Conversion of abstract behavioral scenarios into scenarios applicable for testing

Pavel Drobintsev < drob@ics2.ecd.spbstu.ru>
Vsevolod Kotlyarov <vpk@ics2.ecd.spbstu.ru>
Igor Nikiforov < i.nikiforov@ics2.ecd.spbstu.ru>
Nikita Voinov <voinov@ics2.ecd.spbstu.ru>
Ivan Selin < ivanselin93@gmail.com>
Peter the Great Saint-Petersburg Polytechnic University,
29 Polytechnicheskaya str, St. Petersburg, 195251, Russian Federation

Abstract. In this article, an approach of detailing verified test scenarios for developed software system without losing the model's semantics is proposed. Existing problem of generating test cases for real software systems is solved by using multi-level paradigm to obtain the real system signals, transactions and states. Because of this, the process is divided into several steps. Initial abstract traces (test cases) with symbolic values are generated from the verified behavioral model of software product. On the next step, called concretization, these values in test scenarios are replaced with concrete ones. Resulting concrete traces are then used as input for the next step, data structures conversion. This step is needed because concrete traces do not contain all the information for communicating with developed software and presented in another way with different data structures. After concrete test scenarios are detailed, they can be used for generation of executable test cases for informational and control systems. In this paper, a software tool is suggested for detailing test scenarios. It consists of several modules: a Lowering editor that allows user to create rules of detailing a signal, a Signals editor used to define complex data structures inside the signal and a Templates editor that eases work with similar signals. Process of translating abstract data structures into detailed data structures used in system implementation is presented with examples.

Keywords: model approach; model verification; test mapping

DOI: 10.15514/ISPRAS-2016-28(3)-9

For citation: P. Drobintsev, V. Kotlyarov, I. Nikiforov, N. Voinov, I. Selin. Conversion of abstract behavioral scenarios into scenarios applicable for testing. Trudy ISP RAN / Proc. ISP RAS, vol. 28, issue 3, 2016, pp. 145-160. DOI: 10.15514/ISPRAS-2016-28(3)-9.

1. Introduction

One of the most perspective approaches to modern software product creation is usage of model oriented technologies both for software development and testing. Such technologies are called MDA (Model Driven Architecture) [1,2], MDD (Model Driven Development) [2] and MDSD (Model Driven Software Development) [3]. All of them are mainly aimed to design and generation of application target code based on a formal model.

The article is devoted to specifics of model oriented approaches usage in design and generation of large industrial software applications. These applications are characterized by multilevel representation related to detailing application functionality to the level where correct code is directly generated.

The idea of model oriented approach is in creating of multilevel model of application during design process. This model is iteratively specified and detailed to the level when executable code can be generated. On the design stage formal model specification allows using verification together with other methods of static analysis with goal to guaranty correctness of the model on early stages of application development.

More than 80% [4] 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 is reduced by this way and process of models developing are also simplified.

2. Levels of behavioral models development

One of high level languages for system formal model specification is Use Case Maps (UCM) [5, 6]. It provides visible and easy understandable graphical notation. Further abstract models will be specified in UCM language to demonstrate proposed approach in details. Also considered is VRS/TAT technology chain [7], which uses formal UCM models for behavioral scenarios generation.

Traditional steps of formal abstract model development in UCM language are the following:

1. Specifying main interacting agents (components) and their properties, attributes set by agent and global variables.
2. Introducing main system behaviors to the model and developing diagrams of agent's interaction control flow.
3. Developing internal behaviors for each agent and specifying data flow in the system.

Undoubted benefit of UCM language is possibility to create detailed structured behavioral diagrams. Structuring is specified both by Stub structural elements and reused diagrams (Maps), which are modeling function calls or macro substitution. Unfortunately, standard UCM language deals with primitive and abstract data structures, which are not enough to check implementation of a real system. This drawback is compensated by using metadata mechanism [6]. But metadata does not allow detailing data flow to more detailed levels. That's why for creating detailed behaviors it is proposed to use vertical levels of abstractions during behavioral models development which are: structured system model in UCM language, behavioral scenarios with symbolic values and variables, concrete behavioral scenarios are behavioral scenarios with detailed data structures.

