Verification and testing automation of UML projects

Nikita Voinov, Vsevolod Kotlyarov
Saint-Petersburg State Polytechnic University, Saint-Petersburg, Russia
voinov@ics2.ecd.spbstu.ru, vpk@ics2.ecd.spbstu.ru

***Abstract* – This paper presents an integrated approach to verification and testing automation of UML projects. It consists of automatic model creation from UML specifications in the formal language of basic protocols, model's verification by the means of VRS technology and automatic tests generation in TTCN language using TAT. The actuality of this task arises from necessity of software functionality's correctness checking, including verification and testing, but there is lack of industrial technologies which allow integrating these two activities. Results of the developed approach piloting are also described.**

I. INTRODUCTION

Documents describing software requirements can contain a large amount of miscellaneous types of errors, the main of which is difference between requirements adopted and detailed by technical specialist and customer's initial requirements. Enormous help in solving this problem is brought by using of formal languages and notations for requirements development. These notations allow customer and developer programmer to speak one language and lead to extermination of specifications' ambiguity of different types. Formal notations and formal methods of software quality control are often used in software development practices. The most popular graphical formal language is UML (Unified Modeling Language) [1]. This standard contains a wide range of graphical objects and allows creating system description from different points of view.

One of the main tasks the software developers have to solve is software functionality's correctness checking, which starts from development of software requirements and lasts until software withdrawal. This causes high demands to completeness and productivity of this checking and leads to appearance of new technologies and program instruments for automation of software functionality's correctness checking, which includes verification and testing.

Although there are a lot of miscellaneous instruments of verification (Spin (BellLabs laboratory), SCR (NavalResearch laboratory), VRS (ISS organization), etc. (see [2, 3] for review)) and testing automation (Rational Rose (IBM), TAT (Motorola), Together (Borland), etc. (see [4] for review)), two serious problems can be stated. The first one is lack of industrial technologies which allow integrating testing and verification. This is especially important when a huge amount of system's behavioral scenarios have to be verified in order to guarantee its correctness and this have to be done in limited time. Secondly, verification based on model checking (when a model of the system is created and requirements for every possible model's state are checked) uses some formal language to create a model of the system and the process of model creation from formal specifications is quite long and laborious.

This paper outlines the main principles of verification and testing automation of UML projects, including automatic model creation from UML specifications in the formal language of basic protocols, model's verification by the means of VRS technology and automatic tests generation in TTCN [5] language using TAT (Test Automation Toolset) [6]. Results of the approach's piloting on large telecommunication program project are also presented.

II. APPLIED TECHNOLOGIES

A. Basic Protocols

Basic protocol is a formal representation of an assertion about some actions that have to be applied in a program or algorithm under some conditions. In a general case a basic protocol is a Hoare's triplet [7] in the following notation:

$$\alpha \xrightarrow{\mu} \beta$$

where α and β are the pre-condition and the post-condition respectively and μ – is the process part of the basic protocol. Both α and β conditions are specified by logic formulas of the basic protocols language (a variant of the first-order logic) which can be evaluated for any state of the system.

Basic protocols can be consistently concatenated through their pre- and post-conditions – if the state specified by the post-condition of one basic protocol is to the same as the one specified by the pre-condition of the next basic protocol (actually, the pre-condition formula of the successor should be derivable from the post-condition formula of the predecessor). All such possible concatenations construct the model's behavior graph to be processed by verifier.

The language of basic protocols has an MSC-type syntax [8] and basic protocols can be presented in two ways: textual and graphical (MSC/PR and MSC/GR) [9].

B. VRS Technology

This technology is capable to verify models represented with basic protocols, from small to huge ones. As a result, various incidents of non-deterministic behavior, unreachability of specified system states, or deadlocks are detected. If no such defects are found, the system model is formally proved to be complete and consistent within the specified constraints.

Automated verification of software systems with VRS technology implies the functional requirements, which were used for system implementation, and system's model in the form of basic protocols created from the source code, formal specifications, etc. The technology checks that the model

meets the system requirements. This means that the software system satisfies them as well.

For verification process an ordered list of signals or basic protocols that contain required events (actions, signals, etc.) should be specified. VRS can check that for this model the behavior graph contains paths which include the specified sequences in the specified order. The existence of such paths (traces) is a proof of correctness of the model behavior with respect to this criterion. Search of such traces is realized by looking for respective signal interaction between agents or by looking for the specified basic protocol names in the generated traces and considering their actual ordering.

