2019 IEEE Conference on Information and Communication Technology (CICT)

Virtual Workplaces Testing Method on Accordance with the Technical Task

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

Semen Bochkov
Ulyanovsk State Technical University
Ulyanovsk, Russia
bochkovsi@ido.ulstu.ru

Sergey Kirillov
Ulyanovsk State Technical University
Ulyanovsk, Russia
kirillovsyu@gmail.com

***Abstract*—Virtual industrial computer simulators have significant advantages in training employees. At the same time, conformity of modelled workplaces with a real workplace organized in accordance with the technological process is of great importance. Traditional methods of software checking – manual and automatic unit testing – are not always able to fully cover simulator functionality and test it as soon as possible. Virtual simulators verification method is proposed. It is based on the structural and parametric analysis. To determine level of compliance between virtual and real workplaces the method uses data from the industrial product life cycle management (PLM) system.**

***Keywords*—virtual reality, software verification, virtual simulator**

I. Introduction

Design, implementation and introduction of industrial products assembly virtual simulators solves the problem of training and retraining personnel and specialists in a short time and improving the quality of the training on the job.

During the virtual simulators development [2] [8] [9], one should adhere to the technical task developed and agreed upon with the customer and closely related to the technological process of the modeled subject area, as well as the regulatory and technical documentation referred to by the technological process: product specifications, state and industry standards (GOSTs).

Adding, changing or skipping actions is allowed if and only if it does not violate the logic of the process. Such modifications include, for example, radio element illumination, automatic equipment setup (soldering station heating).

To check the virtual simulator for its adequacy to a real workplace, expert testing, automatic or semi-automatic tests (unit tests based on the "black box" method) are used. However, if the simulator has large functionality and a complex block diagram, manual testing and development of unit tests can take quite a lot of time, and the latter may not cover all the simulator's functionality.

For the fully automatized testing of such simulators, a verification method for a virtual workstation simulator is proposed, based on a structural-parametric analysis of the simulator contents and its compliance with an external data source using an ontology.

II. Software verification

There are the following verification method groups according to Kulyamin V.V. [11].

A. Expertise

This method is applicable to any software properties and life cycle artifacts at different stages of a project. At the same time, different types of expertise can be used for different purposes: organizational, technical, cross-cutting, inspection, audit. It allows identifying almost any kind of error, including at the stage of preparing the corresponding artifact, thereby minimizing the lifetime of the defect and its consequences for the quality of derived artifacts. The lack of expertise is the impossibility of its automation and implementation without the active participation of people.

B. Static Analysis

These methods check formalized rules for the correct construction of these artifacts and to search for common defects according to some patterns. Such an analysis is well automated and can be almost completely assigned to tools that are quite convenient to use and do not require special training. Most of the techniques for static verification of program correctness sooner or later become part of compilers or even translate into semantic rules of programming languages.

Static analysis is applicable only to code or certain project artifacts presentation formats. It can detect only a limited set of error types. Also, the following dilemma is characteristic: either strict analysis methods are used that do not allow errors to be missed (those types that are searched for), but lead to a large number of messages about possible errors that are not, or a set of error messages is accurate, but some of them are to be skipped.

C. Formal methods

These methods use formal requirement models, software behavior and its environment models for the properties analysis. It is performed using specific techniques such as deductive analysis, model checking or abstract interpretation.

Formal verification methods are capable of detecting complex errors that cannot be detected by examinations or testing, are actively used in a number of areas where the consequences of errors can be extremely expensive.

Formal methods are applicable only to those properties that are formally expressed in the framework of some adequate mathematical model, as well as to those artifacts for which an adequate formal model can be constructed.

D. Dynamic methods

Dynamic methods use the results of the actual work of the program or its model in the properties analysis and evaluation. These are testing, monitoring, profiling.

To apply dynamic methods, a working system or the required system components at least at the prototype level must exist. The backward of dynamic methods is that they detect in the software only runtime errors, and, for example, convenience or maintenance defects cannot be detected. However, using dynamic methods helps to control the software characteristics and track errors in a real environment, which cannot be accurately investigated using other approaches.