Another benefit of UCM usage is possibility to execute model verification process. UCM diagrams are used as input for VRS/TAT toolset which provides checks for specifications correctness. These checks can detect issues with unreachable states in the model, uninitialized variables in metadata, counterexamples for definite path in UCM, etc. After all checks are completed the user gets a verdict with a list of all findings and a set of counterexamples which show those paths in UCM model which lead to issue situations. If a finding is considered to be an error, the model is corrected and verification process is launched again. As a result after all fixes a correct formal model is obtained which can be used for further generation of test scenarios.

After formal model of a system has been specified in UCM language, behavioral scenarios generation is performed. Note that behavioral generator is based not on concrete values assigned to global variables and agents attributes, but on symbolic ones which reduces significantly the number of behavioral scenarios covering the model. However symbolic test scenarios cannot be used for applications testing as executing behavioral scenarios on the real system requires concrete values for variables. So the problem of different level of abstraction between model and real system still exists. In VRS/TAT technology concretization step [8] is used to convert symbolic test scenarios. On this step ranges of possible values for variables and attributes are calculated based on symbolic formula and symbolic values are substituted with concrete ones. But concretization of abstract model's behavioral scenarios is not enough for their execution, because on this stage scenarios still use abstract data structures which differ from data structures in real system. As a result conversion of concretized behavioral scenarios of abstract UCM level into scenarios of real system level was integrated into technology chain for behavioral scenarios generation.

2. Concretization

In behavioral scenarios data structures are mainly used in signals parameters. There are two types of signals in UCM model: incoming to an agent and outgoing from an agent. Incoming signals are specified with the keyword "in" and can be sent either by an agent or from outside the system specifying with the keyword "found". Outgoing signals are specified with the keyword "out" and can be sent either to an agent or to outside the system specifying with the keyword "lost".

An example of outgoing signal can be seen on Fig. 1. The element "send_Fwd_Rel_Req_V2_papu" contains metadata with the signal

"Forward_Relocation_Request_V2" and UCM-level parameter "no_dns". Outgoing signals can only be used inside of "do" section as a reaction of the system on some event.

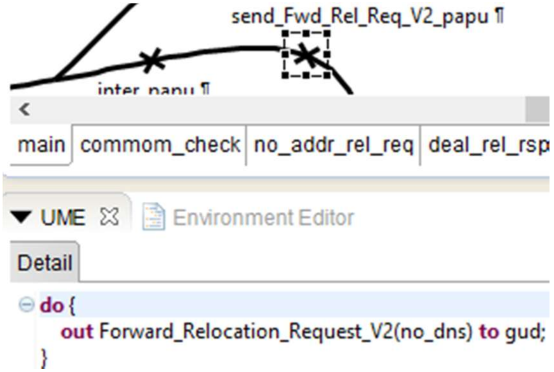

Fig. 1. Description of "Forward_Relocation_Request_V2" signal in metadata corresponding UCM element

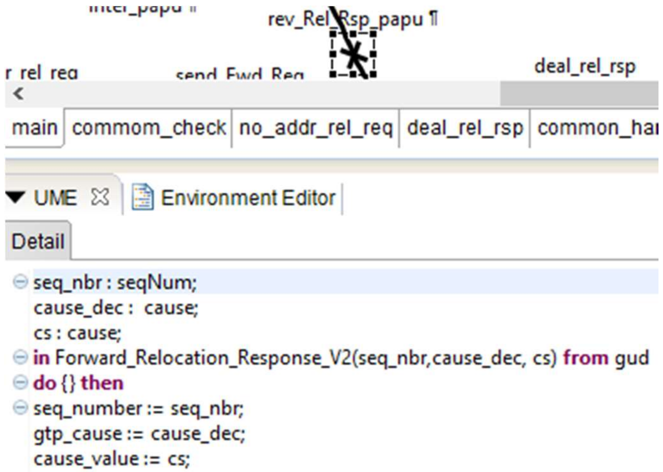

Fig. 2. Description of the " Forward_Relocation_Response_V2" signal in metadata of the "rev_Rel_RSP_papu" UCM element

If the signal Forward_Relocation_Response_V2 is received, then new values taken from signal parameters are assigned to variables.