Thus, a trace is a scenario of a possible model behavior. Since the model was derived from an actual implementation of a program system, we can say, that a trace is a scenario of an actual system behavior. Scenarios are represented as consistent concatenations of relevant basic protocols into one chain. VRS outputs traces in the MSC/PR view.

Results of verification are automatically summarized in a verification report, which describes all found inconsistencies, discrepancies, deadlocks, and other errors in the model. Traces demonstrating the incorrect model behaviors are attached to the report. They are used to identify the root causes of such incidents.

Traces generated by VRS can be used for automated creation of an exhaustive test suite for the program system. The TAT (Test Automation Toolset) tool is used for automated test generation from those traces along with the respective testing environment and subsequent test runs.

C. TAT (TEST AUTOMATION TOOLSET)

A key to make testing technologies cheaper and more efficient lies in the area of test generation techniques, i.e. efficient and compact description of test sets and thus significantly reduces tester's manual efforts to develop them. Another key is visualization of formal description by means of graphs. These problems are solved by testing automation tool – test generator TAT.

TAT is a joint toolset, which provides complete, fully automated testing cycle based on user-defined scenarios developed in formal language MSC - Message Sequence Charts.

TAT encompasses several tools that offer complete set of solutions for efficient specification analysis and test generation.

In addition to standardized MSCs, TAT supports extended MSC notations, enhanced with macros, allowing significant reduction of code size developed manually. This extended notation allows absolute, relative, and more complex time specifications in test scenarios. With such framework, TAT helps to get significant time and cost savings through reuse and efficient workaround of time and macro definitions.

III. MAIN STEPS OF VERIFICATION AND TESTING AUTOMATION

A. Autoformalization of UML Specifications

Before using VRS for verification purposes, system's requirements shall be described in the language of basic protocols. Manual creation of such description is laborious process and can be compared with manual development of test cases in this regard. In some situations the process of formalization can be hastened by automatic creation of basic protocols from initial specifications. This approach is called "autoformalization".

In those UML projects, where system's specification is presented as a state machine, respective set of basic protocols can be generated automatically by using special tool – uml2bp, which in fact is a module of VRS. This module generates sets of basic protocols for every state machine in Telelogic Tau G2 project with .ttp extension as well as creates files with environment and events description, needed for VRS project. All generated files can be imported to VRS for verification.

Example of generated basic protocol is shown in Fig.1.

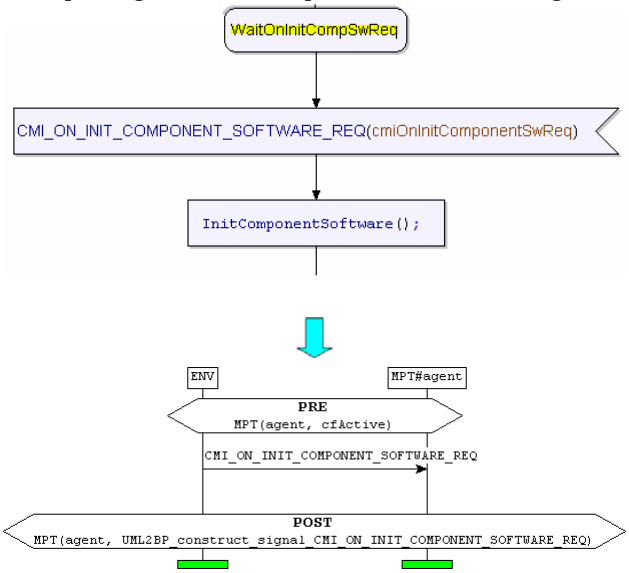

Fig. 1. Converting of a piece of UML state machine diagram into a basic protocol

B. Vefification with VRS

Basic Approach to Verification with VRS

The procedure to check a requirement – is a direct formulation of a sequence of observable causes and results of some activities; after analyzing this sequence a conclusion can be derived whether the requirement is satisfied or not. Such procedure may be used as a criterion for meeting this particular requirement. Basically, the criterion procedure is a method for checking satisfiability of a requirement. Term "chain" can be used for the criterion procedure.

After identifying in the behavioral scenario (hypothetical one or implemented in a real system or system model) the fact that such criterion is satisfied, one can state that the respective requirement in the system being analyzed is satisfied as well.

A procedure for checking requirements (a chain) is specified through formulating all its elements: initial conditions (causes) required for performing a certain activity, the activity itself and observed results of performing the respective activity.

