Моделирование и анализ информационных систем. Т. 22, №6 (2015), с. 750–762
Modeling and Analysis of Information Systems. Vol. 22, No 6 (2015), pp. 750–762

DOI: 10.18255/1818-1015-2015-6-750-762

UDC 004.4'2

Model Oriented Approach for Industrial Software Development

Drobintsev P. D., Kotlyarov V. P., Voinov N. V., Nikiforov I. V.

Received October 21, 2015

The article considers the specifics of a model oriented approach to software development based on the usage of Model Driven Architecture (MDA), Model Driven Software Development (MDSD) and Model Driven Development (MDD) technologies. Benefits of this approach usage in the software development industry are described. The main emphasis is put on the system design, automated code generation for large systems, verification, proof of system properties and reduction of bug density. Drawbacks of the approach are also considered. The approach proposed in the article is specific for industrial software systems development. These systems are characterized by different levels of abstraction, which is used on modeling and code development phases. The approach allows to detail the model to the level of the system code, at the same time store the verified model semantics and provide the checking of the whole detailed model. Steps of translating abstract data structures (including transactions, signals and their parameters) into data structures used in detailed system implementation are presented. Also the grammar of a language for specifying rules of abstract model data structures transformation into real system detailed data structures is described. The results of applying the proposed method in the industrial technology are shown.

The article is published in the authors' wording.

Keywords: model oriented approach; multilevel software models; model specification by control flow and data flow; model verification; substitutions saving the correctness of proved properties

For citation: Drobintsev P. D., Kotlyarov V. P., Voinov N. V., Nikiforov I. V., "Model Oriented Approach for Industrial Software Development", *Modeling and Analysis of Information Systems*, **22**:6 (2015), 750–762.

On the authors:

Drobintsev Pavel Dmitrievich, orcid.org/0000-0003-1116-7765, PhD,
Peter the Great St. Petersburg Polytechnic University,
Polytechnicheskaya str., 29, St.Petersburg, 195251, Russia, e-mail: drob@ics2.ecd.spbstu.ru

Kotlyarov Vsevolod Pavlovich, orcid.org/0000-0003-3973-5218, PhD,
Peter the Great St. Petersburg Polytechnic University,
Polytechnicheskaya str., 29, St.Petersburg, 195251, Russia, e-mail: vpk@spbstu.ru

Voinov Nikita Vladimirovich, orcid.org/0000-0002-0140-1178, PhD,
Peter the Great St. Petersburg Polytechnic University,
Polytechnicheskaya str., 29, St.Petersburg, 195251, Russia, e-mail: voinov@ics2.ecd.spbstu.ru

Nikiforov Igor Valerievich, orcid.org/0000-0003-0198-1886, PhD,
Peter the Great St. Petersburg Polytechnic University,
Polytechnicheskaya str., 29, St.Petersburg, 195251, Russia, e-mail: igor.nikiforovv@gmail.com

1. Model based technologies

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 creation of multilevel model of application during design process. A set of possible models transformations is presented in Fig. 1. 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.

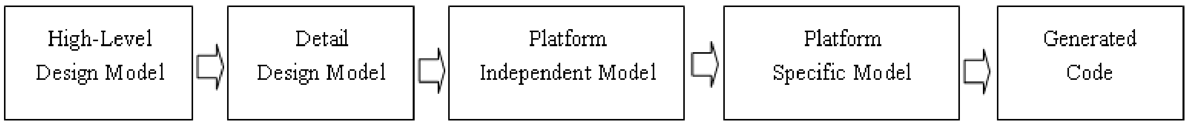


Fig. 1. Designing multilevel model of application

Statistics collected in companies which are using such approaches shows [4] 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 behavior is complexity of system testing for big industrial projects, which is based on huge efforts spent on quality guarantying [5]. To resolve this problem software developing companies are trying to reduce efforts for tests creation and simplify tests execution process. Usually reduction of testing efforts is linked to communication with customers because only customer of software has deep knowledge about domain specifics and model oriented approach helps to simplify such communications.

Researchers also consider that more than 80% [4] of model-oriented approaches use graphical notations, which simplifies working with formal notations for developers. Requirements for testers and customer representatives knowledge 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 [4]:

- increasing productivity and reduction of efforts on development;
- systematic reuse of verified templates and solutions which leads to reduction of bugs density in generated code;
- analyzing and proving formal models properties on early stages of design.

