Method to Convert UML State Chart Diagram to the Virtual Object Mock-Up

N. N. Voit(✉) and S. I. Bochkov(✉)

Ulyanovsk State Technical University, Ulyanovsk, Russia
n.voit@ulstu.ru

Abstract. Virtual reality technology in engineering is one of the most effective tools of solutions of various tasks in product development: prototyping, modeling, modifying. Since VR objects tend to fully simulate the behavior of real products, the volume of project documentation tremendously arises, therefore it becomes hard to model VR objects being based upon it. In this paper, the authors propose a method converting the UML state chart formal description of the product to the virtual object model. The key feature of the method is the mapping event function which connects input signals and user actions. The method is hardware-free, so it is not attached to a certain VR platform. Authors have tested it on the concrete technical object, namely measurement device, output data is also given in the article.

Keywords: Virtual reality · UML · State chart · Diagram

1 Introduction

Nowadays most industrial enterprises [1–4] successfully use virtual reality technology (VR) to solve a variety of different tasks: personnel training and retraining on the workplace, modeling of assembled products, complex physical processes taking place inside products, and on the manufacturing line.

VR is used throughout the entire product lifecycle: from the early design stages, errors identification, ergonomics checking, to the production process organization, the development of maintenance, repair, and modernization processes [5].

In manufacturing, VR technology is widely used to simulate the installation of new equipment simultaneously with the manufacturing line design. In parallel, it becomes possible to train future operators of designed equipment on the VR prototypes. Thus, when opening the updated site, employees will be able to immediately get down to work and not waste time on training. Time saved can also come in handy when setting up factory lines. Production speed and the quality of the manufactured products play a leading role in the success of the enterprise, therefore it is so important to save time on parallel processes.

VR allows simulating emergencies [6]. Often, it is either dangerous or too expensive to reproduce such cases in reality, because the staff only knows in theory how to get

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12957, pp. 453–464, 2021.
https://doi.org/10.1007/978-3-030-87013-3_34

out of a difficult or emergency. Equipped with VR, staff can be prepared to handle any challenge.

Demonstrating the future product to customers even before production with the VR allows the client to see a life-size item and touch it.

It becomes evident that one of the keys to a successful VR software product is the design process and design solutions in form of documentation. The process should be as short as possible, and project solutions, in turn, must contain a detailed description of the developed object. In other words, project solutions should be clear for every member of the design team and adaptive under new requirements.

Complex technical systems, including those using VR technology, are designed using diagrammatic models, which are the artifacts of the visual graphic languages such as AADL, BPMN, SysML, UML, IDEF, Simulink, and others [7]. They significantly increase the efficiency and quality of design procedures by unifying the language of interaction between process participants including developers and client representatives, thorough documentation of design and architectural solutions, formal methods of checking diagrams for correctness. However, the larger the diagram and the more types of them are engaged the more complex system becomes. As a result, the visual system prototype is hard to understand and maintain.

Automata models (automata), or (finite) state machines are the most common means of representing systems at a high level of abstraction, which gives developers the most complete understanding of the software system principles. There is a wide range of reverse engineering techniques - getting a state machine from source code - to obtain concise and up-to-date documentation on how the system works.

UML 2.0 modeling language [8] is de facto the standard in object-oriented modeling and design. It is considered the most popular language for modeling reactive systems [9]. It is intended for designing, documenting, modeling the behavior of system artifacts [10, 11] and is widely used in technology and industry [12].

UML State Chart diagrams are of particular interest. They model the system behavior depending on external actions, or input signals, which are, in fact, analogous to state machine models.

Thus, combining UML State Chart diagrams and state machines to automatically synthesize virtual objects or *virtual mock-ups*, allows expecting reducing time and resource costs in this task. A large number of virtual mock-ups exists but there is no unified approach transforming descriptive models especially in above mentioned visual languages to the ones which can be applied in virtual reality.

This work is aimed to create an effective method for transforming a formal description of the UML State Chart diagram into a programmable model of an object that changes in time according to the principle of a state machine. The structure of the article is the following. Section 2 provides an overview of the work on the automated design of complex technical systems in virtual reality, as well as the design of systems using the UML State Chart. In Sect. 3 short UML review is given. The authors' models of the UML State Chart diagram and a virtual dynamic object, which is a VR representation of an automaton model, and events mapping function are described in Sect. 4. In Sect. 5 the method is given which converts a UML model into a VR one, which will be further used for program code generation. An example of converting is in Sect. 6.