Consider an example of converting signal structure of UCM level into detailed structures of real system for the signal "gtp_forward_relocation_req_s". Based on high level UCM model symbolic behavioral scenarios are generated containing data structures described in metadata of UCM elements. Fig.3 contains symbolic test scenario where the agent "GTP#gtp" receives the signal "gtp_forward_relocation_req_s" from agent "GMG#gmg". In symbolic scenarios actual names of UCM model agents specified in metadata are used.

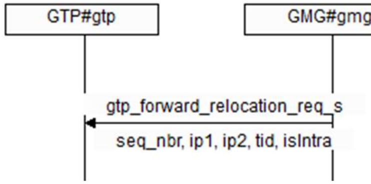

Fig. 3. Symbolic test scenario with the signal "gtp_forward_relocation_req_s"

Symbolic behavioral scenario is input data for concretization module, which substitutes symbolic parameters with concrete values. In current example the parameters "sqe_nbr", "ip1", "ip2", "tid" and "isIntra" are substituted with values "invalid", "valid", "exist", "valid" and "0". Fig.4 contains concrete behavioral scenario.

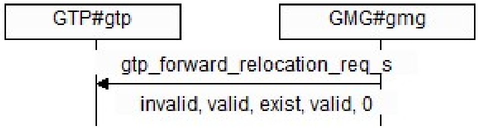

Fig. 4. Concrete test scenario with the signal "gtp_forward_relocation_req_s"

4. Data structures conversion

After concretization, scenarios still have to be processed because their structure does not match with one's of system under test (SUT). The most straightforward approach is to manually review all generated scenarios and edit all used signals so that their structure will reflect SUT interfaces. Obviously, it will require too much time and may be a bottleneck of the whole process. Therefore, there is a need for automation.

The common way is making a wrapper that transforms signals to desired form using one of popular programming languages (C++, Java, etc.). However, it could lead to making new mistakes and loss of correctness of test scenarios. The main reason for this is ability to implement incorrect structures on scenarios level. In addition, other language-specific errors are possible. Cutting down the ability to produce incorrect code will reduce the number of mistakes while still maintaining good level of automation.

4.1 Approach

To be able to satisfy these needs a two-step approach called "Lowering" was suggested. The name comes from descending on lower levels of abstraction. In

general, lowering can be described as creating processing rules for each signal called "lowering rules" and application of these rules to the concrete scenarios.

As said above, there are some restrictions on possible operations to save the correctness of test scenarios, such as:

- It is prohibited to separate constants into several independent parts (e.g. separating value 1536 in 15 and 36 is not possible)
- It is prohibited to separate fields of variables values
- Only structures similar to SUT interfaces can be created
- Only constant template values and values that were obtained during concretization step are allowed

Limitation was made by creating a special language that is used to define lowering rules. Despite having all these limitations, user can define complex signal and protocol structure dependent on UCM signal parameters in accordance with language grammar. On Fig. 5, you can see the grammar in Backus–Naur Form.

```
LoweringSpec ::= UCMSignal "->"
LoweringRule | LoweringSpec UCMSignal "->"
LoweringRule
LoweringRule ::=  LoweringCondition |
LoweringRule LoweringCondition
LoweringCondition ::= <condition STRING>
ConditionContent
ConditionContent ::= LoweredElement |
LoweredElement ConditionContent
LowredElement ::= LoweredDo | LoweredSignal
| LoweredAction
LoweredDo ::= <code STRING>
LoweringSignal ::= <signal name STRING>
SignalContent
SignalContent ::= ValueNotation Instance
Via
ValueNotation ::= <empty> | <value STRING>
| "(." ValueNotation ".)" | ValueNotation
"," ValueNotation
Instance ::= <empty> | "TAT" | "SUT"
Via := <empty> | <port STRING>
UCMSignal ::= Name UCMParam
Name ::= <name STRING>
UCMParam ::= <empty> | <param name STRING>
| UCMParam "," UCMParam
```

Fig. 5. Lowering rules language grammar

4.2 User perspective

For selected UCM-level signal user can define lowering rules. As you can see on Fig. 6, rule consists of trigger condition and content. Content can be either one detailed signal, several signals or actions performed on the variables.