Among drawbacks of the approach the following can be listed:

- different levels of detailing in multilevel formal model and real generated code which may lead to distortion of verified semantics during model detailing;
- complexity (impossibility in some cases) of aggregate proving the multilevel detailed model properties;
- complexity of multi criteria optimization while selecting balanced architecture of software application.

2. Drawbacks of models usage

The main drawback of formal models using in software products development is high level of model abstraction in comprising with a real system. At the stage of model design developers tend to specify only major behavior scenarios and data structures which affect the behavior, ignoring the implementation details. Usage of formal models on this stage allows to prove the correctness of system behavior in accordance with specifications, miscellaneous system's properties and to generate a set of test scenarios providing complete coverage for specified criteria.

As a model of the system and its implementation in the code vary significantly, the semantics of test scenarios generated from abstract model may differ from corresponding behavior observed in the real system. Therefore automatic check of system functioning correctness is impossible.

There are two ways to solve this issue.

The first one is developing of a detailed model which is as close to system program implementation as possible. In this case it is impossible to provide check of complete detailed model of industrial system (even of medium complexity) due to limitations of modern verification toolsets.

The second one is creating an abstract model of such complexity which does not prevent applying toolsets for proving behavioral properties. Further the abstract model can be detailed to the level of real system in such a way that proved properties will be spread on the detailed model. This method satisfies model based software development technology when applied iteratively and guarantees storing proved system properties up to code level.

When control flow is being detailed traditional elements of model control flow structuring can be used. Model fragments which shall be detailed are relocated into separate structural element (for example, an instance of class method). Then its analysis and formalization of its behaviors, which include specifying fragments of alternative and concurrent behavior, fragments of behavior limited by timer, fragments of behavior specific for exceptions and interruptions can be performed.

When data flow is being detailed formalization of new data structures, signals and transactions is performed. Each data structure can be represented by several nested structures of lower level. Signals in the system can be separated into several compound signals and actions of real system. New transactions can be added to the system to provide data consistency. Also detailing of one system signal into complete communication protocols between components becomes possible. This means that a communication protocol can be represented by only one signal on the abstract level while in the real code this protocol can be specified by a set of incoming and outgoing signals.

3. Levels of behavioral models development

One of high level languages for system formal model specification is Use Case Maps (UCM) [6, 8]. 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 [8]. But metadata does not allow detailing data flow to more detailed levels. That's why for creating detailed behaviors it is proposed to use the following vertical levels of abstractions during behavioral models development (Fig. 2).

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 [9] is used to convert symbolic test scenarios. On this step

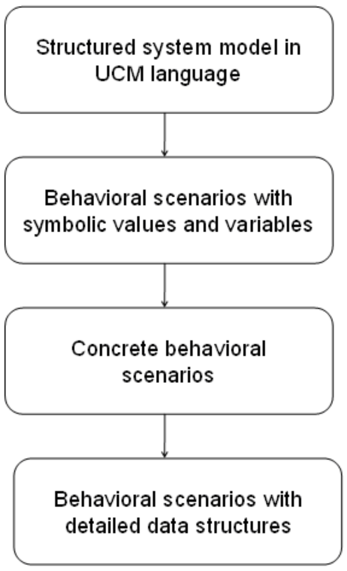
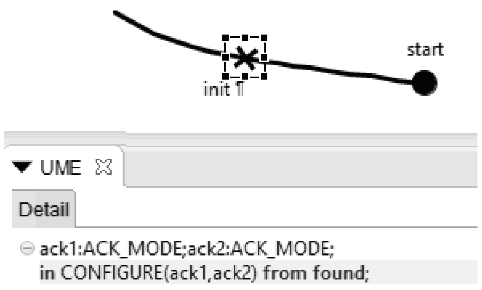

Structured system model in UCM language
Behavioral scenarios with symbolic values and variables
Concrete behavioral scenarios
Behavioral scenarios with detailed data structures

Fig. 2. Abstraction levels during developing behavioral scenarios

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.

