Generation of Test Scenarios for Non Deterministic and Concurrent Telecommunication Applications

Kotlyarov Vsevolod, Drobintsev Pavel, Voinov Nikita
Peter the Great Saint-Petersburg Polytechnic University
Saint-Petersburg, Russia
vpk@spbstu.ru

***Abstract*— The paper presents an approach to tests generation for industrial software systems with non deterministic and concurrent behavior. A brief overview of modern model driven test technologies is presented; benefits and problems of these approaches are highlighted. Specifics of concurrent and non deterministic behavior are analyzed to identify issues with such behavior testing. As result of the issues analysis usage of non-linear symbolic test scenarios for reducing test suite size is suggested and presented in examples. Based on the suggestion an approach for construction of non linear tests from linear ones is described with an example of industrial project. Results with description of main benefits for suggested approach are presented.**

Keywords*— *Model Driven Testing; Automation design of test scenarios; Reducing of tests explosion.

I. INTRODUCTION

Creation of technologies for software quality assurance is one of the most actively growing areas of software developing process. A lot of developing technologies for support of quality guarantying process are based on formal approach. Such technologies are called model-oriented and lead to creation of application formal model with usage of graphical language, verification or some checking of the model and different types of generation. If formal representation of software system is used for generation of application target code then such approaches are called MDD (Model Driven Development) [1], MDSD (Model Driven Software Development) [2] or MDE (Model Driven Engineering)[3]. In case of formal model is used for testing automation purposes such as scripts or test generation based on behavior described in the model the approach can be called MDT (Model Driven Testing) [4] or MBT (Model Based Testing) [5].

The paper is devoted to description of some experience in usage of model-oriented technologies for testing of industrial telecommunication projects. The experience is based on usage of tests generated from formal models for checking of non deterministic and concurrent systems quality. Existing approach is based on linear test generation and execution, paper provides enhancement for such approach with non linear tests producing.

II. MODEL ORIENTED APROACH IN TESTING AUTOMATION

Usage of a model-oriented testing is based on creation of formal models which can be used on different testing phases. The main feature of such approaches is generation of test suite in accordance with user defined criteria of coverage.

Statistics collected in companies which use such approaches shows [6] that model-oriented techniques are usually used on system testing phase (up to 80% of projects) with the main goal - functional testing (up to 96%). The reason of such company's approach is complexity of system testing for big industrial projects, which is based on huge efforts, spent to quality guarantying [7]. To resolve this problem software developing companies try to reduce efforts for tests creation and simplify tests execution process. Usually reducing of testing efforts is linked to communication with customers because only customer of the software has deep knowledge about domain specifics and model oriented approach helps to simplify such communications.

Researchers also consider that more than 80% [6] of model-oriented approaches are using graphical notations, which allows simplifying of work with formal notations for developers. Requirements for knowledge of testers and customer representatives are reduced by this way and process of models developing is also simplified.

The following advantages of model-oriented approaches in comparison with manual test development methods can be found in research papers [6]:

- Reduction of software development cost due to usage of testing automation and verification techniques;
- Ability to use abstract models, which allow to simplify testing development process and involve customer in work with software quality checking;
- Tests generation and execution automation, which allow to reduce cost of testing process;
- Ability to have communication with customer on requirements level starting from earlier phases of development process, which allow to identify inconsistencies on stage of requirements gathering before system code implementation;

The following issues are usually highlighted in process of MDT implementation in software development process [6]:

- Different levels of abstraction in formal model and code of developing software. The issue leads to necessity of generated tests customization before execution on target software code;
- Necessity to customize MDT approaches in projects from different areas of industry. Usually practices and techniques from one target domain are not applicable in another one, this paper presents some unified approach that can be used in different domains because it is based on syntax of the model and not on semantics;
- A shortage of engineers with MDT expertise due to the lack of such specialists in the market of software developers and the need to train qualified personnel for the implementation of MDT approaches within the software development process of companies. Approach presented in the paper is based on very simple and intuitive formal language and this helps to resolve issue with strong MDT expertise;
- The problem of test suite explosion. A potential number of tests generated with MDT for checking of industrial software quality is too huge but the time for testing cycle is restricted. The main target for approach presented in the paper is reducing of test suite in case of non deterministic and concurrent behaviors presented in formal specification.

TABLE I. COMPARATIVE ANALYSIS OF MDT AND MANUAL APPROACHES

Testing phase	**MT**		**MDT**	
	Flexibility	***Speed***	***Flexibility***	***Speed***
Creation of test suite	high	low	middle	high
Maintenance of tests	middle	low	middle	high
Maintenance of test suite	low	low	high	high
Tests execution	high	low	high	high
Adding of tests	high	low	middle	high
Test analysis	middle	low	high	high
Coverage analysis	low	low	high	high

Table 1 contains comparative analysis of traditional manual approach and MDT approaches based on different criteria. Values “high”, “middle” and “low” show qualitative analysis for different criterion of flexibility and speed obtained for manual testing (MT) and model driven testing (MDT). A result of the analysis shows that MDT has a lot of benefits and its usage in industrial software testing is reasonable. The following criteria were used:

- Creation of test suite – developing of tests based on requirements;
- Maintenance of tests – correction of particular tests associated with the change of the functionality of the System Under Test (SUT);
- Maintenance of test suite – configuration activity with goal to make correction in all tests which belong to test suite;
- Tests execution – execution of tests on SUT;
- Adding of tests – creation of tests on a new SUT functionality;
- Tests analysis – analysis of tests suite execution results, debugging and bug fixing;
- Coverage analysis – analysis of test suite coverage based on user defined criterion.