In particular cases to describe the causes and results the states of variables (in form of their values or constraints for tolerance range) may be used. These variables are employed by the activity to track the state changes. In case of non-determinism, possible variants of state changes are tracked. A direct transition from one state to another with a void activity is also possible.

All the above refers to constructing a use-case for a particular requirement. So, a chain or use-case with sequences of activities and states may serve as a criterion for satisfiability of a requirement, sufficient to demonstrate it. Non-determinism in formulations is also covered with a number of chains or use-cases.

Realization of verification stage

Step 1. Formulation of filters and heuristics for the current project (after set of basic protocols as well as files with environment and events descriptions have been generated). They help to decrease number of traces by pointing the trace generator to the definite direction.

Step 2. Performing an automatic trace generation cycle.

Step 3. Analysis of findings with deadlocks, inconsistency and other issues, which prevent the tool from completing trace generation in the Goal or Restricted states. Fixing problems identified in findings and their review with the developers. Based on the review results correcting the generated basic protocols or initial requirements. Repeat steps 1-3.

Step 4. Analysis of generated traces with a script to check whether the coverage criteria are satisfied. Repeat steps 1-4 if needed.

Among traces generated by VRS user can choose those which should be used for automatic tests generation with TAT.

Manual creation of VRS traces

Another useful feature supported by VRS is manual creation of traces with user defined order of basic protocols (on each step of trace generation user himself chooses the protocol from the list of protocols, which can be applied now). This helps to cover concrete scenario of model's behavior on all possible levels of abstraction. Described below is example of this feature.

Sample UML state machine diagram is shown in Fig.2.

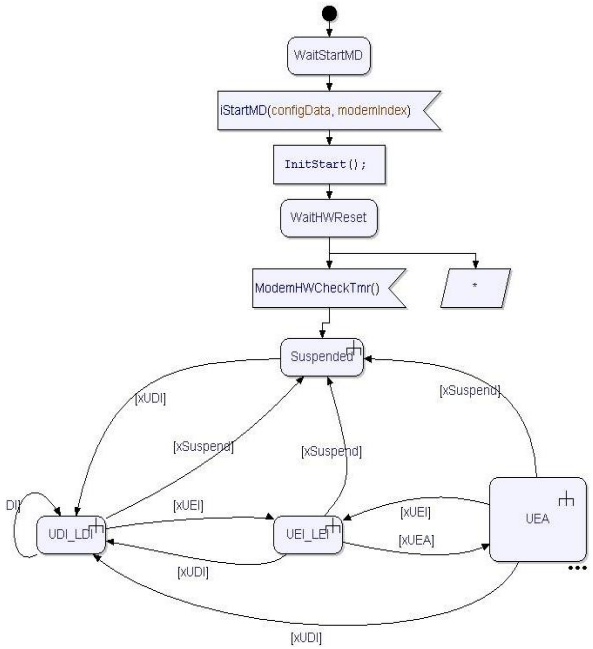

Fig. 2. Sample UML state machine diagram

Its main part is four composite states (Suspended, UDI_LDI, UEI_LEI, UEA) and transitions between them. Each composite state is in turn a state machine itself with complicated behavior inside.

Uml2bp module converts the whole state machine into the set of basic protocols. They are imported to VRS for model's analysis.

Basing on the set of basic protocols, a graph of the model can be constructed. Fig.3 shows the graph without detailed behavior of four composite states.

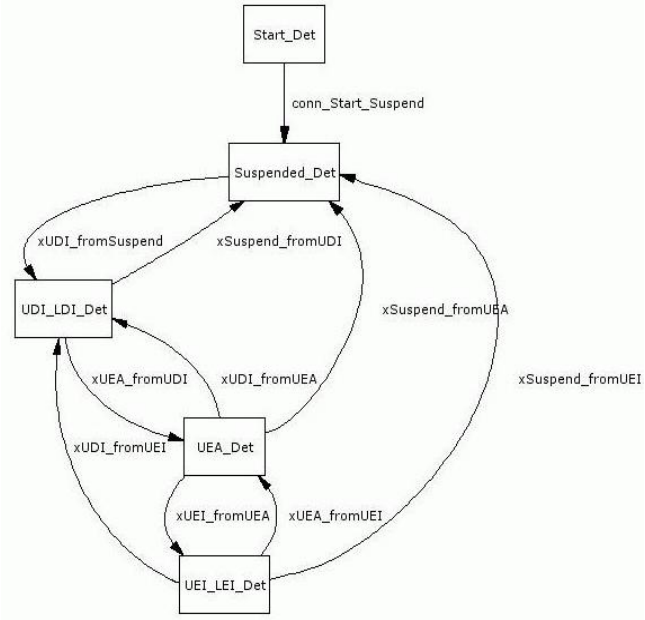

Fig. 3. Graph with composite states

It is also possible to expand any composite state to examine its detailed behavior. Fig.4 shows the same graph but with detailed behavior of Suspended state.