2 Related Works

Authors investigated not only scientific and research works but also the software solutions supporting diagram-to-code transformation. There are few works containing not only applicable but also scientific results in generating virtual mock-ups models, therefore, firstly, we will consider software tools.

Visual Paradigm [13] is a CASE tool supporting UML, SysML, and BPMN languages developed by Visual Paradigm International Ltd. It has a good interface and supports various types of diagrams. Visual Paradigm provides reverse engineering procedures, demo online environment. Also, Visual Paradigm provides code generation from diagrams to C++, C#, Java. However, the tool has a proprietary license and in most cases generates a huge and bulky program code.

Software Ideas Modeler [14] is a smart CASE tool and diagram software that supports various diagram types. It is developed by Software Ideas company. Apart from UML, it works with ERD, BPMN, ArchiMate. Software Ideas Modeler has an interface similar to Microsoft products (Office and Visual Studio) consisting of tabbed ribbon with groups, toggled side panels. The tool supports the export of diagrams to more than 10 languages, and import from C#, C++, Java, etc. The free version has little functionality available which does not solve the problem described in the article. Also, the generated code contains a lot of extra classes and interfaces so it is hard to maintain.

Atmani et al. [15] generate virtual mock-ups of human bones in case of surgical operations. Authors compare several modeling methods and based on requirements they choose a simple-form method where objects are represented with quadrics, planes, etc. 14 parameters are involved in the virtual model synthesis, data is extracted from the radiological images. However, the method errors and in modeling. The method is also highly specialized.

Al-Fedaghi [16] describes a method that converts the UML model to the thinging machine (TM) model and back. Inspired by Heidegger's works, the author offers a specific all-in-one model which should solve problems of UML by internal cross-diagram integration. TM is built as a one-category ontology. TM consists of stages such as creation, process, release, input and output. The conversion process is based on [17] and the following five diagram types are involved: activity, use case, sequence, class and state ones.

3 UML Review

3.1 Shortly About UML

UML diagrams can be divided into structural and behavioral diagram groups [10]. Structural diagrams show the static structure of a system and its components at various levels of abstraction and implementation, as well as the relations between components. Behavioral diagrams simulate the principles of the system and its components in dynamics, thereby showing changes in the system at certain timestamps.

UML State Chart diagram is of particular interest in theory and practice. This diagram type simulates the dynamics of systems, highlighting the control flow following from one state to another. Essentially, a state chart is a special case of a state machine that models

behavior determining a sequence of states over the course of an object's existence. This sequence is considered to be the response to events and includes feedback to those events.

3.2 UML State Chart Diagrams

Consider main UML State Chart diagrams concepts [18]. Any UML state machine consists of regions, containing vertices and transitions between them. A vertex is either a state which may have a complex hierarchical structure of regions or a pseudo-state, regulating how transitions are compound in execution. Transitions have input and output states. They are executed with events (input signals or triggers), thus forming an area of possible state changes. The state is called simple if it has no regions; a complex state has two or more regions and is called orthogonal if it contains several regions visually divided dotted line. A region can contain simple states as well as complex ones. If the current state is not on the top level of the automaton, it should include at least one from the region. It is obvious that a complex state and states recursively layered into it form states tree.

In real automata, a designer works with parallel states; therefore, the transitions contained in them activate not only the target state but also indirect ones. The same can be said for deactivated states. On the contrary, a state can be activated/deactivated not only by a direct "near" transition but also by an indirect "far" transition.

4 Objects Formal Description

4.1 UML State Chart Model

UML State Chart diagram can be expressed via extended finite state machine (eFSM) with the following model [19–26]:

$$A = (S, X, S_0, C, TT, Fstart),$$

where:

- $S = S_{simple} \cup S_{complex}$ is states set, where:
  - $S_{simple} = s_{s1}, s_{s2}, \ldots, s_{sm}$ is simple (atomic) states set,
  - $S_{complex} = s_{c1}, s_{c2}, \ldots, s_{cn}$ is complex states set, $s_{ci} = \{R_{ij} | j \in \mathbb{N}\}$, where $R_{ij} \subset S \backslash s_{ci}$ is a region.
- X is input signals set,
- C is guard conditions set,
- S_0 is an initial state $S_0 \in S_{simple}$,
- $Fstart : R \rightarrow S_{simple}$ is initial state function,
- $TT = \{t_i | i \in \mathbb{N}\}$ is transitions table, $t_i = (src, dst, x, cond)$, $src, dst \in S, x \in X, cond \in C \cup \emptyset$.