E. Synthetic methods

Currently, there are research works and tools using several verification methods listed above. Dynamic methods using formal elements were distinguished in separate areas such as models-based testing [12] and monitoring formal properties. Test building tools make significant use of both formalization of some software properties and static code analysis. The general idea of synthetic methods is a combination of the advantages of the basic approaches to verification, minimizing their shortcomings.

Comparison of methods is shown on the Table I.

TABLE I. VERIFICATION METHODS COMPARISON

Methods group	**Ability to automatize**	**Concerned software development steps**	**Threshold**	**Applicable to the current problem?**
Expertise	Very low	Any	High	+/-
Static analysis	High	Source code	Middle	-
Formal methods	High	Requirements, behavior	High	+
Dynamic methods	High	Run-time testing	Low	+
Synthetic methods	Above average	Any, mostly some and simultaneously	Middle	+

From the table results it follows that the developed verification method should belong to the synthetic methods group combining features of formal and dynamic methods.

III. RELATED WORKS

There are many means of formal verification of programs [14], the most famous are the following.

CONC2SEQ plugin is written for the FRAMA-C platform for verifying parallel C programs [13] [15]. CONC2SEQ converts the source parallel code to serial for correct operation in FRAMA-C, while user specifications are automatically integrated into the new code.

The disadvantage of the plugin is the lack of support for value analysis and dynamic verification.

Lazy-Cseq is a tool that allows translating a multi-threaded C program into a non-deterministic sequential code that preserves reachability for all cyclic traces with a given number of iterations [16] [17]. It reuses existing high-performance bounded model checking (BMC) tools. The translation is carefully designed so that it consumes as little memory and non-determinism sources as possible, it is also organized within the framework of strict SAT / SMT formulas. Lazy-Cseq contains a script linking translation and verification.

Lazy-Cseq accepts a C program using POSIX streams as input; verification is done using Lazy sequentialization, working both with partial store order (PSO) and total store order (TSO) memory.

Java Pathfinder was originally created as a means of checking a model with an explicit state for Java bytecode [18]. In this way, it can test the model of programs written in languages oriented to the Java virtual machine. The initial implementation approach was to convert Java bytecode to Promela for analysis using the Spin model validator [19]. Currently, Java Pathfinder checks Java bytecode directly.

Java PathFinder is used to analyze parallel programs and abstract models, including applications on the Android platform [21].

The plugin architecture allows the user to expand the type of properties being checked. They are implemented in the form of user code with a listener model for checking user properties when examining a state space. However, the state space exploration algorithm does not take into account the type of custom properties and cannot configure the search to be effective for these properties. In its purest form, Java PathFinder looks for deadlocks, statement errors, and null pointers.

The disadvantages include the limited size of the programs that can be analyzed, approximately 10 thousand lines of code. Due to these limitations, it is used to analyze small libraries and program fragments.

Malpas (MALvern Program Analysis Suite) is a toolset for software analysis [20], which allows performing static analysis and program compliance with specifications. The analyzed program should be written in the Malpas intermediate language (MIL), while there are automatic translators from common programming languages and assembler to MIL.

Static analysis tools allow removing code metrics, analyzing the program for dead code, uninitialized data, unexpected dependencies, etc. Formal analysis tools verify the mathematical compliance of the program specifications, including pre- and postconditions, invariants, statements.

IV. TEST METHOD

The virtual workplace model has the following form:

Workstation simulator = (assembly objects set; consumables set; tools set; documents set; target assembly object; technological process), where:

Assembly object = (key-value pairs set),

Consumable = (key-value pairs set),

Document = (name; assembly object or consumable; key-value pairs set), *Tool = (key-value pairs set).*

The *target assembly object* is the state of the assembly object(s) characterizing completed assembly of a particular product.

Technological process = (main operations, intermediate operations), where:

Main operation: assembly objects × materials × tools × documents → assembly objects × materials × tools – an operation changing the state of assembly object(s).

Intermediate operations = (tool activation / deactivation; object activation / deactivation; material activation / deactivation), where:

Tool activation / deactivation: tool → tool – active tool changing operation,

Object activation / deactivation: assembly object → assembly object – active assembly object changing operation,

Material activation / deactivation: consumable → consumable – active consumable changing operation.

The data source for testing on accordance with the technical task is the industrial product life cycle management system (PLM) [10]. Figure 1 shows the data organization in PLM formed as a classification ontology.