Comparative analysis shows that the main problems with MDT are flexibility of creation and adding of test suite. The issue is mainly based on usage of different levels of abstraction which are used in software model and real implementation. The solution here is selection of appropriate abstraction level for SUT model with usage of effective graphical notation which not only allows system semantic description but also simplifies process of model creation.

III. TESTING AUTOMATION IN INDUSTRIAL SOFTWARE SYSTEMS

One of the main characteristics of industrial software systems testing is constraint on test suite size, which shall be used for quality checking. This is connected with automatic or semi automatic approaches to test creation and execution when formal model is used for generation of test suite as a set of system behavior scenarios. Usage of such approach for models of industrial systems leads to generation of huge test suite, which supports all scenarios of SUT behavior. But execution of such test suite is impossible because of constraints on time to testing and therefore only subset of tests shall be selected from all test suite for execution on SUT. This process is called test suite reduction [8].

The goal of test suite reduction is creation of minimal test suite, which will allow to find the same set of defects as initial test suite [9]. Usage of such reduction allows to reduce efforts for maintenance of tests and their execution and as a result efforts for all testing cycle. Traditional approach to reduction is based on static analysis and instrumentation of SUT code to determine a set of tests which satisfy to selected criterion [10]. The main difference in reduction methods is criterion of test selection. Usually the following criteria are used:

- “All-uses” coverage [11];
- Definition-use associations coverage [12];
- Control flow graph coverage [13, 14];
- Modified Condition/Decision Coverage [15];
- Branches coverage [16];

- Paths coverage [16].

Common part of presented coverage methods is necessity to SUT code instrumentation for test suite reduction and this leads to a set of usage limitations. First of all usage of methods is impossible before finishing of development phase because code of SUT is needed. Secondly process of code analysis for big software systems is very complicated task and therefore testing with initial test suite without reduction can be more effective than code analysis.

Usage of MDT for test suite reduction is more effective because code of application is not needed and all coverage analysis can be done based on system model. Therefore it is not needed to wait for the finish of code development and test suite creation can be started when only requirements exist. The only condition for such method is that semantics of the model shall not be changed during development, and if such changes happen then they shall be addressed in model checking procedure. At the same time, the existence of a formal model makes it possible to apply the following methods of software verification that can be used effectively to reduce the number of tests:

- Model checking;
- Symbolic verification;
- Searching for equivalence classes;
- Searching for cycle invariants;
- e.t.c.

Thus, the use of MDT test approaches in industrial software is justified not only on test generation step, but also during analysis and reduction of the test set to be executed.

IV. Model of SUT

In modern project documentation formulation of initial requirements is specified either constructively, when the checking procedure can be reconstructed from the requirement text in a natural language, or non constructively, when functionality specified in the requirement contains no hints on how to check it.

A procedure which checks the current requirement is an exact sequence of the causes and the results of some activities (coded with actions, signals, states), which analysis can prove or refute that this requirement is covered. Such checking procedure can be used as a criterion that a certain requirement is covered; i.e., it can be a so-called criteria procedure. In the text below the term "sequence" or "chain" of events will be used for a criteria procedure.

Different notations can be used for defining such sequences with to reduce manual efforts for tests development. One of them is high level notation TTCN [22]. But usage of formal notation for definition of test suite cannot give any guaranty of coverage or consistency of test specification. To achieve this verification procedure shall be used for proving of different tests properties.

In suggested approach an existing technology based on VRS/TAT toolset is used [23]. In VRS/TAT the Use Case Maps (UCM) notation [17] is used for a model high level description, while tools for automation of checking and test generation work with a model in the basic protocols language [18].