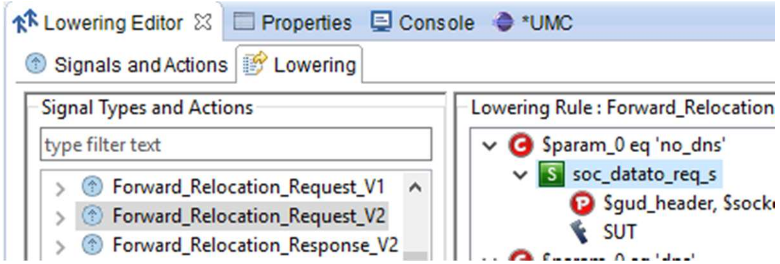

Fig. 6. Lowering editor with signal "Forward_Relocation_Request_V2" being selected

After specifying the condition and choosing the type of content, user can edit it in the right part of the editor. This part dynamically changes depending on what is selected in the middle of the editor.

For example, some signal was selected. Signals editor will appear in the right part of Lowering Editor (Fig. 7).

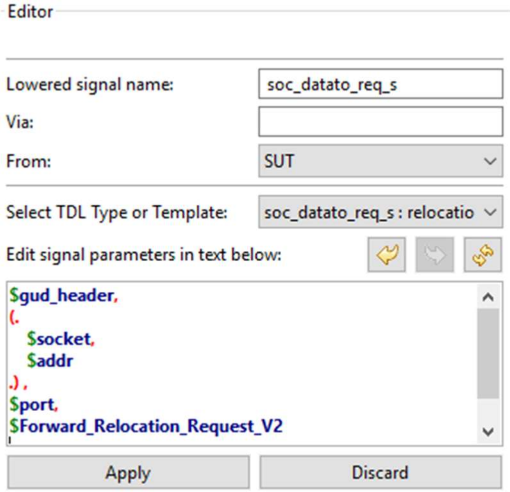

Fig. 7. Signals editor

User selects the needed SUT interface in the drop-down list named "Select TDL Type or Template". Then user names the signal and puts concrete values in the fields of detailed signal. Often similar conversion rules are required for different signals. Templates can be used to simplify this approach. A developer can define a template of detailed signal, specify either formula or concrete values as a parameter

of detailed signal and then apply this template for all required signals. For each case of template usage a developer can specify missed values in the template, change the template itself or modify its structure without violating specified limitations. Templates mechanism simplifies significantly the process of conversion rules creation.

Consider the process of templates usage. Templates are created in separate editor (Templates Editor). In Fig.8 the template "template_0" is shown which contains detailed data structures inside and the dummy values which shall be changed to concrete values when template is used.

Note that template can be created only from SUT interfaces description or another template.

When a template of data structure is ready, it can be used for creation of conversion rules. Fig.9 represents usage of the template "template_0" with substituted concrete values of signal parameters instead of the dummy value "value_temp", which then will appear in behavioral MSC scenario.