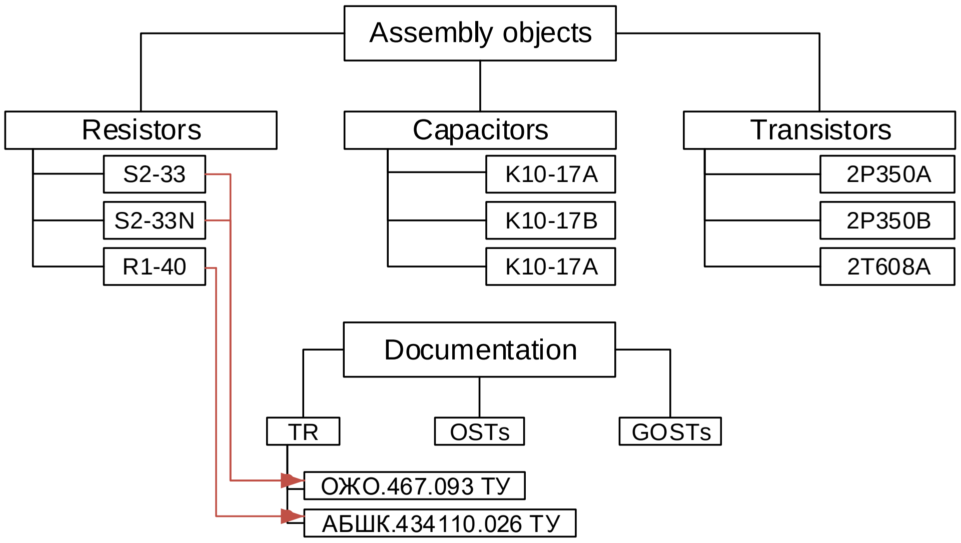

Fig 1. Ontological data model in PLM

Figure 2 presents diagrams explaining the operation of the method of structural-parametric verification of a virtual workplace (VWP).

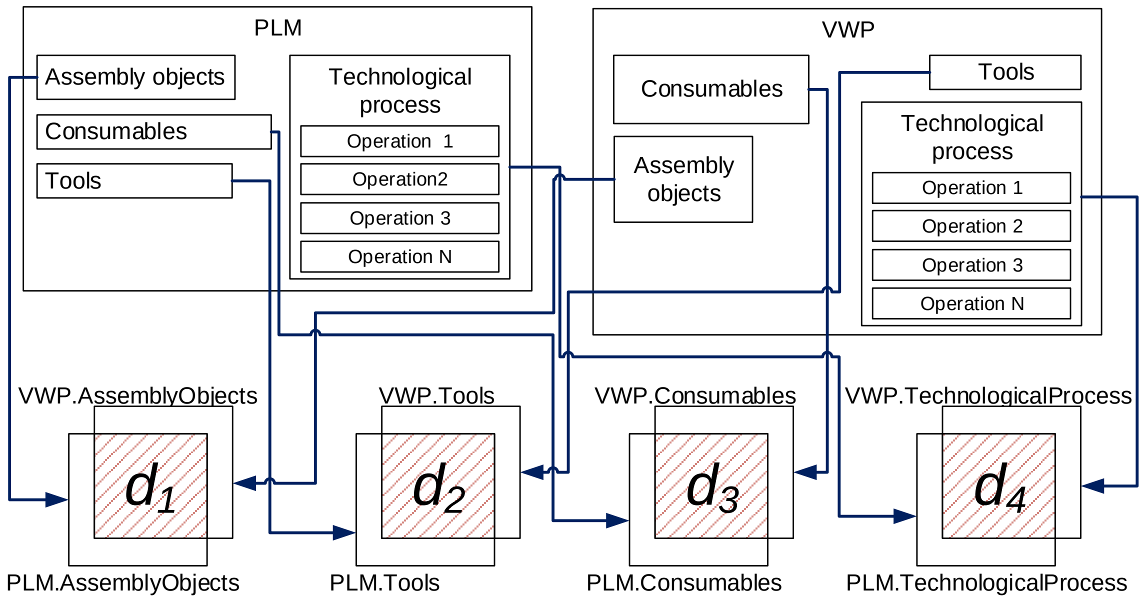

Fig 2. Virtual workplace (VWP) verification process block diagram

The verification process includes the following subprocesses:

- correspondence degree of assembly objects in VWP to assembly objects specified in the specification ($d_1 = \frac{AO_V}{AO_{PLM}}$, where AO_V is the VWP assembly objects count specified in the specification from PLM, AO_{PLM} is the assembly objects count specified in the specification in PLM) ;
- correspondence degree of tools in VWP to tools specified in the specification ($d_2 = \frac{T_V}{T_{PLM}}$, where T_V is the VWP tools count specified in the specification from PLM, T_{PLM} is the tools count specified in the specification in PLM);
- correspondence degree of consumables in VWP to consumables specified in the specification ($d_3 = \frac{CM_V}{CM_{PLM}}$ where CM_V is the VWP consumables count specified in the specification in PLM, CM_{PLM} is the consumables count specified in the specification in PLM);
- correspondence degree of the technological process operations content and ordering in VWP to the technological process stored in PLM ($d_4 = \frac{O_V}{O_{PLM}}$, where O_V is the technological process steps count embedded in the VWP, which, starting from the first, correspond to the technological process steps indicated in the documentation in PLM, O_{PLM} is the technical process steps count in the documentation in PLM).

The result of VWP testing is the correspondence degree value D equal to the minimum of the subprocesses execution results, i.e. $D = min\,(d_1, d_2, d_3, d_4)$.

V. Conclusion

Due to the impossibility of manual expert testing of a virtual workplace simulator or automatic unit testing covering all the simulator's functional capabilities and the lack of suitable tools, new verification method based on sources governing the real workflow process is developed.

The method allows automatically verify virtual workplace in such aspects as count and nomenclature of the objects set including assembly objects, tools and consumables. It also verifies technical process whether it matches real one and has the same order like in the manufacturing.

For this method, a virtual workplace model is proposed that intensively uses the “assembly object”, “tool”, and “consumable material” concepts.

The developed method is universal and suitable for virtual workplaces regardless of the working position subject area.

Acknowledgments

The reported research was funded by the Ministry of Education and Science of the Russian Federation, project No. 2.1615.2017/4.6.

The reported research was funded by Russian Foundation for Basic Research and the government of the region of the Russian Federation, grant № 17-07-01417.

The reported research was funded by Russian Foundation for Basic Research and the government of the region of the Russian Federation, grant № 18-47-730032.

References

[1] M. H. Abidi, A. Al-Ahmari, A. Ahmad et al. “Assessment of virtual reality-based manufacturing assembly training system”. Int J Adv Manuf Technol, pp. 1-17, May 2019. DOI=https://doi.org/10.1007/s00170-019-03801-3

[2] A. Afanasyev, T. Afanasyeva, S. Bochkov, and N. Voit. “Application of virtual reality technology in the learning process”. In Proceedings of 10th annual International Conference on Education and New Learning Technologies (Barcelona, Spain, July 02 – 04, 2018). EDULEARN18. IATED, Palma, Spain, 10220-10225. DOI=https://doi:10.21125/edulearn.2018.2485

[3] A. M. Al-Ahmari, M. H. Abidi, A. Ahmad, and S. Darmoul. “Development of a virtual manufacturing assembly simulation system”. Advances in Mechanical Engineering, 8, 2, March 2016, pp. 1-13. DOI=https://doi.org/10.1177/1687814016639824

[4] C. Trakunsaranakom, S. Butdee, F. Nöel, and P. Marin. “Product design review in a virtual reality environment”. KMUTNB Int J Appl Sci Technol, 11, 2, April-June 2018, pp. 137-149. DOI=https://doi.org/10.14416/j.ijast.2018.04.004

[5] A. Neumayr, and M. Otter. “Algorithms for Component-Based 3D Modeling”. In Proceedings of the 13th International Modelica Conference (Regensburg, Germany, 04 - 06 March 2019). IMC’19. Linköping University Electronic Press, Linköpings Universität. 13th International Modelica Conference, 383-392. DOI=https:///doi.org/10.3384/ecp19157383

