Industrial approach in requirement engineering

Igor Chernorutsky[1], P.D. Drobintsev[2], V.P. Kotlyarov[3]

[1]Ph.D., prof., SPbPU
[2]Ph.D., assistant prof., SPbPU,
drob2@ics2.ecd.spbstu.ru
[3]Ph.D., prof., SPbPU,
vpk@ics2.ecd.spbstu.ru

Abstract. The article describes the approach how to transform initial requirements in natural language into the formal requirement model. The model is used to analyze the behavior of the designed system to meet the original requirements during the design stage. Successfully verified the formal requirement model is used to generate test cases that completely cover the behavior of the analyzed application.

Keywords: requirement engineering, requirement specification, formal specifications development, requirements behavioral model.

1 Introduction

The issue of verifying implementation of large-scale and complicated requirements specification is among the main issues of industrial software development and test automation. Documents with requirements specification are usually formulated in natural language and may contain hundreds and thousands of requirement items. Therefore requirements formalization for describing behavioral scenarios used for automated tests or manual test procedures development is characterized as a task of huge complexity and big efforts.

Applicability of formal methods in the industry is largely determined by the level of formalization language adequacy to the engineering practice involving not only code developers and testers but also customers, project managers of different levels, marketers and other specialists. Quite obvious that none of the logical languages can be used for adequate requirements formalization which would keep the semantics of the developed application and at the same time satisfy all «persons involved» [1].

In contemporary project documentation initial requirements are formulated either constructively, when scenario of requirement fulfillment checking can be extracted directly from the text of this requirement in natural language, or unconstructively, when there is no clarification how to check some feature contained in the requirement.

For example, behavioral requirements for telecommunication applications are formulated constructively given that requirements execution procedure is specified, and in this case verification and testing can be used to check their feasibility. Non behavioral requirements are often formulated unconstructively, which requires some additional information in formalization process allowing to work out scenarios of their checking or in other words to transform unconstructive way of formulation into constructive one.

2 Checking requirements fulfillment

Procedure of a requirement checking is exact sequence of causes and consequences of some activities (encoded by actions, signals and states), whose analysis allows to confirm whether this requirement has been fulfilled or not. Such checking procedure being used as criteria of requirement fulfillment can be called criteria procedure. The term "sequence" or "chain of events" will be used along with the term “criteria procedure” in the text of this article.

Tracking the steps of performing criteria procedure in system’s behavioral scenario (either hypothetical or implemented in a model or in a real system) can be served as confirmation that the corresponding requirement is fulfilled in the system under analysis.

Procedure of a requirement checking (a chain) is formulated by specifying the following information for all elements of this chain:

- conditions (a set of causes) required for activation of some activity;
- the activity, which shall be executed under current conditions;
- consequences - observable (measurable) results of activity execution.

All the following is used to specify causes and consequences: signals, messages or transactions, commonly used in communications between instances of reactive systems [2], as well as variables states specified in terms of values or restrictions on region of admissible values. Coverage of corresponding chains can be observed by tracing changes in the states caused by chains activities. During analysis it is allowed to consider direct transition into a state with empty activity and alternative ways of changing states in case of nondeterministic behavior.

Issues with unconstructive formulating of requirements are solved in the process of procedures development that should check requirements fulfillment on user interfaces or interfaces between components. It means that the requirement for some functionality should be interpreted by functions of user interface or API (Application Program Interface) of components. These functions should describe the encoded functionality in such a way that, based on this interpretation, you can set the procedure for checking the corresponding functionality in the application.

Similar interpretation is formulated in Use Case [3] diagrams based on knowledge of GUI (Graphical User Interface) of the developed application or API (Application Program Interface) of its components.

Thus, chains with sequences of activities and states can be used as criterion of requirements fulfillment. Besides that, there may be cases in which the criterion for the fulfillment of a certain requirement is specified by several chains instead of a single chain.

3 Initial documents with requirements specification

The following types of documentation are usually used while formulating technical requirements in industrial software projects:

- Marketing Requirement Specification (MRS) – enumeration of new functionality from customer's point of view, often described in Use Case format;
- Technical Requirement Specification (TRS) – enumeration of complete functionality to be implemented with brief explanation of each function;
- Functional Requirement Specification (FRS) – detailed description of complete functionality to be implemented, full enough for test set creation to test software product on all stages of the lifecycle.