Define the function $parent : S \rightarrow S$ which takes s_{child} as argument and returns state s_{parent} so that $\exists s_{parent}, r | r \in s_{parent} \cap s_{child} \in r$; if s_{child} has no parent state, the function returns A.

4.2 Virtual Mock-Up Model

Virtual environment consists of objects the user interacts with. Assume such objects as *virtual objects,* or *virtual mock-ups* and formalize them with the following view:

$$VO = (Ctrls, Events, F, CE, S, S_0),$$

where:

- *Ctrls* is controlers set,
- *Events* is events set,
- F is virtual object functions set, $F = \{f_i | f_i = (src_i, dst_i)\}$, $src_i \in S$, $dst_i \in S$,
- CE is a function mapping, $CE = \{ce_i | ce_i : Ctrls \times Events \rightarrow F\}$,
- $S = \{(Name, Values, CurrentValue)\}$ is states set, $CurrentValue \in Values$;
- S_0 is an initial state.

4.3 Mapping Functions

Introduce the *events mapping function* which maps each eFSM input signal from X to the couple of event type from *Events* and controller from *Ctrls*. It has one-to-one relation and thus the following formal description:$fEventsMap : X \rightarrow Events \times Ctrls$. The function is purposed to model human-computer interaction.

Also introduce the states map which links eFSM and virtual object states: $Map_S = \{(S_{Ai}, S_{VOj}),\ S_{Ai} \in A[S], S_{VOj} \in VO[S]\}$. Functions to find corresponding states are the following:$fgetSA(s_{VO}) = s_A | (s_A, s_{VO}) \in Map_S, fgetSVO(s_A) = s_{VO} | (s_{VO}, s_A) \in Map_S$.

5 Conversion Method

Method converting state chart model to the virtual dynamic object model is the following.

Copy All States

1. For each s in $A[S_{simple}]$:

   a. Insert s' in $VO[S]$ and (s, s') into Map_S.

2. For each s in $A[S_{complex}]$:

   a. Create empty state s'_c.
   b. For each r in s:

      (1) Create values set vs, initial state s_0.
      (2) For each s' in r:

      i. If $s' \in A[S_{simple}]$ go to step ii else go step v.
      ii. Insert s' into vs.

iii. If $s' = A[Fstart(r)]$, assign s' to s_0.
iv. Continue.
v. Go to step (2) and replace $A\left[S_{complex}\right]$ with s'.

(3) Insert (r, vs, s_0) into s'_c.

c. Insert s'_c in $VO[S]$ and $\left(s, s'\right)$ into Map_S.

Copy All Transitions

1. For each t in $A[TT]$:

a. Get $src = t[src]$, $dst = t[dst]$.
b. Get $srcExternal = P(parent(src) == A), dstExternal = P(parent(dst) == A)$.
c. If $srcExternal \wedge dstExternal$ insert $f = (fgetSVO(src), fgetSVO(dst), fEventsMap(t[x]))$ into $VO[F]$.
d. Else if $\neg srcExternal \wedge \neg dstExternal$:

(1) Create $src' = fgetSVO(src)$.
(2) Create $dst' = fgetSVO(dst)$.
(3) Update $src'[CurrentValue] = x \in src'[Values] | x \equiv src'[CurrentValue]$.
(4) Update $dst'[CurrentValue] = x \in dst'[Values] | x \equiv dst'[CurrentValue]$.
(5) Insert $f = \left(src', dst', fEventsMap(t[x])\right)$ to the F.

e. Else if $srcExternal \wedge \neg dstExternal$:

(1) Create $dst' = fgetSVO(dst)$.
(2) Update $dst'[CurrentValue] = x \in dst'[Values] | x \equiv dst'[CurrentValue]$.
(3) Insert $f = \left(src, dst', fEventsMap(t[x])\right)$ to the F.

f. Else if $\neg srcExternal \wedge dstExternal$:

(1) Create $src' = fgetSVO(src)$.
(2) Update $src'[CurrentValue] = x \in src'[Values] | x \equiv src'[CurrentValue]$.
(3) Insert $f = \left(src', dst, fEventsMap(t[x])\right)$ to the F.