4. Data structures conversion

In behavioral scenarios data structures are mainly used in signals parameters. Consider an example of converting signal structure of UCM level into detailed structures of real system for the signal "CONFIGURE".

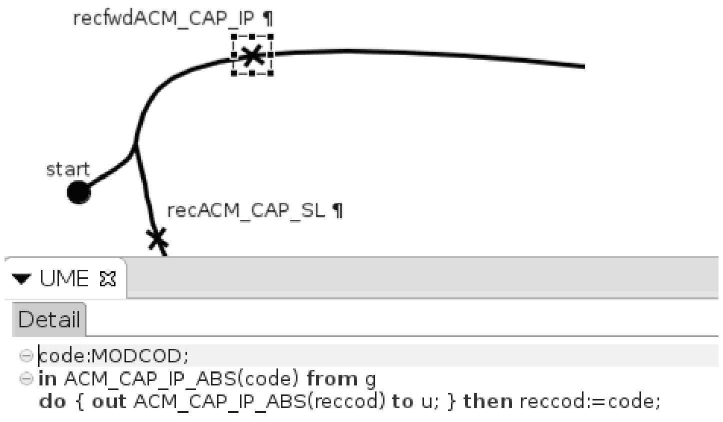


```
ack1:ACK_MODE;ack2:ACK_MODE;
in CONFIGURE(ack1,ack2) from found;
```

Fig. 3. Description of the "CONFIGURE" signal
in metadata of the "init" UCM element

In UCM model the element "init" contains metadata with the signal "CONFIGURE" and two signal parameters of UCM level: "ack1" and "ack2". Fig. 3 contains metadata of the UCM element "init" including description of a signal of UCM level.

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".



```
code:MODCOD;
in ACM_CAP_IP_ABS(code) from g
do { out ACM_CAP_IP_ABS(reccod) to u; } then reccod:=code;
```

Fig. 4. Description of the "ACM_CAP_IP_ABS" signal in metadata of the "recfwdACM_CAP_IP" UCM element

As an example in UCM model the element "recfwdACM_CAP_IP" contains metadata with the outgoing signal "ACM_CAP_IP_ABS" and the signal parameter "reccod" of UCM level. This signal shall be sent after the signal "ACM_CAP_IP_ABS" with the parameter "code" received from the agent "g". Fig. 4 contains metadata of the UCM element "recfwdACM_CAP_IP" including description of a signal of UCM level. The outgoing signal can be used only inside of "do" section as reaction of the system on some event.

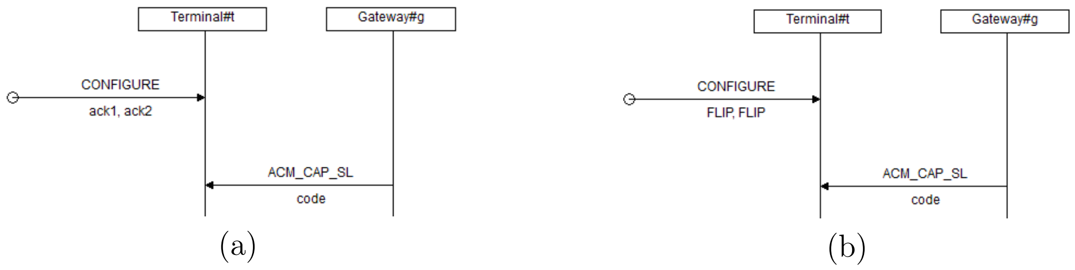
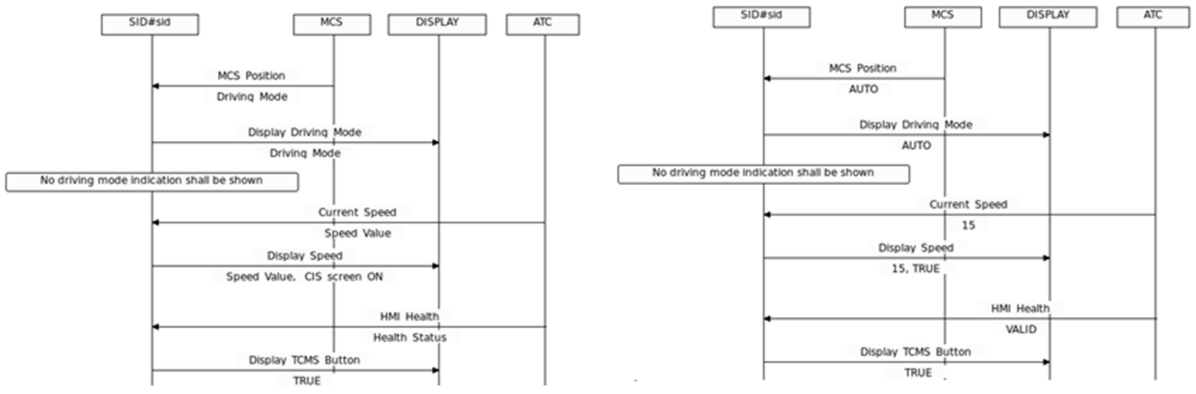