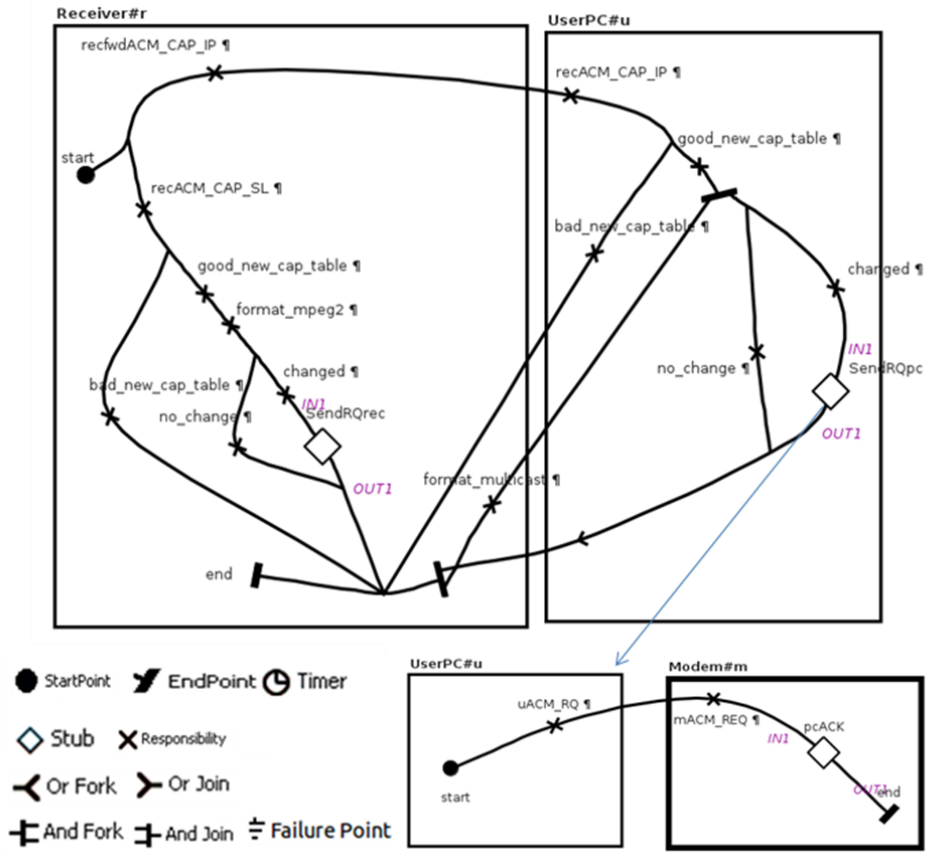


Fig.1 Example of UCM model

A UCM model (Fig. 1) contains a description of a model of two interacting instances. Each path in the graph from the event "start" to the event "end" represents some behavioral scenario. Each path contains a certain number of events (Responsibilities). Events in the diagram are marked with the symbol ×, while Stub elements which encode inserted diagrams are marked with the symbol ◇. As a result, each scenario contains a certain sequence of events. The set of possible scenarios is specified through a set of such sequences. In these terms a chain is determined as a subsequence of events enough to conclude that the requirement is satisfied. A

path in the UCM diagram, containing the sequence of events of some chain is called a trace covering the respective requirement. Tests for experimental evidence that the requirement is covered can be generated from such trace.

V. Formal Model in Basic Protocols

A basic protocol (BP) [18] is the main element of the formal model. It codes the minimal observable step of the system behavior and represents an analog of the Hoare triplet with a pre-condition, a post-condition, and a process (an observable action, a series of actions). The pre and post conditions are logical formulae with inequalities and arithmetic operations which describe a subset of system states before and after the process actions which consist in sending/receiving messages and/or changing the values of application variables. BPs may contain symbolic or concrete parameter values (i.e., for variables in pre and post-conditions, in expressions for signals, state descriptions, etc.), the respective tolerance ranges being specified for symbolic values. Specifying concrete values for BP parameters is called BP concretization.

A BP may refer to one or more requirements, as well as a number of BPs or a number of ordered BP sets (BP chains) may refer to one requirement.

A set of BPs composes a requirement model. One may construct scenarios or traces to visualize possible behaviors of the system under design by just combining consistent pre and post conditions of various BPs. Traces with symbolic parameters are called symbolic traces. The verifier [3, 4] proves correctness of a behavior case represented by a concrete or symbolic trace [19].

A trace set which covers all chains of all requirements forms the requirement model of the application under development. This model may be used for generation of a complete test suite which checks the functional coverage of all constructively specified requirements [8].

Let s_0 be some initial system state, which includes the state of the environment and the states of all agents inserted into it [20]. Then all possible traces (histories of system functioning) may be obtained as sequences of the form:

$$s_0 \xrightarrow{b_1(n_1,m_1)} s_1 \xrightarrow{b_2(n_2,m_2)} \dots$$

$b_1(n_1,m_1), b_2(n_2,m_2), \dots$ being the concretized BPs. Here $b_1, b_2, \dots$ are the names of the respective BPs; $n_1, n_2, \dots$ are the names of key agents (a key agent is a particular BP, whose state may change in this BP); and $m_1, m_2, \dots$ are the values of the remaining parameters which satisfy the pre-conditions of the respective BPs. Post-conditions defining deterministic transformations, the states $s_1, s_2, \dots$ are unambiguously determined by the initial system state and BP concretization. Let's describe a system S(P) which realizes a system P of basic protocols in form of an attributed (or labeled) transition system. Let's define the behavior of the system S(P) in its different states. The states of the system S(P) are equated to basic language formulae extended with intermediate states which correspond to execution of the respective BPs [20].