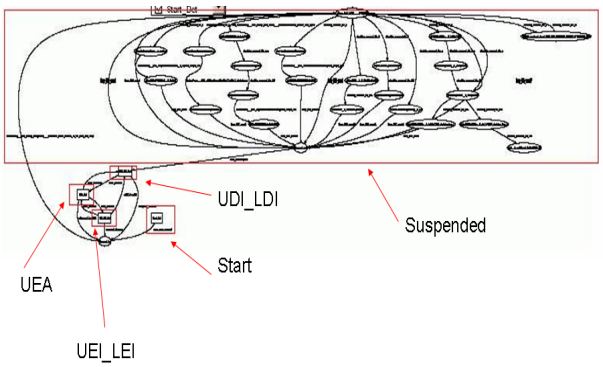

Fig. 4. Graph with detailed behavior of Suspended state

Detailed behavior is represented by a number of states (ovals) with transitions between them. Each transition is performed by a basic protocol. So, in accordance with the graph, one can construct his own trace in interactive mode of trace generation with any abstraction level (high level or detailed) for all composite states and for the whole initial state machine. Traces covering high level behavior of initial state machine and one of possible scenarios of Suspended state's detailed behavior are presented in Fig.5 and Fig.6 respectively.

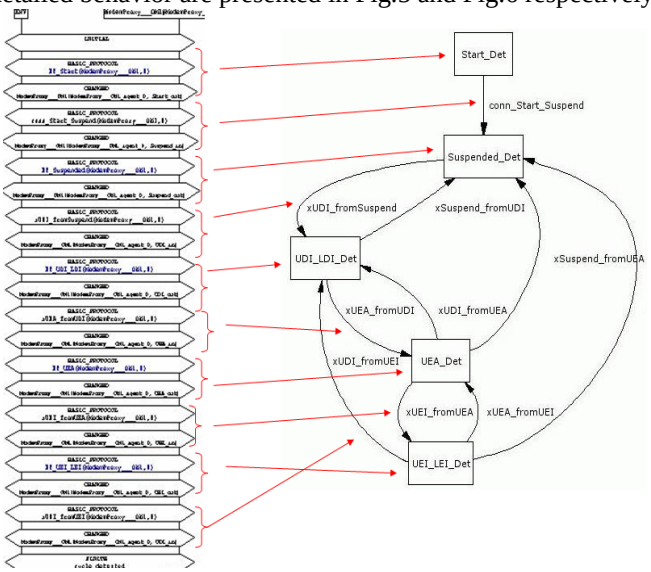

Fig. 5. Sample trace of high level behavior of initial state machine

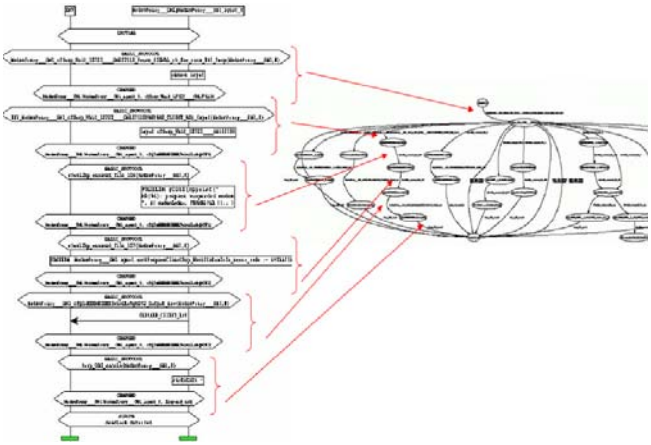

Fig. 6. Sample trace of Suspended state behavior

Now these two traces can be merged: the second trace can be glued in to the respective part of the first one, where Suspended state is covered as composite state. Traces merging is presented in Fig.7.

Fig. 7. Traces merging

Traces created in interactive mode can also be used for automatic tests generation.

C. Automatic Tests Generation with TAT

The approach described below is aimed at automatic tests generation on standard language of telecommunication applications testing – TTCN (Testing and Test Control Notation). The proposed approach allows test engineers to exclude manual development and focus on test scenarios, which hastens testing and bugs detection process.