Fig. 5. Symbolic (a) and concrete (b) test scenarios containing the signal "CONFIGURED"

Based on high level UCM model symbolic behavioral scenarios are generated containing data structures described in metadata of UCM elements. Fig. 5(a) contains symbolic test scenario where the agent "Terminal#t" receives the signal "CONFIGURED". In symbolic scenarios actual names of UCM model agents specified in metadata are used. For example, the agent "Gateway#g" is the source of the signal "ACM_CAP_SL" and the agent "Terminal#t" is the destination. While the source of the signal "CONFIGURE" is outside the system.

Symbolic behavioral scenario is input data for concretization module which substitutes symbolic parameters with concrete values. In current example the parameters "ack1, ack2" are substituted with values "FLIP" and "FLIP". Fig. 5(b) contains concrete behavioral scenario. Fig. 6 contains another example of concretization where integer parameters are substituted together with parameters of string type. For example, the symbolic parameter "Speed Value" in the signals "Current Speed" and "Display Speed" (Fig. 6(a)) is concretized with value "15" (Fig. 6(b)).



(a) (b)

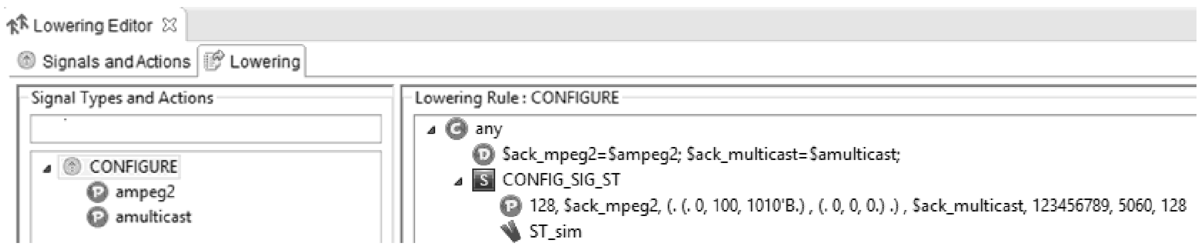
Fig. 6. Symbolic (a) and concrete (b) test scenarios containing string and integer parameters

Note that after concretization interacting agents are not changed in any way. To convert concrete data structures into detailed structures a developer shall specify the rules of structures conversion: for each signal of UCM model a corresponding conversion condition (Lowering Condition) and detailed signal (Lowered Signal) are specified.

To keep proved system properties there are following limitations on the conversion:

- rules which allow separating constants into several independent parts (sets of variables) are prohibited;
- separating fields of variables values is prohibited;
- converting abstract signal into a protocol if this protocol is not represented by verified template is prohibited;
- only constant template values or values obtained at concretization step are allowed;
- violating consistent communication protocol is prohibited.

Fig. 7 contains the rule for converting the signal "CONFIGURED" into the signal "CONFIGURED_SIG_ST" for all occurrences of this signal in test scenario. This condition is specified by the keyword "any" in the rule.



```
CONFIGURE
    ampeg2
    amulticast

any
    $ack_mpeg2=$ampeg2; $ack_multicast=$amulticast;
    CONFIG_SIG_ST
        128, $ack_mpeg2, (. (. 0, 100, 1010'B.) , (. 0, 0, 0.) .) , $ack_multicast, 123456789, 5060, 128
        ST_sim
```

Fig. 7. Rule for converting the signal "CONFIGURED"
into the signal "CONFIG_SIF_ST"