Each requirement in these documents is more often formulated in natural language and expressed in one of two ways:

- as behavioral requirement, when scenario of requirement fulfillment checking can be extracted directly from the text of this requirement in natural language;
- as non behavioral requirement, when just a structure or wish of having some feature is expressed without explanation of how this feature can be checked or tested.

Due to constructive way of expression behavioral requirements allow static and/or dynamic methods of verification and testing for checking their implementation.

Non behavioral requirements are expressed in unconstructive way and require additional information for creating a constructive scenario of their checking in order to make verification and testing applicable.

Formalization of any constructively formulated requirement as well as effective automated analysis of software requirements are possible and implemented in VRS/TAT technology [4, 5, 6].

4 UCM notation

Use Case Maps (UCM) notation [3] is used in VRS/TAT technology for high level model description while tools which perform automation of checking and generation processes work with model in basic protocols language [3, 4].

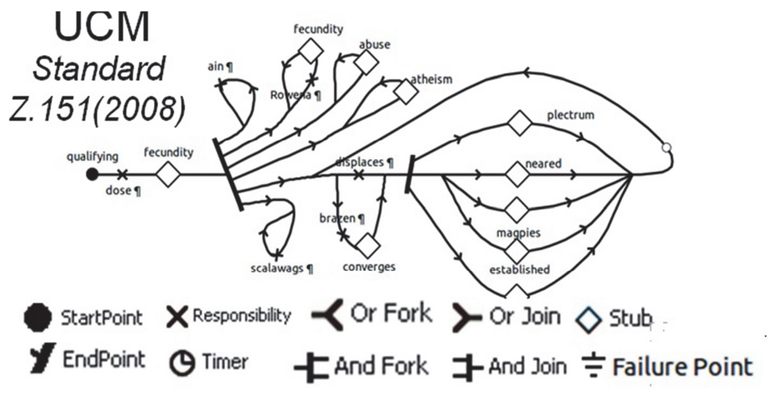


Fig. 1. UCM notation

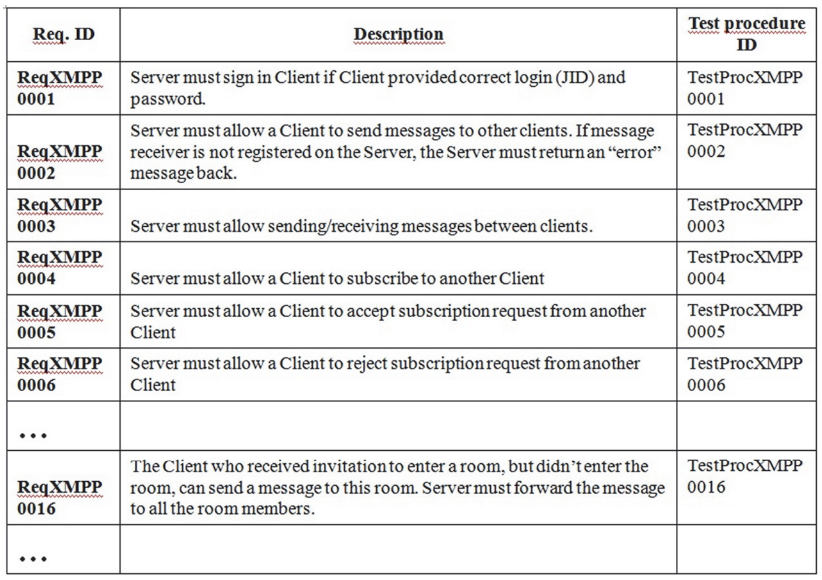
UCM notation is clear for a designer, a tester and a customer. It allows describing behavior of complex software projects in a formal way. Although a transition from informal requirements to their formal model is performed manually, it can be significantly simplified by using developed solutions based on templates. The rest stages of VRS/TAT verification and testing integrated technology are automated [6]. For application or tests source code generation a detailing (technique of Lowering [11, 12]) can be used which does not break proved results of abstract model's correct behavior.

5 Expressing initial requirements

Considered below are the stages of requirements processing on the example of Extensible Messaging Presence Protocol (XMPP) implementation project. XMPP [7] is extensible protocol for publish-subscribe systems, VoIP, video, files transfers, Internet of Things (IoT) applications, games and so on. XMPP is an open standard based on XML.