Let's denote the behavior of the system S in the state α as S_α. Letichevsky proved [18] that the equation for $\mathbf{S}_\alpha^\infty$ ($\mathbf{S}_\alpha^\infty$ means that all non-deadlock traces are infinite) has the form:

$$\mathbf{S}_\alpha^\infty = \sum_{p \in P(\alpha)} \mathbf{proc}(p) * (\mathbf{T}(\alpha, p) : \Delta) * \mathbf{S}_{\mathbf{T}(\alpha,p)}^\infty$$

Here * is a composition of transitions (proc) and states of the behavior graph, $P(\alpha) = \{p \in P_{inst} \mid \alpha \to \mathbf{pre}(p)\}$, P_{inst} is the set of initiated concretized protocols, proc (p) - is the BP process, β is a constant of successful termination (shows that the protocol was applied successfully),

$\mathbf{T}(\alpha, p) = \mathbf{Tr}(\alpha, \mathbf{post}(p))$. $\mathbf{Tr}(\alpha, \beta)$ - is a predicate transformer. Two formulas α and β are provided to its input, and a new formula γ is generated as its output which strengthens the post-condition β and considers no changed variables in pre-condition β. In other words, the formula γ is such that $\gamma \to \beta$. This strengthening is necessary to apply next BPs whose post-condition is an indirect corollary of the post-condition β and the pre-condition β.

The next definition includes finite traces with a successful termination. For generality, let's identify a set P^0 of terminating protocols and let $P^1 = P \setminus P^0$. Terminating protocols are supposed to terminate the system work and do not expect its continuation. The equation for a complete system has the form:

$$\mathbf{S}_\alpha = \sum_{p \in P^1(\alpha)} \mathbf{proc}(p) * (\mathbf{T}(\alpha, p) : \Delta) * \mathbf{S}_{\mathbf{T}(\alpha,p)} + \sum_{p \in P^0(\alpha)} \mathbf{proc}(p) * (\mathbf{T}(\alpha, p) : \Delta) * (\mathbf{S}_{\mathbf{T}(\alpha,p)} + \Delta)$$

VI. Methods of Tests creation for non Deterministic and Concurrent Behavior

Guarantying of SUT quality with usage of MDT approach is based on execution of generated test suite. For example generated test can represent a counterexample of some property violation obtained from verification system. Usage of verification counterexamples for testing leads to issue with coverage of system behavior because it covers only particular system behavior, but for testing purposes it is more interesting to cover some particular part of all behaviors of the SUT. One of the possible solutions to avoid the issue with coverage is traversing of all behavior tree of the system with the goal to obtain test suite, which will satisfy some coverage criterion. In

this case the user will have test suite, which covers for example all branches of behavior or all instructions of the program. Very important that such coverage will be produced by verification system so we can say that system behavior was proved in accordance with user defined properties and criteria. The tests will be linear paths with stimulus sent to the SUT and responses for controlling of behavior, where linear means absence of alternative behaviors in the scope of one test. But usage of such linear tests is not correct in some cases for testing of complex software systems. The main problems will appear for testing of passive alternatives and concurrent behavior. These two cases will be considered in details.