Specification of conversion rules is based on the grammar of conversion language. Common view of the grammar for converting signals of abstract level into detailed level in BNF form is shown in Fig. 8.

```
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. 8. Grammar of the conversion rules language

Based on the specified conversion rule each abstract signal in concrete behavioral scenario is processed and in case the signal satisfies to a conversion rule it is converted into detailed signal. Fig. 9 contains executable scenario with the detailed signal "CONFIG_-SIG_ST" which can be used for testing. Note that on this stage system agents are joined into two instances – TAT and SUT, which is required for testing process.

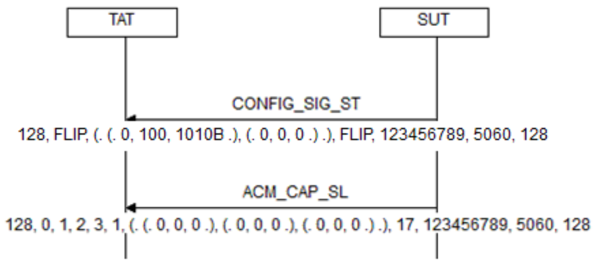


Fig. 9. The detailed signal "CONFIG_SIG_ST" of the real system

To exclude limitations on conversion of signals with templates usage the following techniques can be used. In case of some particular signal is converted into a set of signals (protocol) which are used as parameters variables verified on previous phase then combined conditions shall be used. In Fig. 10 (a) a signal with four parameters is presented. Consider that based on a template the signal shall be substituted into a set of signals.