This paper illustrates integrated methodic for formalizing initial requirements and creating requirements formal model which can be applied after detailing for symbolic verification and generation of symbolic test scenarios and further executable test sets. Functioning XMPP server called Apache Vysper is used as a system under test (SUT) in this project.

Example fragment of XMPP initial requirements version is shown in Fig. 2:



Req. ID	Description	Test procedure ID
ReqXMPP 0001	Server must sign in Client if Client provided correct login (JID) and password.	TestProcXMPP 0001
ReqXMPP 0002	Server must allow a Client to send messages to other clients. If message receiver is not registered on the Server, the Server must return an "error" message back.	TestProcXMPP 0002
ReqXMPP 0003	Server must allow sending/receiving messages between clients.	TestProcXMPP 0003
ReqXMPP 0004	Server must allow a Client to subscribe to another Client	TestProcXMPP 0004
ReqXMPP 0005	Server must allow a Client to accept subscription request from another Client	TestProcXMPP 0005
ReqXMPP 0006	Server must allow a Client to reject subscription request from another Client	TestProcXMPP 0006
…		
ReqXMPP 0016	The Client who received invitation to enter a room, but didn't enter the room, can send a message to this room. Server must forward the message to all the room members.	TestProcXMPP 0016
…		

Fig. 2. A fragment of requirements table

Each requirement in Fig. 2 has a corresponding criteria chain or a test procedure (Fig. 3). Test procedure contains pre- and post- conditions as well as actions for covering the requirement.

For example, the ReqXMPP0003 requirement: «Server must allow sending/receiving messages between clients» has the corresponding TestProcXMPP0003 test procedure which covers this requirement.

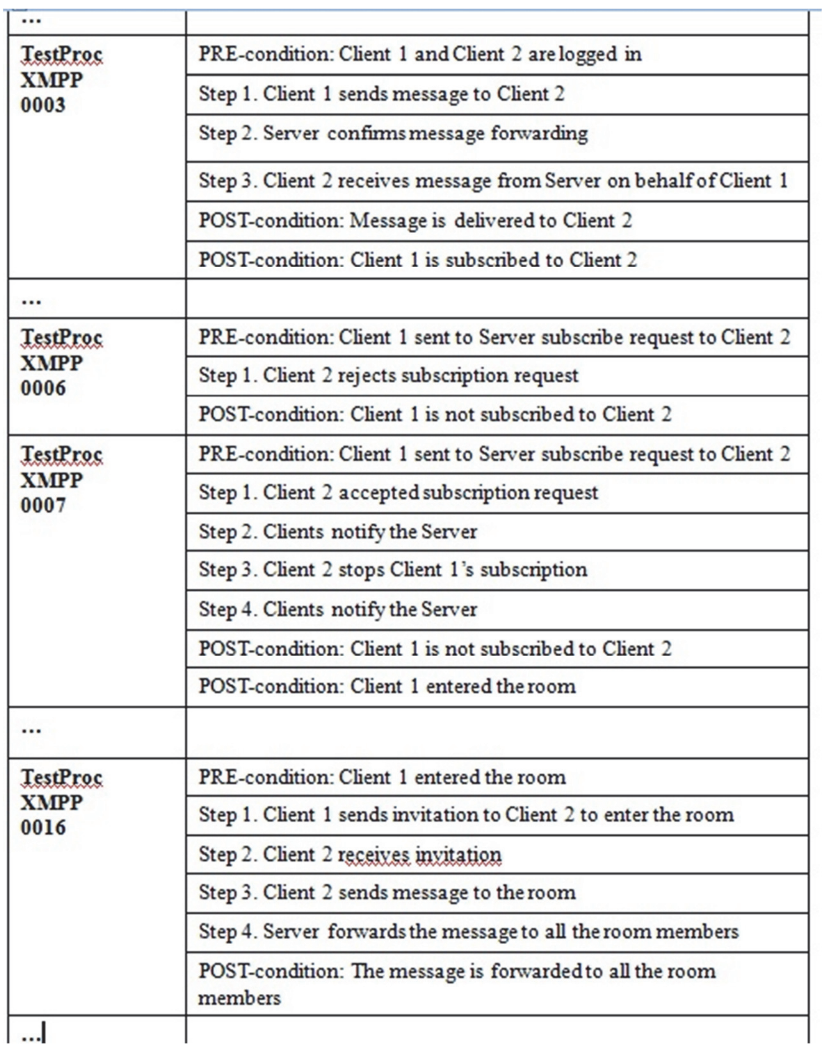
Pre-condition: Clients have been initialized on the server.