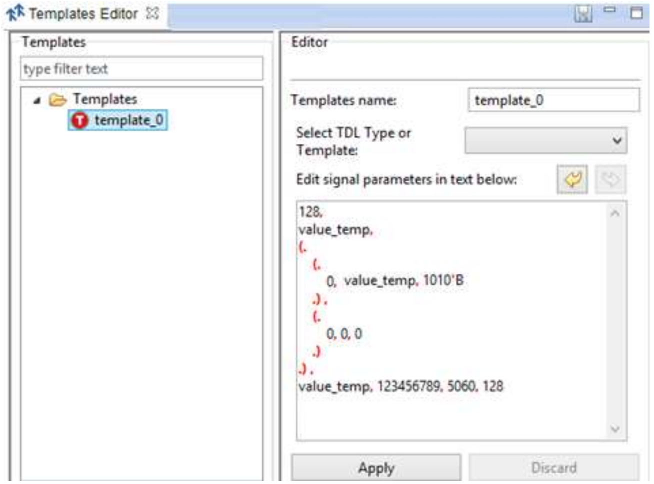

Fig. 8. Templates editor

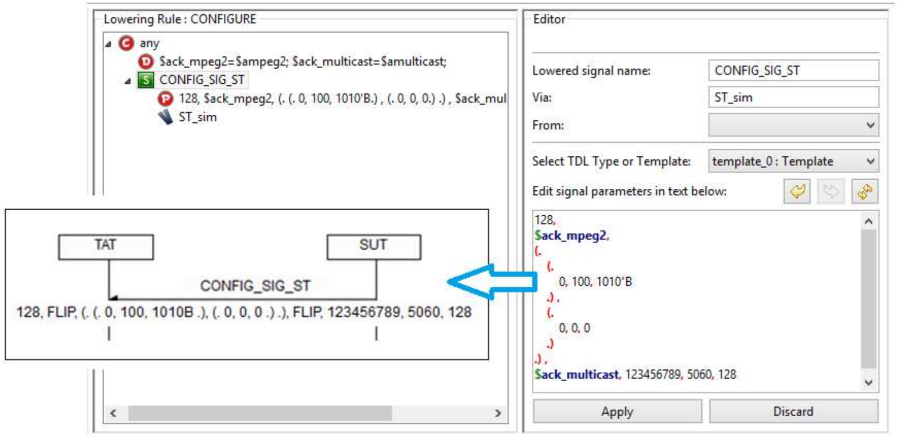

Fig. 9. Template used in signals editor

In both signal and template editors user can use variables – some values that are too big to remember of retype every time. On the Fig. 7 all the values are taken from variables. Variables can be selected in the middle of the lowering editor. There are different types of variables with different editors and checks. For example, the contents of variable "$gud_header" used in "soc_datato_req_s" detailed signal are shown on Fig. 10.

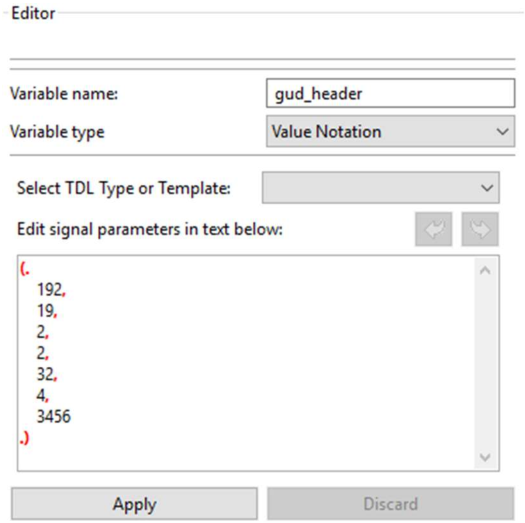

Fig. 10. Contents of the variable "$gud_header"

Variables can contain very complex structures and therefore greatly reduce expenses on creating detailed signals.

Overall process of selecting UCM-level signal, creating lowering rules and editing the resulting signal repeats for all UCM-level signals in the project.

4.3 Scenarios processing

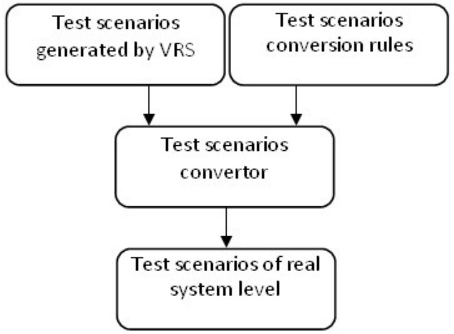

Fig. 11. Test scenarios conversion scheme

Implemented module of behavioral scenarios conversion takes as an input the concrete behavioral scenarios and specified rules of conversion and the output is behavioral scenarios of the real system level, which can be used for testing. Overall scheme of conversion is shown in Fig.11.

Detailing stage is based on the grammar of data structures conversion rules described in Fig. 5 and conversion algorithm. The specific feature of test automatic scenarios detailing to the level of real system is allow to storing of proved properties of the system obtained in process of abstract model verification.

