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

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.

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.

```
    GTP#gtp
    gtp_forward_relocation_req_s
    seq_nbr, ip1, ip2, tid, islntra

    GMG#gmg
```

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 "islntra" are substituted with values "invalid", "valid", "exist", "valid" and "0". Fig. 4 contains concrete behavioral scenario.

```
    GTP#gtp
    gtp_forward_relocation_req_s
    invalid, valid, exist, valid, 0

    GMG#gmg
```

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

### 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.

#### 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 "->"
ConditionContent ::= ConditionContent | ConditionCondition
ConditionCondition ::= Condition | ConditionCondition
Condition ::= ConditionElement | ConditionCondition
ConditionElement ::= ConditionElement | ConditionElement Condition
ConditionElement ::= ConditionElement Element
ConditionElement ::= ConditionElement Do | ConditionElement Signal
ConditionElement ::= ConditionElement View | ConditionElement Via
ConditionElement ::= ConditionElement Instance | ConditionElement Instance
ConditionElement ::= ConditionElement Instance | ConditionElement Instance
ConditionElement ::= ConditionElement Instance | ConditionElement Instance
```

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.

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).

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.
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

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.
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.

**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 cycle of test suite automated generation for industrial telecommunication applications.


In this paper, we propose an approach to generating test scenarios for telecommunication applications by using a model driven approach. The approach involves converting abstract behavioral scenarios into concrete test scenarios that can be used for testing real-time systems. The key steps in this process include:

1. Identifying the requirements and functional requirements of the telecommunication system.
2. Generating abstract behavioral scenarios based on the requirements.
3. Converting the abstract scenarios into concrete scenarios that can be used for testing.
4. Automating the generation process to improve efficiency.
5. Testing the generated scenarios to ensure they cover all necessary aspects.

The proposed approach is effective in reducing manual effort in testing processes and increasing productivity. It also allows for automated generation of test scenarios, which is particularly useful in complex telecommunication systems.
Список литературы