Step 1. The first client sends a message to the second client.

Step 2. The server confirms message dispatch.

Step 3. The second client receives the message from the first client.

Post-condition: The message has been successfully delivered.



…	
TestProc XMPP 0003	PRE-condition: Client 1 and Client 2 are logged in
	Step 1. Client 1 sends message to Client 2
	Step 2. Server confirms message forwarding
	Step 3. Client 2 receives message from Server on behalf of Client 1
	POST-condition: Message is delivered to Client 2
	POST-condition: Client 1 is subscribed to Client 2
…	
TestProc XMPP 0006	PRE-condition: Client 1 sent to Server subscribe request to Client 2
	Step 1. Client 2 rejects subscription request
	POST-condition: Client 1 is not subscribed to Client 2
TestProc XMPP 0007	PRE-condition: Client 1 sent to Server subscribe request to Client 2
	Step 1. Client 2 accepted subscription request
	Step 2. Clients notify the Server
	Step 3. Client 2 stops Client 1's subscription
	Step 4. Clients notify the Server
	POST-condition: Client 1 is not subscribed to Client 2
	POST-condition: Client 1 entered the room
…	
TestProc XMPP 0016	PRE-condition: Client 1 entered the room
	Step 1. Client 1 sends invitation to Client 2 to enter the room
	Step 2. Client 2 receives invitation
	Step 3. Client 2 sends message to the room
	Step 4. Server forwards the message to all the room members
	POST-condition: The message is forwarded to all the room members
…	

Fig. 3. A fragment of test procedures table

UCM diagram is used in the project as a requirements model. UCM diagram shown in Fig. 4 describes communication between XMPP server and XMPP client.

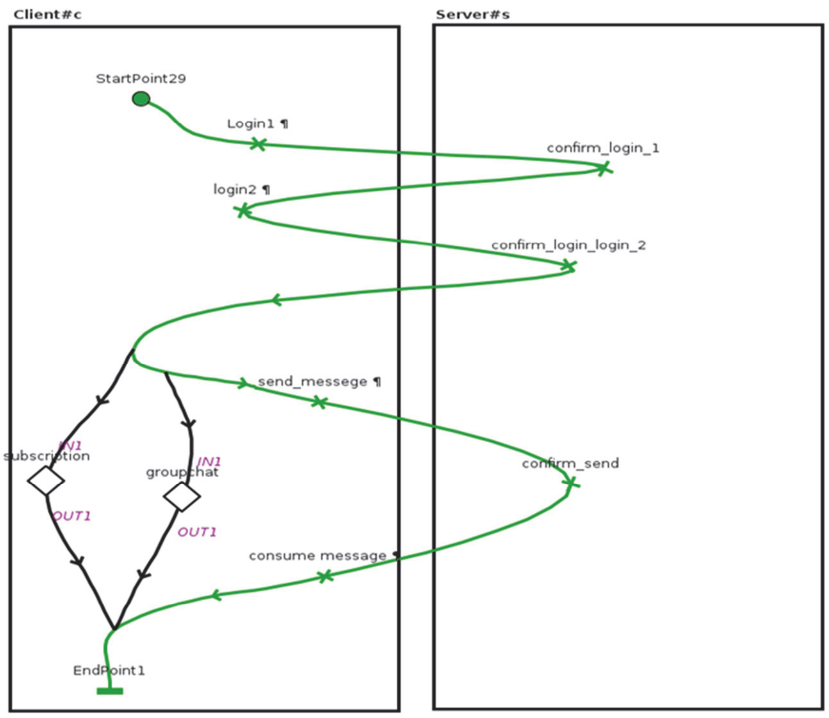


Fig. 4. Behavioral model represented as a UCM diagram

The trace marked with green color on the diagram corresponds to the TestProcXMPP0003 test procedure.

Note that requirement fulfillment can be traced by its steps:

1. Four events Login1, Login2, Confirm_Login1 and Confirm_Login2 (both clients logging in to the server and get confirmation from the server) altogether compose a precondition for the send_message event - sending a message from the first client to the second client (Precondition: Clients have been initialized on the server).
2. The confirm_send event is the server's confirmation of message dispath.
3. The consume_message event means that the second client has successfully received the message and post-condition has been achieved: the message has been successfully delivered. The fact, that the EndPoint1 point has been reached, means the end of the test procedure and successful fulfillment of the ReqXMPP0003 requirement.