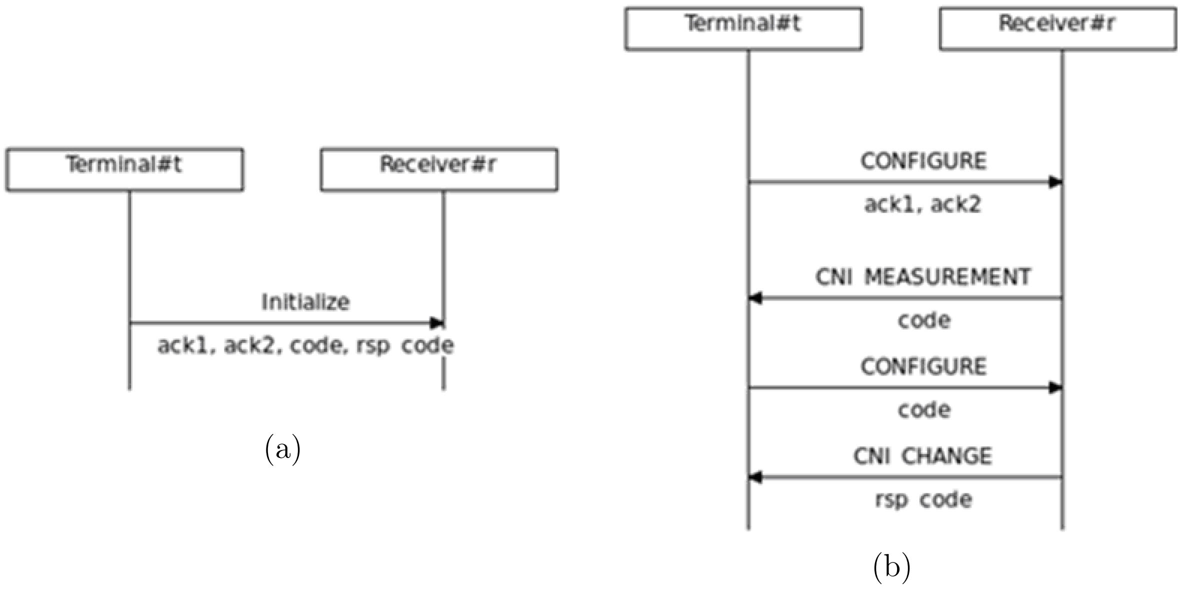


Fig. 10. Initial signal before (a) and after (b) conversion

Fig. 10 (b) contains a diagram with substituted signal. To verify correctness of such substitution a special filters shall be added into target code of application and test.

Another case when signal parameter is separated into parameters of two signals as presented in Fig. 11 (a,b).

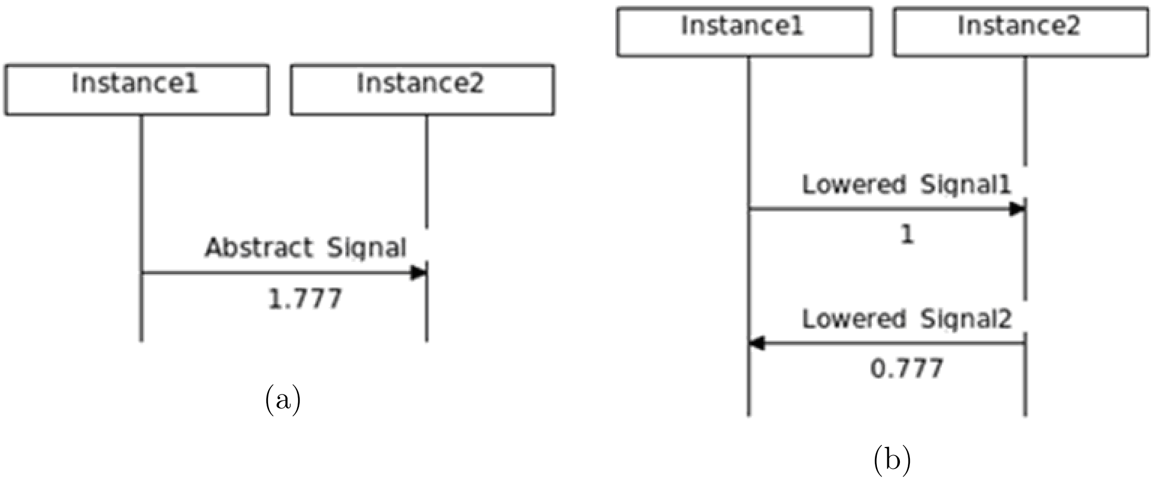


Fig. 11. Initial signal before (a) and after (b) parameters separation

The same solution with filter in target code of application and test can be used. The filter shall check that correctness of the model was not broken via generation of ranges for parameters of separated signals.

Usage of approach with filters allows to raise limitations of lowering conversion connected to maintenance of model correctness.

5. Overall scheme of conversion

Implemented module of behavioral scenarios conversion takes the concrete behavioral scenarios and specified rules of conversion as an input 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. 12.

