), 2014.

[21]. Messaoud Rahim, Malika Boukala-Ioualalen, Ahmed Hammad. Petri Nets Based Approach for Modular Verification of SysML Requirements on Activity Diagrams. PNSE'14, a satellite event of Petri Nets 2014 and ACSD 2014, Tunis, Tunisia, pp 233-248.

[22]. Erwan Bousse, Tanja Mayerhofer, Benoit Combemale, Benoit Baudry. Advanced and efficient execution trace management for executable domain-specific modeling languages. Software & Systems Modeling, 2017,

https://link.springer.com/article/10.1007/s10270-017-0598-5 (дата обращения 20.07.2018)

[23]. D. Savic, S. Vlajic, S. Lazarevic. Use Case specification using the SilabReq domain specific language. Computing and Informatics, vol. 34, 2015, 877–910.

[24]. Yu Feng, Ruben Martins, Osbert Bastani, and Isil Dillig. 2018. Program Synthesis using Conflict-Driven Learning. In Proc. of 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'18). ACM, New York, NY, USA, 16 p.

[25]. Ефремов Д.В, Мандрыкин М.У. Формальная верификация библиотечных функций ядра Linux. Труды ИСП РАН, том 29, вып. 6, 2017 г., стр. 49-76. DOI: 10.15514/ISPRAS-2017-29(6)-3