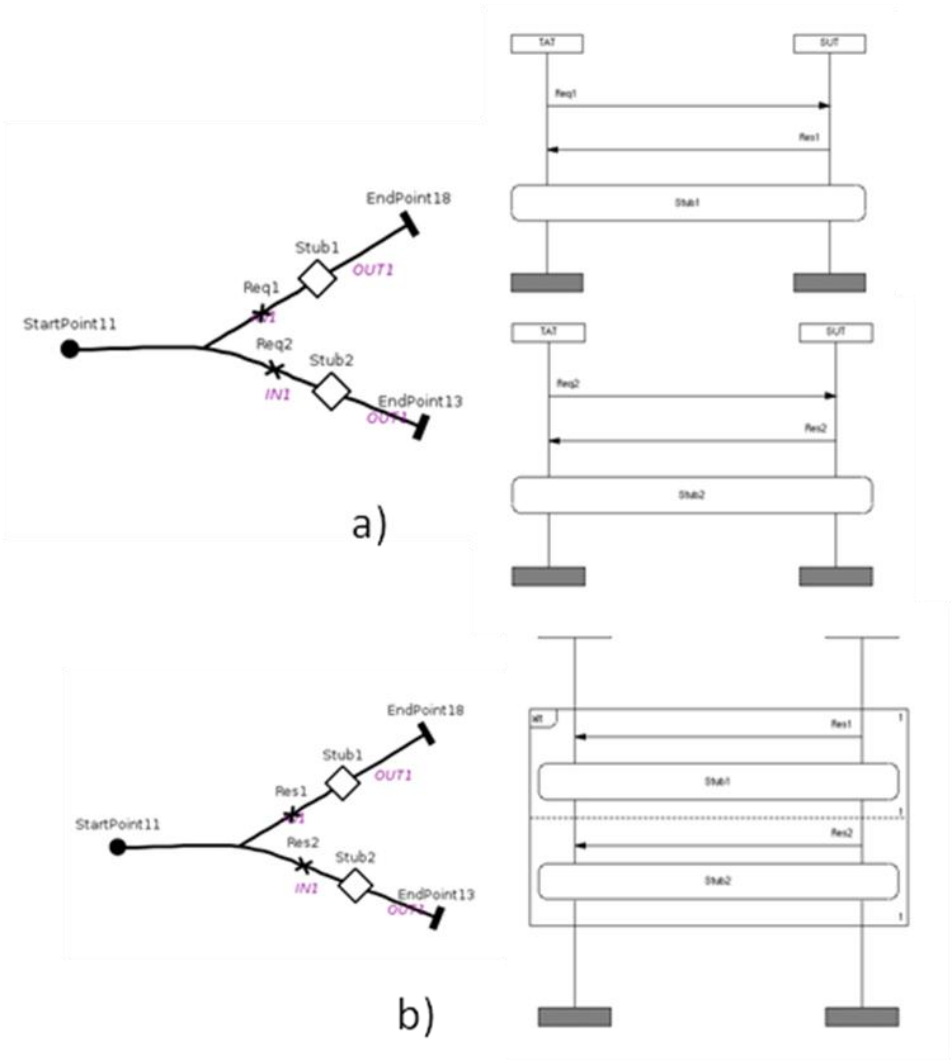


Fig 2. Example of active alternative behavior

A. Testing of Passive Alternatives

Alternative behavior is describing behavior of the SUT, which is associated with usage of deterministic or non deterministic selection of future path. If this selection is deterministic then it can managed by test with sending of some stimulus or signal to the SUT and controlling of SUT response. In this case alternative is called "active". Fig. 2 a) shows example of active alternative in UCM notation and test diagram in MSC [21] language, which will cover such behavior.

The example illustrates the main feature defining that this situation belongs to active alternative template – this is direction of the signals Req1 and Req2 in corresponding responsibilities. These signals are sent into SUT and its behavior depends on the signals. In other words behavior of the SUT is managed by test. Testing of such behavior can be done based on a set of two linear tests, which are presented on the figure.

"Passive" alternative is a situation when a signal which determines future system behavior comes from SUT. More over this signal cannot be determined based on analysis of behavior history presented in the test. Such behavior is non deterministic and cannot be tested with usage of linear test, because test will check only one of possible behavior and its correctness will be determined only on execution phase, so some mechanism for definition of a set of tests shall be used. One of the possible solutions for such situation is usage of non-linear test with "alt" construction. Fig. 2 b) shows a passive alternative example and non-linear test for checking of its correctness.

Decision about correctness of signal coming from SUT is made in process of the test execution, but not on test generation phase. So the test will work in both cases with incoming signal Res1 and Res2 and will fail only if some different signal comes. Usage of non-linear constructions allows to decompose tests and use them for testing of passive alternatives

However usage of non-linear tests is not usual practice for MDT approach because generated tests are based on counterexamples, which are still linear. So generation of the non-linear tests shall be defined by additional algorithms, which can be based on UCM structure analysis with future gluing of a set of traces into non-linear test.

B. Testing of Concurrent Behavior

As it was described earlier one of the main problems for testing of concurrent systems is state explosion which is caused by interleaving between interacting concurrent threads. A number of states which shall be tested is huge and grows dramatically in case of adding states to each particular thread. Testing of such systems becomes very complex due to the following reasons:

- First of all, in the process of test suite running all possible combination of states which are traversed while execution of the system shall be checked.
- Secondly, initialization of the system and its transition into some particular state is very complex because execution of one thread can affect another threads and it leads to necessity of threads interaction analysis.

In the scope of described approach usage of partial order reduction method is suggested for generation of test suite with coverage of independent concurrent threads behavior. Also in this approach a special analysis of threads dependencies based on UCM language is provided to define additional synchronization points which helps to enlarge area of state reduction method applicability. Below detailed description of the approach is presented and illustrated in examples.

If independency of threads set is proved by some way then task of test generation is pretty simple. On the first step the state of the model before AndFork element shall be stored and concurrent threads shall be defined (threads start from AndFork and finish in AndJoin elements). After threads were determined they shall be glued into linear test scenario. As a result test scenario with coverage of all branches is obtained without any interleaving.

If concurrent threads are dependent to each other then static analysis of such dependency shall be performed. As a result of analysis additional synchronization points shall be added into the system description and in this case the task will be reduced to the previous one.

It should be noted that even if independency of threads was proved, usage of one linear trace is not enough due to the fact that in concurrent systems in most cases it is impossible to control execution of each thread. This leads to usage of non-linear traces for testing.

Let's consider an example of simple concurrent system described in Fig. 3.