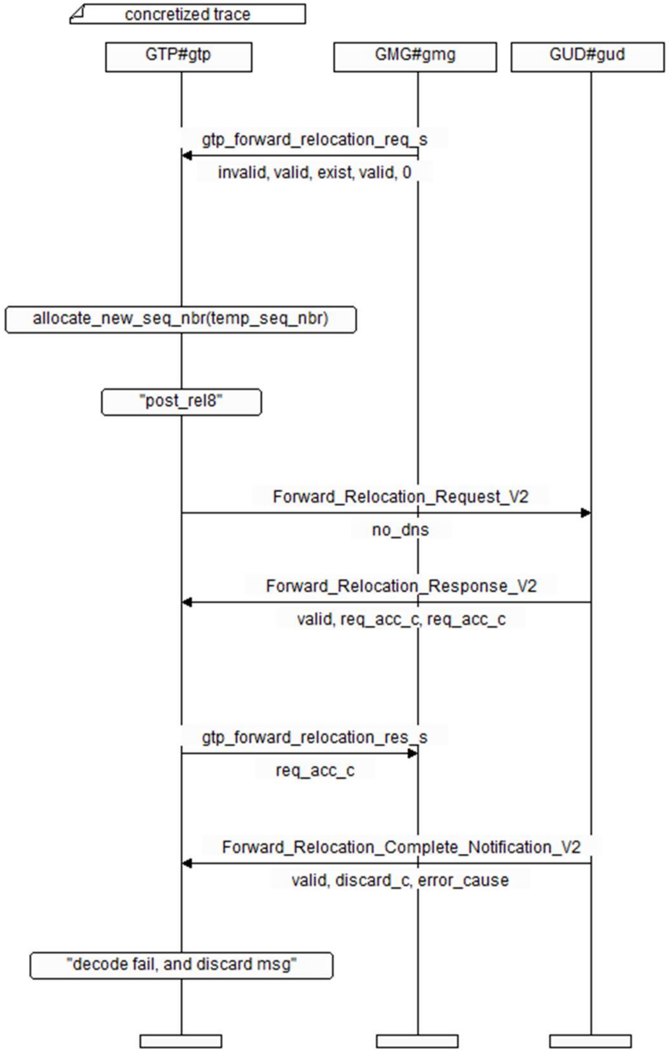

Fig. 12. Concrete scenario to be lowered

Based on the specified conversion rule each abstract signal in concrete behavioral scenario is processed. Signal parameters are matched to rule conditions and if the signal satisfies them, then it is converted into detailed form. Fig.12 shows concrete scenario, which will be processed.

In this scenario you can see 3 agents: "GTP#gtp", "GMG#gmg" and "GUD#gud". For example, we want to test an agent "GTP#gtp". On following trace it will be described as SUT.

Other agents (or whichever we choose in the settings of the trace preprocessing) are marked as TAT and joined together.

After data structures conversion, concrete signals are replaced with detailed signals specified in lowering rules. Once simple signal structure unfolds in very complex nested data while still maintaining its correctness. You can see the results on Fig. 13.