Tests generation process is supported by number of scripts and templates of automatic generation of result TTCN-files. Scripts and templates use auxiliary files with data types description, signals templates description, configuration description, etc.

Tests generation is based on using two input files: a trace (scenario) with signals and their parameters and .xls file, which contains values for parameters used in this trace.

Overall scheme of the process is shown in Fig.8.

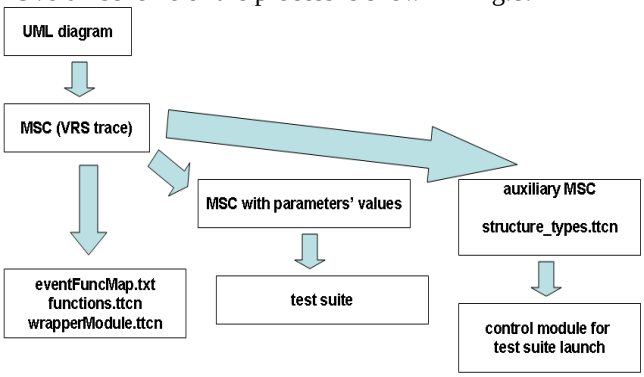

Fig. 8. Automatic tests generation scheme

Several steps can be listed in generation process:

- Automatic generation of functions description, which provide access to the system under test according to signals in the diagram. File with functions description

is imported to test project. As a result, only functions calls will be used in test scenario without their bodies.

- Automatic assignment of values to signals parameters in the diagram. Values are taken from .xls file. If one signal is presented several times in the diagram and values of its parameters change, it also should be mentioned in .xls file.
- Generation of test suite from MSC file with assigned parameters values. TAT's template is used on this step.
- Generation of auxiliary TTCN-file, which performs test suite execution.

After that all generated files are imported to test project. As project is compiled and built, test suite can be executed in automatic mode. Test results are saved in log-file.

IV. RESULTS OF PILOTING

The described approach to verification and testing automation was performed on one of the modules of telecommunication project of wireless network. Estimated size of the whole network's model is about 50000 basic protocols.

About 4000 basic protocols were generated on the stage of autoformalization of the module under test, which took three minutes of uml2bp work. Concerning the fact that estimation for manual creation of basic protocols is 10-50 per day, time saving is obvious and considerable.

Model's verification with VRS discovered about 100 findings. 12 of them were considered to be defects and fixed in future versions of the product.

Automatic tests generation also takes just several minutes, which is much less than manual development. Even considering the efforts required on creation of UML diagram and .xls file for concrete test run adjustment, time savings are estimated as several hours. Besides, when initial requirements are changed or new ones are added, it is enough just to modify initial UML diagrams and spend several minutes on another cycle of tests generation instead of long process of manual correcting the test code with possibility to introduce new errors. Requirements changes during the project realization happen very often, which is caused mainly by large size of projects. This makes the described feature more actual.

V. CONCLUSION

The developed approach to verification and testing automation proved its advantages in a large telecommunication project and can be further reused in other projects based on UML specifications. As this development process is one of the most preferable nowadays, the field of this approach application enlarges.

REFERENCES

[1] UML Distilled Second Edition. A Brief Guide to the Standard Object Modeling Language.

[2] Visser W., Havelund K., Brat G., Park S., and Lerda F. Model checking programs. Automated Software Engineering Journal, 10(2), April 2003.

[3] Fernandez J.-C., Jard C., Jeron Th., and Viho C. Using on-the-fly verification techniques for the generation of test suites. In Proc. 8th Conference on Computer Aided Verification, volume 1102 of Lecture Notes in Computer Science, New Brunswick, August 1996.

[4] Drobintsev P.D. Integrirovannaia tehnologia obespechenia kachestva programmnih produktov s pomoshiu verifikacii i testirovania. Kand. dis., SPbGPU. 2006. 238 p.

[5] .ITU-T Recommendations Z.140-142 (2002): The Testing and Test Control Notation Version 3 (TTCN-3).

[6] TAT+Beta User's Manual © 2001-2005 MOTOROLA.

[7] Hoare C.A.R. Communicating sequential processes, Prentice Hall, London, 1985.

[8] Letichevsky A., Kapitonova J., Letichevsky Jr., A., Volkov V., Baranov S., Weigert T. Basic protocols, message sequence charts, and the verification of requirements specifications, Computer Networks: The International Journal of Computer and Telecommunications Networking, v.49 n.5, p.661-675, 5 December 2005.

[9] ITU Recommendation Z.120. Message Sequence Charts (MSC), 11/99.