6 Virtual Object Example

Consider the generation method on the (fragment) example of the oscilloscope S1-116 [27].

The UML state chart model of the device is structured below:

- Input signals: X = {powerOn, Polarity2Click, switchClick, VTClick, FClick, FLongClick, ScanA, ScanSettings},
- Guard conditions: C = {Fbutton}
- States: S = {Off, Introduction, WorkState = {ChannelI, ChannelII = {Norm, Inv}, View1, View2, View1 + 2, VTOff, VTQueue, VTDiscrete, ScanA, ScanAhighlight, ScanB, ScanMix}}
- Initial state: $S_0 = Off$

Transitions are visualized on the UML state chart diagram shown in the Fig. 1. Entities above are also shown on the diagram.

After the method application the VR model will be structured as shown below:

- Controlers: *Ctrls* = {Switch button, Scan view button, VT mode button, Change polarity button, Scan A settings button, Scan select button, Scan settings button, Channel I input, Channel I Y-scale switch, Channel II Y-scale switch, Channel II input},
- Event types: *Events* = {trigger click, trigger click and move, trigger double click, trigger click and hold},
- States: see "States" in the UML model structure above,
- Initial state: $S_0 = Off$.

Events mapping function for this device is given in the Table 1. 'Input signal' column stands for X set, 'Event type' – for *Events* set.

The virtual mock-up of the device is shown in Fig. 2. Controlers in visual form are given in the Table 2.

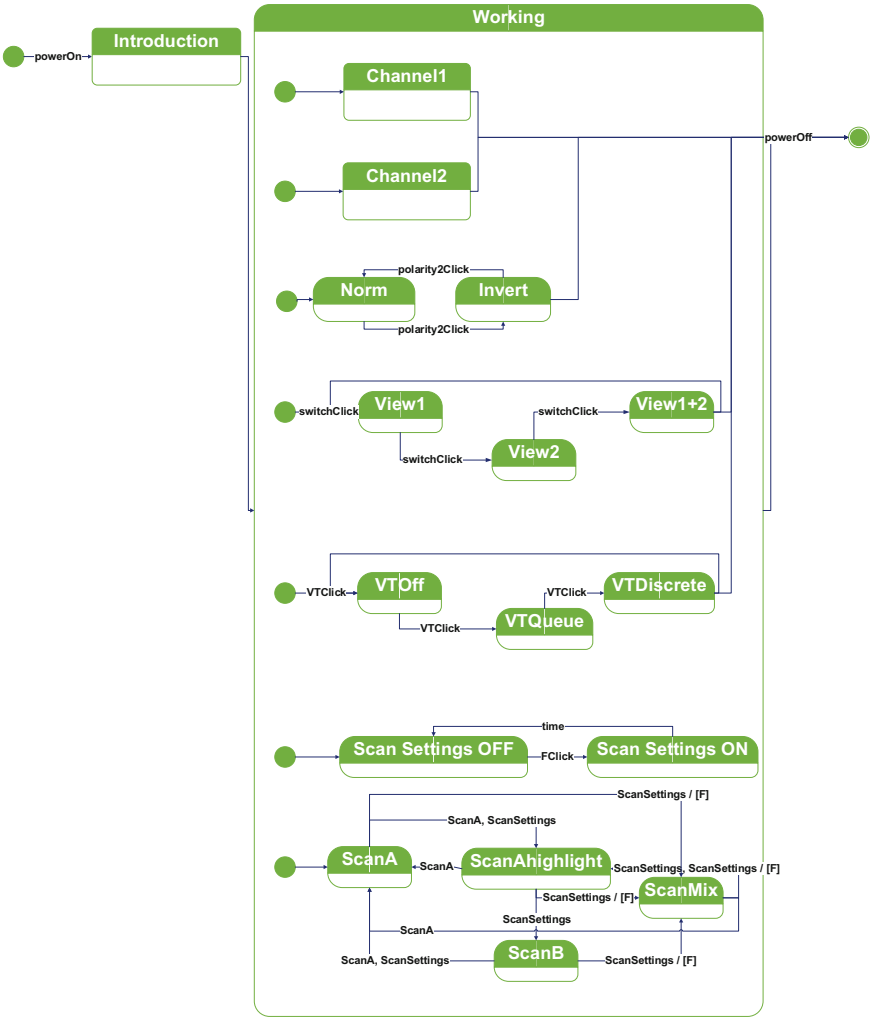


Fig. 1. S1–116 oscilloscope UML state chart diagram.