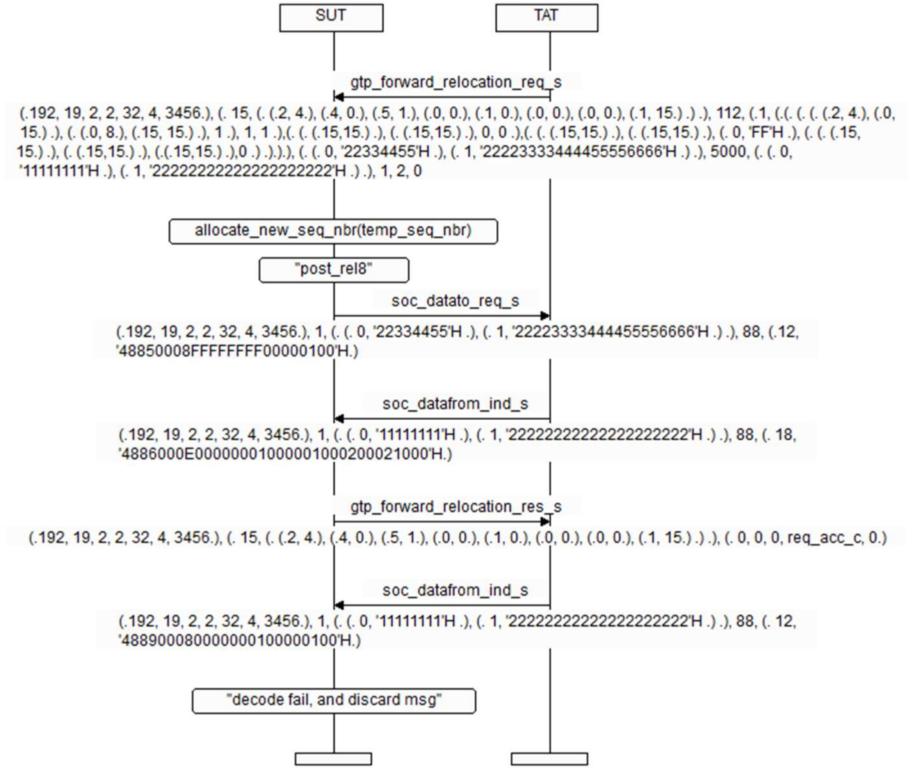

Fig. 13. Lowered trace with detailed signals

5. Conclusion

Proposed approach to behavioral scenarios generation based on formal models differs from existing approaches in using the process of automatic conversion of behavioral scenarios with abstract data structures into behavioral scenarios with detailed data structures used in real applications. Proposed language and overall scheme of this process allow automating of creation a set of covering behavioral scenarios.

In the scope of this work, the analyzer/editor for conversion rules of signals from abstract UCM model level into signals of real system level was developed and called Lowering Editor. It supports the following functionality: automatic binding between conversion rule and signal of UCM level, conversion rules correctness checking, templates usage, highlighting the syntax of conversion rules applying conditions specification, variables usage, libraries and external scripts (includes) usage, splitting UCM signal or action into several signals of real system in according to communication protocol, copy/paste/remove operations, import and export from/to storage file. Availability of described in the article features is able to make process of automatic conversion powerful and flexible for a different types of telecommunication applications.

Adding Lowering Editor into technology process of telecommunication software applications test automation allowed to exclude effort-consuming manual work in the cycle of test suite automated generation for industrial telecommunication applications, increase productivity of test generation in 25% and spread the properties proved on abstract models into generated code of executable test sets. Excluding of manual work allow to reduce human factor in testing process and guaranty quality of generated test suite based on verification results.

References

[1]. Model Driven Architecture- MDA (2007). Available at: http://www.omg.org/mda

[2]. Oscar Pastor, Sergio España, José Ignacio Panach, Nathalie Aquino. Model-Driven Development. Informatik Spektrum, vol. 31, no. 5, pp. 394-407 (2008)

[3]. Sami Beydeda , Matthias Book, Volker Gruhn. Model Driven Software Development.: Springer-Verlag Berlin Heidelberg, 464 p. (2005)

[4]. Robert V. Binder, Anne Kramer, Bruno Legeard. 2014 Model-based Testing User Survey: Results, 2014. Available at: http://model-based-testing.info/wordpress/wp-content/uploads/2014_MBT_User_Survey_Results.pdf

[5]. Buhr R. J. A., Casselman R. S. Use Case Maps for Object-Oriented Systems. Prentice Hall. 302 p. (1995)

[6]. 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. Problemy programuvannja [Problems of programming] (4), pp. 13–39 (2008).

[7]. I.Anureev, S.Baranov, D.Beloglazov, E.Bodin, P.Drobintsev, A.Kolchin, V. Kotlyarov, A. Letichevsky, A. Letichevsky Jr., V.Nepomniaschy, I.Nikiforov, S. Potienko, L.Pryima, B.Tyutin. Tools for supporting integrated technology of analysis and verification of specifications for telecommunication applications. Trudy SPIIRAN [SPIIRAS Proceedings], 2013, issue 26. pp. 349-383 (in Russian).

[8]. A. Kolchin, A. Letichevsky, V. Peschanenko, P. Drobintsev, V. Kotlyarov. Approach to creating concretized test scenarios within test automation technology for industrial software projects. Automatic Control and Computer Sciences, vol. 47, no. 7, pp. 433–442 (2013).