[6] F. Noël, and D. Dori. “Towards 3D Visualization Metaphors for Better PLM Perception”. In: PLM 2015: Product Lifecycle Management in the Era of Internet of Things, A. Bouras, B. Eynard, S. Foufou, K.D. Thoben, Eds. IFIP Advances in Information and Communication Technology, vol 467. Springer, Cham, pp. 461-475, 2016. DOI=https://doi.org/10.1007/978-3-319-33111-9_42

[7] C. Trakunsaranakom, F. Noël, and P. Marin. “Assessment of Virtual Reality Environments for design activities”. In EvroVR2014 - Conference and Exhibition of the European Association of Virtual and Augmented Reality, J. Perret, V. Basso et al. Eds. The Eurographics Association, pp. 31-37, 2014. DOI=http://dx.doi.org/10.2312/eurovr.20141336

[8] S. Bochkov. “Virtual radio technical workplaces development on the example of wiring harness assembly”. In Proceedings of 10th All-Russian School-Seminar of Postgraduates, Students and Young Scientists "Information Systems, Computer-Aided Design" (ISCAD-2018). (Ulyanovsk, Russia, November 27 – 28, 2018). ISCAD ’18. UlSTU, Ulyanovsk, Russia, pp. 67-78.

[9] N. Voit, D. Kanev, S. Bochkov, and M. Ukhanova. “Development of Trainee Actions Evaluation Software in Virtual Environment Based on Expert System”. In: Proceedings of 5th International Conference “E-Learning in Continuing Education-2018”. (Ulyanovsk, Russia, April 18 -20, 2018). EONO ’18. UlSTU, Ulyanovsk, Russia, pp. 151-159.

[10] M. Ukhanova. “Ontology-Based Organizational and Technical Components Semantic Model Development”. Information and measuring and control systems, 1, 11, pp. 98-108, November 2018. DOI=https://doi.org/10.18127/j20700814-201811-16

[11] Kulyamin, V.V. Software verification methods. Ivannikov Institute for System Programming of the RAS, Moscow, 2008.

[12] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner. Model Based Testing of Reactive Systems. LNCS 3472, Springer-Verlag Berlin, Heidelberg, 2005.

[13] Frama-C: https://frama-c.com/index.html

[14] R. C. Armstrong, R. J. Punnoose, M. H. Wong, J. R. Mayo. Survey of Existing Tools for Formal Verification. Technical Report. Sandia National Laboratories, 2014.

[15] A. Blanchard, N. Kosmatov, M. Lemerre, and F. Loulergue. “Conc2Seq: A Frama-C Plugin for Verification of Parallel Compositions of C Programs”. In 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM) (Raleigh, NC, USA , October 02-03 2016). SCAM ’16. IEEE, Los Alamitos, CA, pp. 767-772, 2016. DOI=https://doi.org/10.1109/SCAM.2016.18

[16] E. Tomasco, T. L. Nguyen, O. Inverso, B. Fischer, S. La Torre, and G. Parlato. “Lazy sequentialization for TSO and PSO via shared memory abstractions”. In Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design. (Mountain View, California, October 03 - 06, 2016). FMCAD ’16. FMCAD Inc, Austin, TX, pp. 193-200, 2016. DOI= https://doi.org/10.1109/FMCAD.2016.7886679

[17] C Sequentialisation Tool for Software Verification: https://www.southampton.ac.uk/~gp1y10/cseq/

[18] K. Havelund, and T. Pressburger. “Model checking JAVA programs using JAVA PathFinder”. International Journal on Software Tools for Technology Transfer, 2, 4, pp. 366-381, March 2000. DOI=https://doi.org/10.1007/s100090050043

[19] K. Havelund. Java PathFinder: A Translator from Java to Promela. In Proceedings of the 5th and 6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking, vol. 1860. (London, UK, August 27, 1999). SPIN ’99. Springer-Verlag Berlin, Heidelberg, pp. 152-152, 1999. DOI=https://doi.org/10.1007/3-540-48234-2_11

[20] MALPAS: http://malpas-global.com/

[21] A. Kohan, Y. Mitsuharu, C. Artho, Y. Yoriyuki, M. Lei, H. Masami, and T. Yoshinori. “Java Pathfinder on Android Devices”. ACM SIGSOFT Software Engineering Notes, 41, 6, pp. 1-5, November 2016. DOI=https://doi.org:10.1145/3011286.3011292