Table 1. Events mapping function

Input signal	Event type
powerOn	Trigger click
Polarity2Click	Trigger click
switchClick	Trigger click
VTClick	Trigger click
FClick	Trigger click
FLongClick	Trigger click and hold
ScanA	Trigger click
ScanSettings	Trigger click

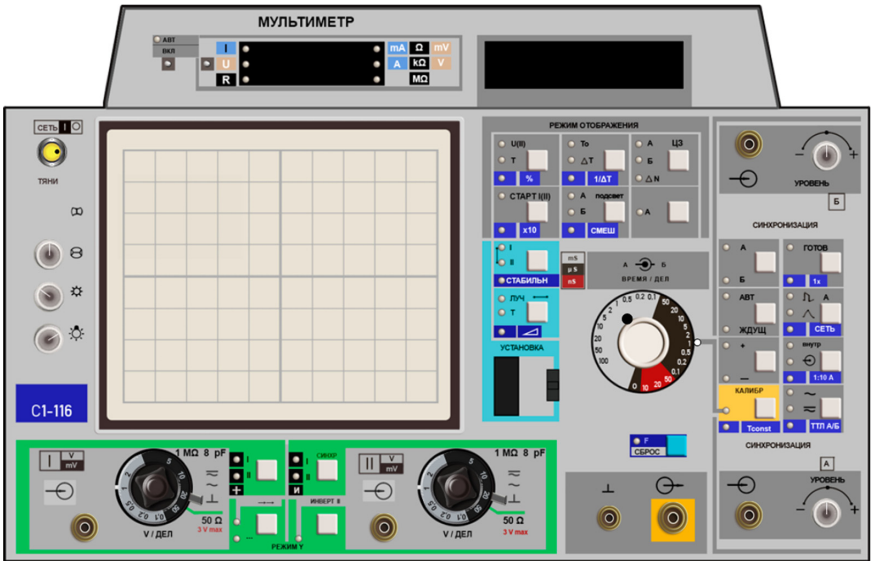
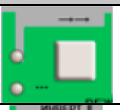
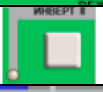
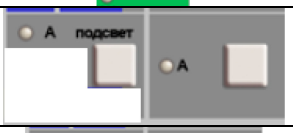
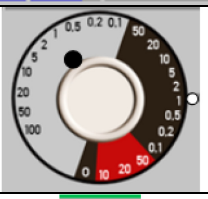


Fig. 2. S1-116 virtual mock-up

Table 2. Controllers of virtual mock-up

Graphical view	Name
	Switch button
	Scan view button
	VT mode button
	Change polarity button
	Scan A settings button
	Scan select button
	Scan settings button
	Channel I input
	Channel I Y-scale switch
	Channel II Y-scale switch
	Channel II input

7 Conclusion

Authors have developed method converting UML State Chart model to the virtual mock-up. It differs in the use of events mapping function forming human-computer interaction scheme. The method uses state chart model developed in accordance with UML 2.0. Generated virtual mockup reflects its behavior features depending on external triggers.

The method is hardware free and can be applied in generation of the program code modelling object behavior in VR.

The next research topics are the following: extending the method for the multi-object dynamic systems, development and analysis of virtual environment using other UML diagram models such as use case, sequence etc.