Преобразование абстрактных поведенческих сценариев в сценарии применимые для тестирования

П.Д. Дробинцев <drob@ics2.ecd.spbstu.ru>
В.П. Котляров <vpk@ics2.ecd.spbstu.ru >
И.В. Никифоров < i.nikiforov@ics2.ecd.spbstu.ru >
Н.В. Воинов <voinov@ics2.ecd.spbstu.ru >
И.А. Селин < ivanselin93@gmail.com>
Санкт-Петербургский политехнический университет Петра Великого, 195251, Россия, г. Санкт-Петербург, ул. Политехническая, дом 29

Аннотация. В данной статье рассмотрен подход детализации верифицированных тестовых сценариев для разрабатываемой программной системы без изменения семантики набора, то есть с сохранением корректности. Существующая проблема генерации тестов реальных приложений на основе верифицированных абстрактных сценариев, сгенерированных по поведенческой модели, решается на основе детализации абстрактных сценариев до уровня конкретных состояний, транзакций, протоколов и сигналов. Поскольку характерной особенностью рассматриваемых абстрактных моделей является символьное представление поведенческих сценариев, то их детализация происходит в два этапа. На первом этапе – этапе конкретизации, символьные параметры сигналов получают конкретные значения, образуя тем самым конкретные поведенческие сценарии. На втором этапе – этапе собственно детализации, конкретные абстрактные сценарии необходимо представлять в виде структур данных, формы представления и значения которых содержат всю необходимую информацию для обмена с реальными приложениями. Полученные таким образом детальные сценарии предназначены для генерации исполнимых тестовых наборов для информационных и управляющих систем. В работе предложен инструментарий детализации тестовых сценариев, позволяющий не только описать реальные сигналы, но и детализировать протоколы обмена сигналами. В его состав входит Lowering editor, позволяющий описывать правила преобразования сигналов в соответствии с приведенной разработанной грамматикой правил преобразований, Signals editor, используемый для удобного описания сложных структур сигналов и Templates editor, позволяющий однократно описывать типовые структуры. Приведён пример процесса преобразования от абстрактных структур данных к детализированным, использующимся при тестировании целевого кода.

Ключевые слова: model approach; model verification; test mapping

DOI: 10.15514/ISPRAS-2016-28(3)-9

Для цитирования: П.Д. Дробинцев, В.П. Котляров, И.В. Никифоров, Н.В. Воинов, И.А. Селин. Преобразование абстрактных поведенческих сценариев в сценарии применимые для тестирования. Труды ИСП РАН, том 28, вып. 3, 2016 г. стр. 145-160 (на английском). DOI 10.15514/ISPRAS-2016-28(3)-9

Список литературы

[9]. Model Driven Architecture – MDA (2007), доступно по ссылке: http://www.omg.org/mda

[10]. Oscar Pastor, Sergio España, José Ignacio Panach, Nathalie Aquino. Model-Driven Development. Informatik Spektrum, Volume 31, Number 5, pp. 394-407 (2008)

[11]. Sami Beydeda , Matthias Book, Volker Gruhn. Model Driven Software Development.: Springer-Verlag Berlin Heidelberg, 464 p. (2005)

[12]. Robert V. Binder, Anne Kramer, Bruno Legeard. 2014 Model-based Testing User Survey: Results, 2014, доступно по ссылке: http://model-based-testing.info/wordpress/wp-content/uploads/2014_MBT_User_Survey_Results.pdf

[13]. Buhr R. J. A., Casselman R. S.: Use Case Maps for Object-Oriented Systems. Prentice Hall. 302 p. (1995)

[14]. 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, pp. 13–39 (2008).

[15]. Ануреев И.С., Баранов С.Н., Белоглазов Д.М., Дробинцев П.Д., Колчин А.В., Котляров В.П., Летичевский А.А., Летичевский А.А. мл., Непомнящий В.А., Никифоров И.В., Потиенко С.В., Прийма Л.В., Тютин Б.В., Бодин Е.М. Средства поддержки интегрированной технологии для анализа и верификации спецификаций телекоммуникационных приложений. Труды СПИИРАН. 2013, вып. 26, стр. 349-383.

[16]. A. Kolchin, A. Letichevsky, V. Peschanenko, P. Drobintsev, V. Kotlyarov. Approach to creating concretized test scenarios within test automation technology for industrial software projects. Automatic Control and Computer Sciences, vol. 47, no. 7, pp. 433–442 (2013)