A scenario generated from a formal model in UCM format and corresponding to the test procedure can be automatically presented as a symbolic test scenario in MSC [5,8] format (Fig. 5).

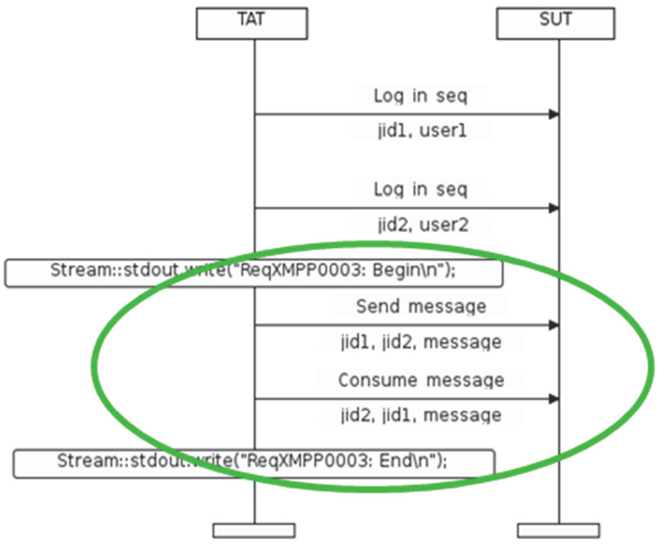


Fig. 5. Behavioral scenario in MSC format

If criteria chains checking requirements fulfillment are formulated and agreed with customer for all requirements, it provides criteria for functionality checking of the whole software product.

Checking criteria chains for each constructive requirement means checking its semantics from customer's point of view, of course only after all criteria procedures had been approved by the customer. Implementation of such check is rather complex task if it is not fully automated.

UCM models of requirements may have complicated structured view (Fig. 6). For example, UCM model of the ReqXMPP0007 requirement: «Server must allow client to unsubscribe other client at any time» is represented in Figure 6 and contains encapsulated UCM diagram.

The test procedure corresponding to the ReqXMPP0007 requirement model contains joint sequence of events from both diagrams in its behavioral scenario (Fig. 7):