References

1. Bider, I., Jalali, A.: Agile business process development: why, how and when—applying Nonaka's theory of knowledge transformation to business process development. Inf. Syst. e-Bus. Manag. **14**(4), 693–731 (2014). https://doi.org/10.1007/s10257-014-0256-1
2. Sherehiy, B., Karwowski, W., Layer, J.K.: A review of enterprise agility: concepts, frameworks, and attributes. Int. J. Ind. Ergon. **37**, 445–460 (2007)
3. A global Swiss company offering advanced intelligent application software for multiple business sectors. http://whitestein.com/
4. Abidi, M.-A., Lyonnet, B., Chevaillier, P., Toscano, R., Baert, P.: Simulation of manufacturing processes via virtual reality. In: Virtual and Augmented Reality: Concepts, Methodologies, Tools, and Applications, pp. 918–953 (2018). https://doi.org/10.4018/978-1-5225-5469-1.ch044
5. Connect WIT, vol. 10, October 2019
6. Dozortsev, V.: Virtual reality technology in teaching operators of technological processes. Autom. Remote. Control. **6**, 42–50 (2018)
7. Liu, B., Zhang, Y.-R., et al.: A survey of model-driven techniques and tools for cyber-physical systems. Front. Inform. Technol. Electron. Eng. **21**(11), 1567–1590 (2020). https://doi.org/10.1631/FITEE.2000311
8. UML 2.0: https://www.omg.org/spec/UML/2.0/About-UML/
9. UML State Machine Diagrams. https://www.uml-diagrams.org/state-machine-diagrams.html. Accessed 10 Nov 2020
10. Maklaev, V.A, Sosnin, P.I.: Creation and usage of automatized design organization experience base. UlSTU, Ulyanovsk (2012)
11. Fowler, M.: UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd edn. Pearson Education, Inc., London (2004)
12. Binder, R.V.: Testing object-oriented software: a software testing verification reliability, **6**(3/4), 125–252 (1996)
13. Ideal Modeling & Diagramming Tool for Agile Team Collaboration: https://www.visual-paradigm.com/
14. Diagram CASE Tool for Software Modeling & Analysis - UML, BPMN, ERD. https://www.softwareideas.net/
15. Atmani, H., Merienne, F., Fofi, D., Trouilloud, P.: From medical data to simple virtual mock-up of scapulo-humeral joint. In: Proceedings of Eighth International Conference on Quality Control by Artificial Vision, vol. 6356. https://doi.org/10.1117/12.736712
16. Al-Fedaghi, S.: UML modeling to TM modeling and back. Int. J. Comput. Sci. Netw. Secur. **1**(21), 84–96 (2021)

17. Sommerville, I.: Software Engineering, 10th edn. Pearson Education Limited, Essex (2016)
18. About the Unified Modeling Language Specification Version 2.5.1. https://www.omg.org/spec/UML/2.5.1
19. Afanasyev, A., Voit, N., Ukhanova, M., Ionova, I., Epifanov, V.: Analysis of the design and technological workflow in a large radio engineering enterprise. Radioengineering **6**, 49–58 (2017)
20. Afanasyev, A., Voit, N., Kirillov S.: Development of RYT-grammar for analysis and control dynamic workflows. In: Proceedings of International Conference on Computing Networking and Informatics (ICCNI 2017), pp. 1–4. IEEE (2017)
21. Afanasyev, A., Voit, N.: Grammar-algebraic approach to the analysis and synthesis diagrammatically models of hybrid dynamic design workflows. Inf. Meas. Contr. Syst. **15**(12), 69–78 (2017)
22. Voit, N.: Development of timed RT-grammars for controlling processes in cyber-physical systems. In: Proceedings of the 13th International Conference on Interactive Systems: Problems of Human-Computer Interaction, pp. 87–91. UlSTU, Ulyanovsk (2017)
23. Afanasyev, A., Voit, N., Ukhanova, M.: Control and analysis of denotative and significant semantic errors diagrammatic models of design flows in designing automated systems. Radioengineering **6**, 84–92 (2018)
24. Afanasyev, A., Voit, N.: Grammar-algebraic approach to the analysis of hybrid dynamic design workflows. In: Maschenko, E. (ed.) Information Technologies and Information Security in Science, Technique and Education "INFOTECH-2017", pp. 43–48 (2017)
25. Kirillov, S., Voit, N., Molotov, R., Stepanov, A., Voevodin, E., Brigadnov, S.: Development and research of methods for analysis and control of semantic integrity and consistency of dynamic distributed workflows diagrammatic models based on temporary RV-grammarIn: 9th All-Russian school-Seminar for Graduate Students, Students and Young Scientists "Information Systems and CAD 2017", pp.135–139. UlSTU, Ulyanovsk (2017)
26. Voit, N.: Automated methods and software for designing of workflows. Inf. Meas. Contr. Syst. **16**(11), 84–89 (2018)
27. Bochkov, S.: Development of S1-116 oscilloscope virtual simulator. In: Tolok, A.V. (ed.) CAD/CAM/PDM–2017 Proceedings, pp. 249–251. Trapeznikov Institue of Control Sciences RAS, Moscow (2017)