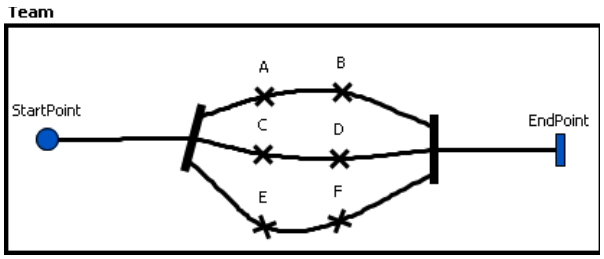
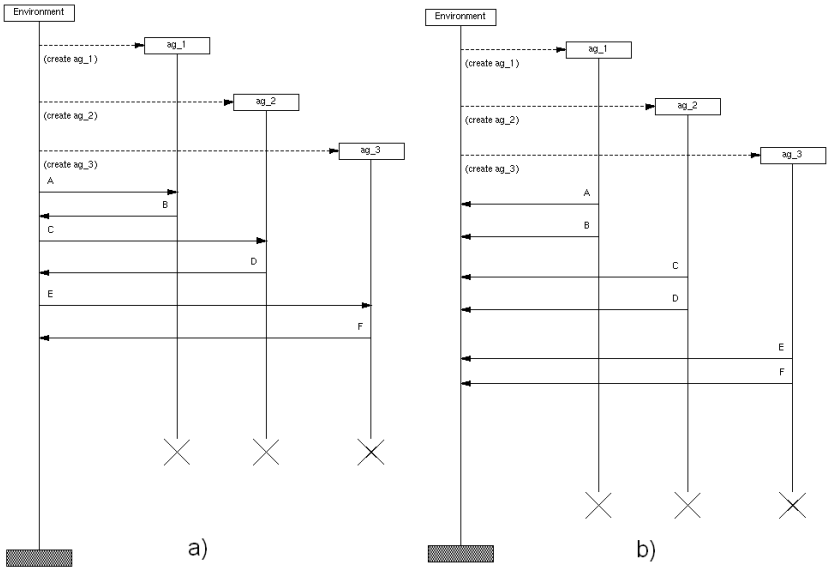


Fig.3 Test cases for concurrent threads

Suppose that responsibility elements in this diagram describe receiving and sending of signals pairs (A;B), (B;C) and (C;D) by system under test. Then the test which is based on assumption about threads independency can be presented like diagram in Fig. 4 a).

For description of the test MSC language is used. Fig 4 a) shows that only one sequence of system behavior is A,B,C,D,E,F and in this case it will be correct because the test manages signals sending into system under test. Now let's suppose that responsibilities describe receiving pairs of signals (A;B), (B;C) and (C;D) from system under test. The test for such behavior can be described by diagram in Fig. 4 b). This test will be incorrect because it will fail in cases when the system behavior is correct. The problem in this case is that in testing of real software it is impossible without special control of the test to ensure that the signal pair (A; B) will come first



(the same statement applies to the remaining pairs of signals). For example sequence C,D,A,B,E,F will be correct from system point of view, but it will be failed by test case.

Fig.4 Test cases for concurrent threads

To solve this problem, the operator "par" can be used [21], which is the syntactic structure of language MSC and allows to describe the interaction of parallel processes. Fig. 5 shows a valid test, obtained by adding the operator par.

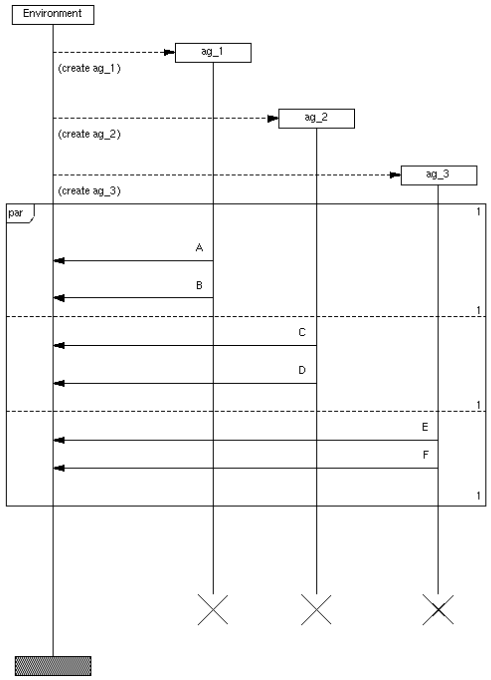


Fig.5 Test scenario with "par" construction

The figure shows that the operator consists of three blocks, each of which describes the interaction of the environment with one of the concurrent threads. Such record determines the sequence of arrival of signals in one particular stream and allows interleaving (without explicitly specifying all possible options, which significantly reduces the entry test) between the concurrent processes. It should be noted that the addition of the operator "par" is a simple procedure and can be fully automated on the basis of information on the generation and synchronization of threads.

Thus, the presented approach allows to automatically generate test suite for testing of concurrent systems based on information extracted from UCM diagram. The main advantage of the approach is automatic obtaining of tests without interleaving which is dramatically reduces test description. Another benefit is automatic definition of reduction methods applicability area based on threads independency analysis.

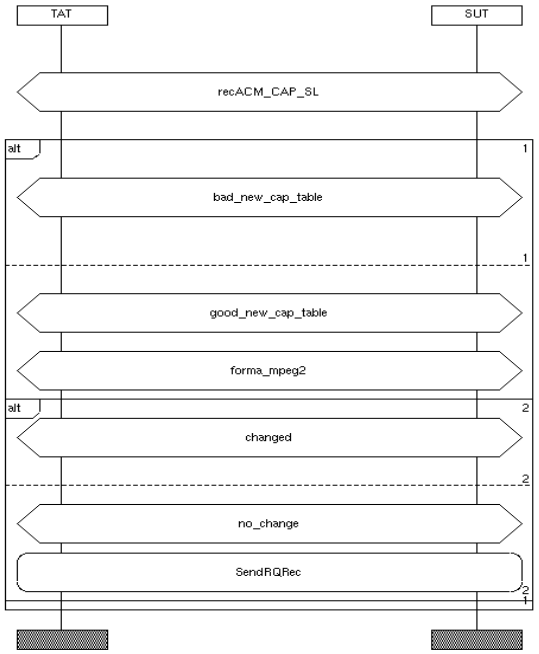
VII. An Example of Presented Approach

Let's consider an example of work with passive alternative and concurrent behavior on small part of industrial project. The project describes a communication protocol between user terminal and receiver of satellite signal. Only small part devoted to configuration activities is considered. The behavior is presented on Fig 1. in UCM notation.