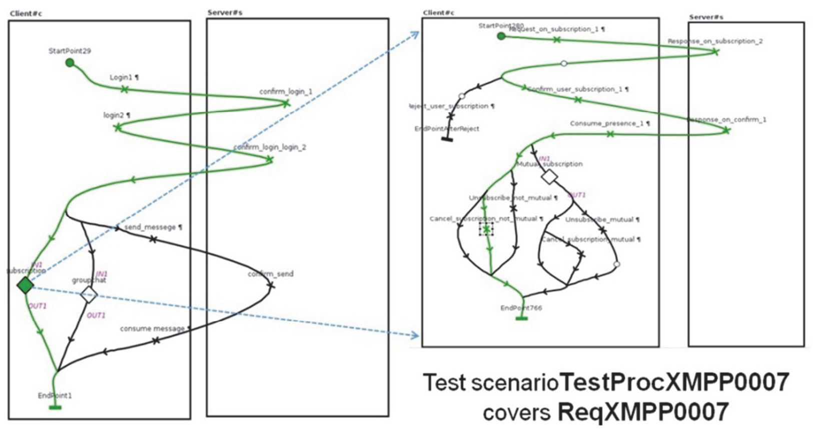


Fig. 6. UCM diagram for structured requirement

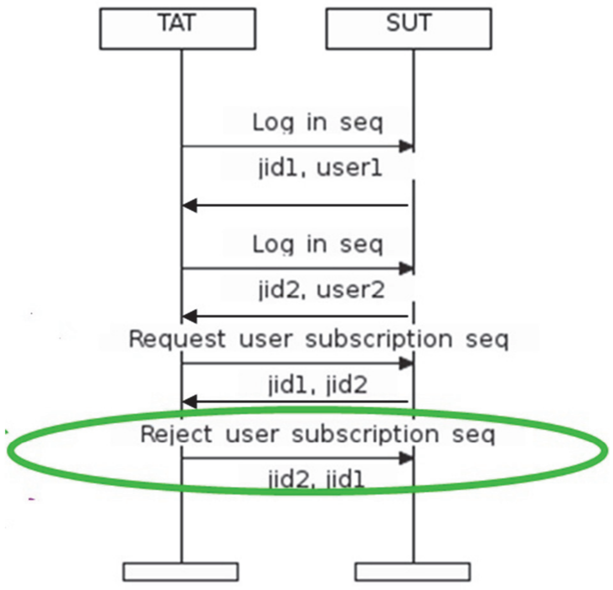


Fig. 7. Symbolic scenario for checking the ReqXMPP0007 requirement

Symbolic MSC scenario uses signals with identification parameters of the jid1 and jid2 communicating clients, i.e. the described behavior can be spread on a number of clients. Allowed restrictions on region of admissible values for any parameter shall be controlled by means of static and dynamic control [9].

Means of automated model correctness checking are provided by VRS/TAT [6] integrated technology of verification and testing which includes tools for symbolic verification, test sets generation and tests execution within a single automated process with minimal efforts required for tests development and execution steps.

6 Detecting and fixing bugs in requirements

The criterion of error in the specification of the requirement is the fact that the criteria chain is unreachable and this fact is detected during the verification process (Fig. 8).

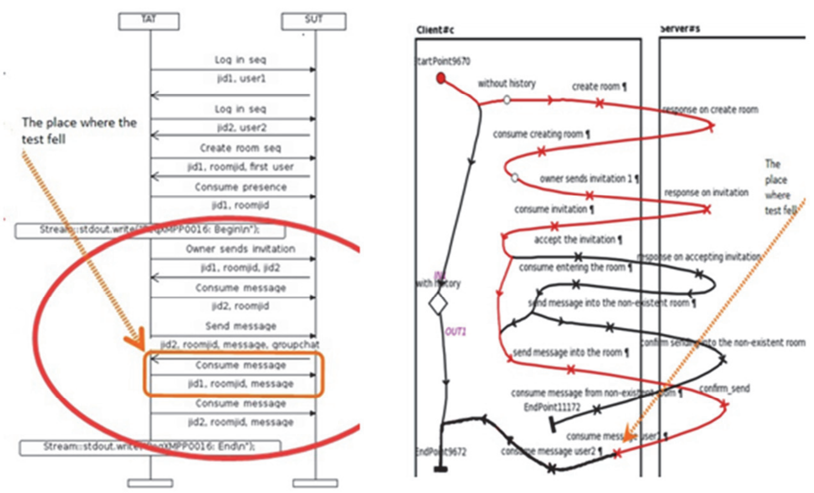


Fig. 8. Identifying a bug during verification in MSC scenario and UCM diagram

The reason of test scenario unreachability (or failing) detected during verification process lies in incorrect formulation of the ReqXMPP0016 requirement (Fig.9). The problem is that it is not enough to enter a room to be allowed to send messages in this room. In other words, an invitation received by a user shall not give him rights to use this room.

The corrected ReqXMPP0016 requirement is the following: «Client who received invitation to enter a room but didn't enter a room can't send messages to this room. Server shall response to the client with «error» message».

Changes in the semantics of requirement lead to corresponding correction of UCM model (Fig. 9), where the ReqXMPP0016 requirement is reachable.

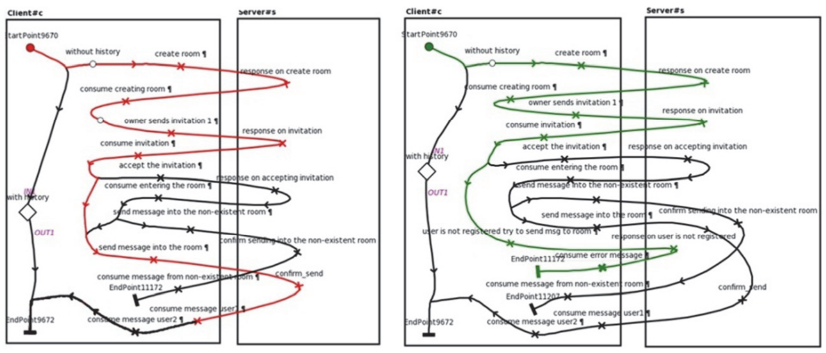


Fig. 9. Initial and corrected UCM models of a requirement

Covered requirements are displayed on UCM model with colored path. There can be individual color for each requirement which is very useful for coverage analysis or model correction.