Behavior of the system starts from ability to receive two signals which are recfwdACM_CAP_IP and recACM_CAP_SL. These signals are coming from environment (in the process of test execution they will be sent by the test), such situation can be determined as active alternative behavior.

In case when signal recACM_CAP_SL was received receiver shall analyze address table and based on analysis results move to one of the possible alternative: "good_new_cap_table" or "bad_new_cap_table". Condition which determines future path is hidden in variables of SUT and can not be managed by signals on test level so this situation is identifying passive alternative behavior and shall be tested with usage of non-linear construction "alt". Branches "changed" and "no_change" have the same semantics.

In case when signal recfwdACM_CAP_IP was received analysis of address table shall be done by user application. Passive alternative here is presented in pair of branches "good_new_cap_table" and "bad_new_cap_table". In case of "good_new_cap_table" concurrent behavior with branches "format_multicast" and "changed/no_change" shall be tested. As described earlier operator "par" can be used for such testing. Fig. 6 shows two tests which can be used for testing of described SUT behavior.



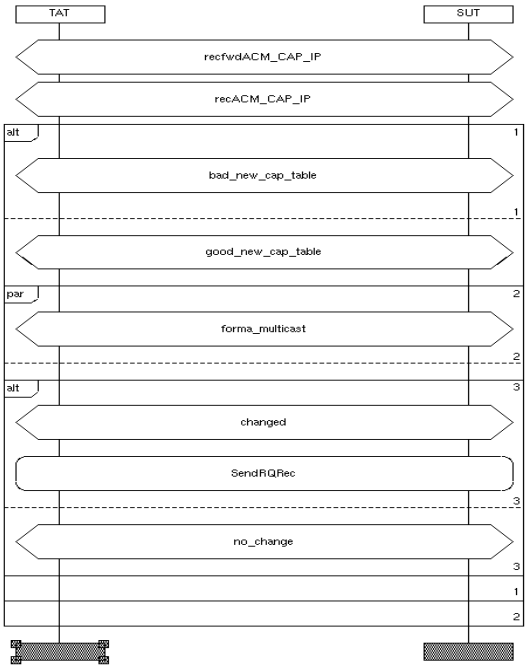


Fig.6 Example of non linear tests

On the figure 6 "conditions" elements of MSC notation are used to show responsibilities of UCM map with appropriate signal and actions, which are described in metadata of these responsibilities. Also MSC element "reference" is used to describe stubs of UCM map. The first test describes checking of case with receiving of "recACM_CAP_SL" signal. Here two "alt" constructions are used and one of them with "changed" and "no_change" branches is nested. The second test describes receiving of recfwd_ACM_CAP_IP signal and consists of two "alt" and one nested "par" constructions. These two tests fully cover all possible behaviors of the system and will be converted into linear representation in process of tests execution.

VIII. Problems of Suggested Approach

Let's consider possible problems of suggested approach implementation. For passive alternative case the problem with coverage analysis still exists. The issue is based on test execution, which leads to coverage of only one behavior of the system in one execution cycle. As described earlier one non-linear test contains a set of behaviors but in process of its execution the only one behavior will be used. To avoid such issue additional coverage analysis shall be made after test execution procedure. This analysis will allow to identify which particular behavior were covered and which shall be covered by additional tests executions.

Another problem is provided by concurrent behavior testing. As described earlier such tests can be created based on linear tests analysis. But a number of tests can be huge because of interleaving in concurrent systems. Therefore complexity of non-linear tests creation will grow with adding of concurrent processes into SUT specification. So additional methods to reduce a number of tests shall be used to simplify process of non-linear tests creation.

IX. Results of the Approach Piloting

The approach was piloted on three different industrial telecommunication projects with different number of concurrent parts. SMTP project devoted to description of communication protocol has small size (50-100 BPs). Projects CDMA and Satellite terminal which describe modules of telecommunication systems have middle size (100-500 BPs). The Table 2 presents results of the piloting.

TABLE II. Results of piloting in telecommunication projects.

Project	**BP number**	**Non linear BP %**	**Test scenario number (Linear)**	**Test scenario number (Linear+ non linear)**	**Reduce of test suite %**
SMTP	30	10	10	8	20
CDMA	205	43	1171	615	48
Satellite terminal	191	15	396	291	27

The results show that number of non-linear tests is less than number of linear for all of the projects. The reducing of test suite strictly depends on a number of protocols which present non linear behavior. Average reducing of test suite with usage of suggested approach is near 30%.

X. Conclusions

Usage of presented approach is effective for testing of industrial telecommunication systems. It allows to reduce a number of tests to be executed due to hiding of interleaving with non linear tests in case of concurrent behavior checking. Also passive alternative cases can be checked correctly with non linear tests. Reducing of tests amount depends on structure of the SUT but in average for piloted project is was near 30 per cent. This work was supported by FCP grant.

References

[1] Oscar Pastor, Sergio Espana, Jose Ignacio Panach, Nathalie Aquino. Model-Driven Development. Springer-Verlag Informatik-Spektrum October 2008, Volume 31, Issue 5, pp 394-407.

[2] Sami Beydeda, Matthias Book, Volker Gruhn. Model-Driven Software Development. Springer-Verlag Berlin Heidelberg 2005.

[3] Paul Baker, Shiou Loh, FrankWeil, Model-Driven Engineering in a Large Industrial Context — Motorola Case Study.

[4] Baillargeon, R. and Flores, R., "Model Driven Testing," SAE Technical Paper 2008-01-0743, 2008, doi:10.4271/2008-01-0743.

[5] Dias-Neto, A.C.; Horta Travassos, G. "Supporting the Combined Selection of Model-Based Testing Techniques", Software Engineering, IEEE Transactions on, On page(s): 1025 - 1041 Volume: 40, Issue: 10, Oct. 2014.

[6] Robert V. Binder, Anne Kramer, Bruno Legeard, 2014 Model-based Testing User Survey: Results, 2014 http://model-based-

testing.info/wordpress/wp-content/uploads/2014_MBT_User_Survey_Results.pdf

[7] Fenton, N.E., Ohlsson, N. Quantitative analysis of faults and failures in a complex software system. Software Engineering, IEEE Transactions on. Issue 8 Aug 2000

[8] P. Drobintsev, V. Kotlyarov, A. Letichevsky, A Formal Approach to Test Scenarios Generation Based on Guides. Automatic Control and Computer Sciences, 2014, Vol. 48, No. 7, pp. 415–423.

[9] McMaster, S. and Memon, A. M. 2005. Call Stack Coverage for Test Suite Reduction. In Proceedings of the 21st IEEE international Conference on Software Maintenance (ICSM'05) - Volume 00 (September 25 - 30, 2005). ICSM. IEEE Computer Society, Washington, DC, 539-548.

[10] Y. Kichigin, An approch to test suite reduction Proceedings of System Programming Institute http://citforum.ru/SE/testing/kichigin/#1

[11] M. J. Harrold, R. Gupta, and M. L. Soffa. A methodology for controlling the size of a test suite. ACM Transactions on Software Engineering and Methodology, 2(3):270-285, July 1993.

[12] H. Zhu, P. A. V. Hall, and J. H. R. May. Software Unit Test Coverage and Adequacy. ACM Computing Surveys, Vol.29, No.4, pp.366-427, December 1997

[13] G. Rothermel, M. J. Harrold, J. von Ronne, and C. Hong. Empirical studies of test-suite reduction. Journal of Software Testing, Verification, and Reliability, V. 12, no. 4, December, 2002.

[14] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria. In Proceedings of the 16th International Conference on Software Engineering, pages 191-200, May 1994.

[15] J. A. Jones, M. J. Harrold. Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage. IEEE Transactions on Software Engineering, Vol. 29, No. 3, March, 2003, pp. 195-209.

[16] B. Beizer, "Software Testing Techniques," Second Edition, Van Nostrand Reinhold Company Limited, 1990.

[17] Z.151 : User requirements notation (URN) - Language definition http://www.itu.int/rec/T-REC-Z.151-200811-I/en

[18] Letichevsky A.A., Kapitonova J.V., Kotlyarov V.P., Letichevsky O.O., Volkov V.V., Baranov S.N., Weigert T.: Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications. In: Proc of ISSRE04 Workshop on Integrated Reliability Engineering (ISSRE04:WITUL), IRISA, Rennes France (2004)

[19] Kolchin, A., Letichevsky, A., Peschanenko, V., Drobintsev, P., Kotlyarov, V. An approach to creating concretized test scenarios within test automation technology for industrial software projects. Automatic Control and Computer Sciences Volume 47, Issue 7, December 2013, Pages 433-442

[20] A.A. Letichevsky, J.V. Kapitonova, V.P. Kotlyarov, A.A. Letichevsky Jr., N.S.Nikitchenko, V.A. Volkov, and T.Weigert. Insertion modeling in distributed system design // Проблеми програмування. – 2008. – С. 13–38.

[21] Recommendation ITU_T Z. 120. Message Sequence Chart (MSC), 11/2000.

[22] Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language; ETSI ES 201 873-1 V4.7.1 (2015-06)

[23] Baranov, S., Kotlyarov, V., Letichevsky, A., and Drobintsev, P., The technology of automation verification and testing in industrial projects, Proc. St. Petersburg IEEE Chapter, Int. Conf., St. Petersburg, 2005, pp. 81–86.