Providing the full coverage of all requirements implies proving the correctness of all behavioral scenarios kept in requirements. However this does not guarantee absence of some incorrect (not specified in requirements) behavior which can be a result of the environment not considered in behavioral model.

The described approach provides features of automated analysis of incorrect behavior conditions [10] and algorithms (filters) for their detection in the process of application functioning. This is done in the process of backword - the reverse proof of the correctness of the trace from the found point of unreachability to the beginning with the addition of the indispensable Precondition As a result the correspondence between behavioral model and initial requirements is provided.

7 Results

The proposed approach was tested in industrial telecommunications projects of the Motorola Saint-Petersburg department of software development and allowed to automate the process of designing and developing functional tests. Due to automation it was possible to reduce the complexity of this process more than 5 times, while ensuring the guaranteed completeness of testing.

8 Conclusion

The proposed approach has the following advantages:

- Usage of formal methods is a good practice to guarantee the software quality.
- High abstraction level of formal model allows working with customer on the development phase. It provides proving the model's behavior correctness.

- Analyzing the behavior of all functionality modes mapped to requirement specifications.
- Calculate permissible range of parameters used in the behavioral scenarios.
- For application or tests source code generation a detailing (technique of Lowering [12]) can be used which does not break proved results of abstract model's correct behavior.
- Generation of software filters to control and prevent behavioral scenario from leaving the calculated limits.
- The integrated technology of design and testing was applied in the domain of wireless telecommunications applications and demonstrated efforts reduction in 26% on software product development.

Acknowledgments. The results of this work were obtained using ccNUMA system and RSC Tornado cluster in Supercomputer Center of Peter the Great St. Petersburg Polytechnic University. The work was financially supported by the Ministry of Education and Science of the Russian Federation in the framework of the Federal Targeted Programme for Research and Development in Priority Areas of Advancement of the Russian Scientific and Technological Complex for 2014-2020 (№ 14.584.21.0022, ID RFMEFI58417X0022).

References

1. Baranov S., Kotlyarov V., Letichevsky A.: An Industrial Technology Mobile System Test Automation Based on Verified Behavioral Models of Project Requirement Specifications. In: Proc. International scientific conference: space, astronomy and programming. Lavrov Readings. May 20-22, SPbU, Math-Mech Feculty, St.Petersburg (2008), pp. 134-145.
2. Z. Manna, A. Pnueli.: The Temporal Logic of Reactive and Concurrent Systems. Springer-Verlag, 1992.
3. Recommendation ITU-T Z.151. User requirements notation (URN), 11/2008.
4. S. Baranov, V. Kotlyarov, A. Letichevsky, P. Drobintsev. The technology of Automation Verification and Testing in Industrial Projects. // Proc.of St.Petersburg IEEE Chapter, International Conference, May 18-21, St.Petersburg, Russia, 2005 – pp.81-86.
5. A. Letichevsky, J. Kapitonova, A. Letichevsky Jr., V. Volkov, S. Baranov, V. Kotlyarov, T. Weigert. Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications. Proc of ISSRE04 Workshop on Integrated-reliability with Telecommunications and UML Languages (ISSRE04:WITUL), 02 Nov 2004: IRISA Rennes France.
6. I. Anureev, S. Baranov, and others. Tools for supporting integrated technology of analysis and verification of specifications for telecommunication applications // SPIIRAN works- 2013-№1-28P.
7. Day M., Rosenberg J. and H. Sugano, "A Model for Presence and Instant Messaging," RFC 2778, February 2000.
8. Recommendation ITU_T Z. 120. Message Sequence Chart (MSC), 11/2000.

9. Kolchin A., Kotlyarov V., Drobintsev V. Method of generating test scenarios in the environment insertion simulation. // "Control systems and machines", Kiev, "Academperiodika, t.6-2012, S.42-48.
10. Drobintsev P., Kotlyarov V., Nikiforov I., Letichevsky A., Peschanenko V. Approach to Behavior Scenarios Debugging, Modeling and Analysis of Information Systems, 2015, p.14.
11. Drobintsev P., Kotlyarov V., Nikiforov I., Voinov N. Model Oriented Approach for Industrial Software Development , Modeling and Analysis of Information Systems, 2015, p.10.
12. Drobintsev P., V. Kotlyarov, I. Nikiforov, N. Voinov, I. Selin. Conversion of abstract behavioral scenarios into scenarios applicable for testing. SYRCoSE-2016, ISPRAS, pp. 96-101.