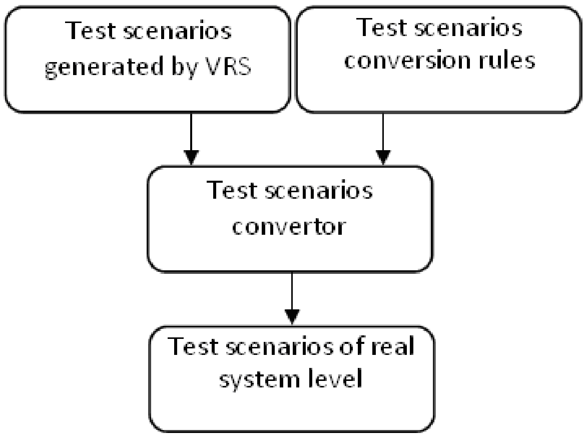

Test scenarios generated by VRS
Test scenarios conversion rules
Test scenarios convertor
Test scenarios of real system level

Fig. 12. Test scenarios conversion scheme

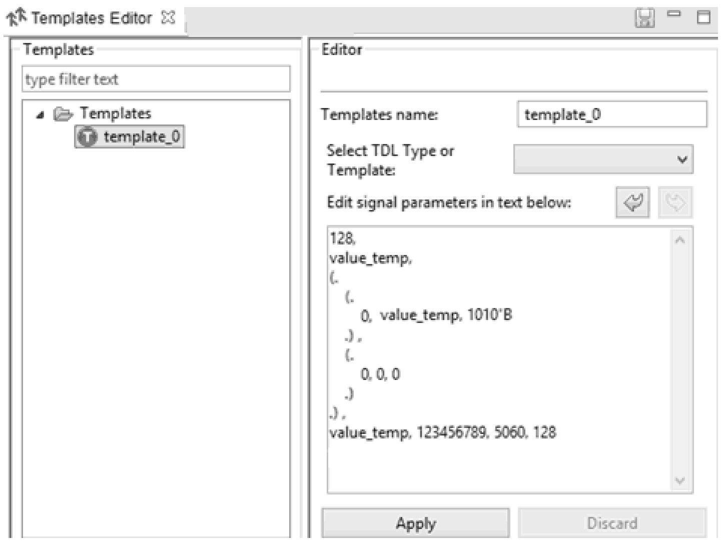
Detailing stage is based on the grammar of data structures conversion rules described in Fig. 8 and conversion algorithm. The specific feature of test automatic scenarios detailing to the level of real system is storing of proved properties of the system obtained in process of abstract model verification.

6. Templates

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. 13 the template "template_0" is shown which contains detailed data structures inside and the dummy value "value_temp" which shall be changed to concrete values when template is used.

When a template of data structure is ready, it can be used for creation of conversion rules. Fig. 14 represents usage of the template "template_0" with substituted concrete



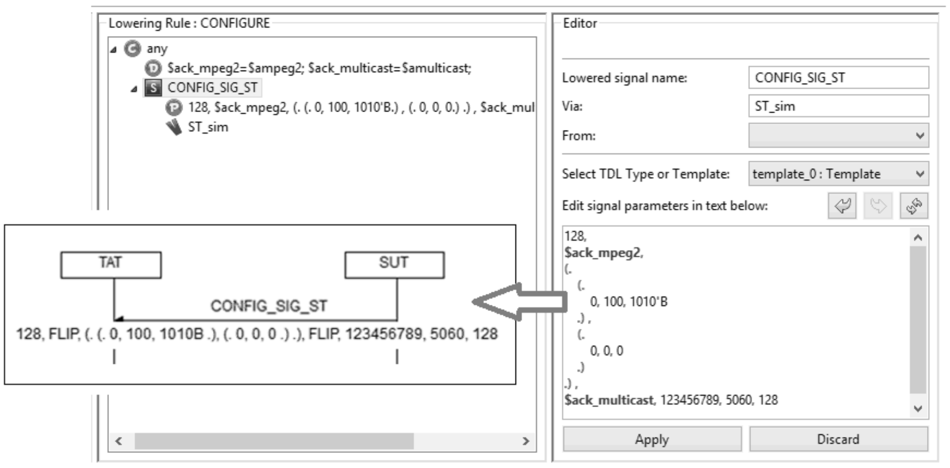
Templates name: template_0

Select TDL Type or Template:

Edit signal parameters in text below:

```
128,
value_temp,
(.
   (.
      0, value_temp, 1010'B
   .) ,
   (.
      0, 0, 0
   .)
.) ,
value_temp, 123456789, 5060, 128
```

Fig. 13. Example of the template "template_0"



$ack_mpeg2=$ampeg2; $ack_multicast=$amulticast;
CONFIG_SIG_ST
128, $ack_mpeg2, (. (. 0, 100, 1010'B.) , (. 0, 0, 0.) .) , $ack_mul
ST_sim

128, FLIP, (. (. 0, 100, 1010B .), (. 0, 0, 0 .) .), FLIP, 123456789, 5060, 128

Lowered signal name: CONFIG_SIG_ST

Via: ST_sim

From:

Select TDL Type or Template: template_0 : Template

Edit signal parameters in text below:

```
128,
$ack_mpeg2,
(.
   (.
      0, 100, 1010'B
   .) ,
   (.
      0, 0, 0
   .)
.) ,
$ack_multicast, 123456789, 5060, 128
```

Fig. 14. Applying the template "template_0" for the signal "CONFIG_SIG_ST"

values of signal parameters instead of the dummy value "value_temp" which then will appear in behavioral MSC scenario.

Note that in conversion rules editor complex data structures are represented with formatted text which makes parameters and values more readable than in linear representation of MSC scenario.

Templates usage reduces efforts on creation and coding complex data structures on 25%-30% and reduces possibility of introducing extra bugs because of user inaccuracy.

7. 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 LoweringEditor. 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 according to communication protocol, copy/paste/remove operations, import and export from/to storage file. Features described in the article make process of automatic conversion powerful and flexible for different types of telecommunication applications. Adding LoweringEditor 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 allows to reduce human factor in testing process and guaranty quality of generated test suite based on verification results.

References

[1] "Model Driven Architecture - MDA", *http://www.omg.org/mda*, 2007.

[2] Pastor O. et al., "Model-Driven Development", *Informatik Spektrum*, **31**:5 (2008), 394–407.

[3] Beydeda S. , Book M., Gruhn V., "Model Driven Software Development", *Springer-Verlag Berlin Heidelberg*, 2005, 464.

[4] Binder R.V., Kramer A., Legeard B., "2014 Model-based Testing User Survey: Results", *http://model-based-testing.info/wordpress/wp-content/uploads/2014_MBT_User_Survey_Results.pdf*, 2014.

[5] Fenton N.E., Ohlsson N., "Quantitative analysis of faults and failures in a complex software system", *Software Engineering, IEEE Transactions on*, 2000, № 8.

[6] Buhr R. J. A., Casselman R. S., "Use Case Maps for Object-Oriented Systems", *Prentice Hall*, 1995, 302.

[7] Anureev I. et al., "Tools for supporting integrated technology of analysis and verification of specifications for telecommunication applications", *SPIIRAN works*, **1** (2013), 28.

[8] Letichevsky A.A. et al., "Insertion modeling in distributed system design", *Problems of programming*, 2008, 13–39.

[9] Kolchin A. et al., "Approach to creating concretized test scenarios within test automation technology for industrial software projects", *Automatic Control and Computer Sciences, Allerton Press, Inc.*, **47**:7 (2013), 433–442.

DOI: 10.18255/1818-1015-2015-6-750-762

Особенности применения модельно-ориентированного подхода при разработке промышленных приложений

Дробинцев П. Д., Котляров В.П., Воинов Н.В., Никифоров И.В.

получена 21 октября 2015

В статье рассмотрены особенности применения технологий разработки программных систем на основе модельно-ориентированного подхода: Model Driven Software Development (MDSD), Model Driven Architecture (MDA) и Model Driven Development (MDD). Описаны преимущества использования подходов в промышленности. Основной акцент сделан на проектирование систем, автоматическую генерацию кода больших систем, верификацию, доказательство свойств систем и уменьшение плотности ошибок. Приведены недостатки использования данного подхода, одним из которых является различная степень детальности модели и реальной реализованной системы на языке программирования. В работе предлагается подход, характерный для систем, имеющих многоуровневое представление, связанное с детализацией функциональности приложения до уровня, на котором осуществляется прямая генерация корректного кода. Подход позволяет детализировать модель до уровня реального кода системы, при этом сохранить проверенную семантику модели и обеспечить проверку всей детальной модели. Детализация проводится как по потоку управления, так и по потоку данных. Представлены шаги по преобразованию абстрактных структур данных (в том числе транзакций, сигналов и их параметров) в структуры данных, используемых в реализации систем. Приведена грамматика языка задания правил преобразования структур данных абстрактной модели в детальные структуры данных реальной системы и общая схема преобразования. Приведены результаты применения предложенного метода в промышленной технологии.

Статья представляет собой расширенную версию доклада на VI Международном семинаре "Program Semantics, Specification and Verification: Theory and Applications", Казань, 2015.

Статья публикуется в авторской редакции.

Ключевые слова: модельно-ориентированный подход; многоуровневые модели приложения; спецификация моделей по управлению и структурам данных; верификация моделей; подстановки, сохраняющие корректность доказанных свойств

Для цитирования: Дробинцев П. Д., Котляров В.П., Воинов Н.В., Никифоров И.В., "Особенности применения модельно-ориентированного подхода при разработке промышленных приложений", *Моделирование и анализ информационных систем*, **22**:6 (2015), 750–762.

Об авторах:
Дробинцев Павел Дмитриевич, orcid.org/0000-0003-1116-7765, канд. техн. наук, доцент,
Санкт-Петербургский политехнический университет Петра Великого,
ул. Политехническая, 29, г. Санкт-Петербург, 195251 Россия, e-mail: drob@ics2.ecd.spbstu.ru

Котляров Всеволод Павлович, orcid.org/0000-0003-3973-5218, канд. техн. наук, профессор,
Санкт-Петербургский политехнический университет Петра Великого,
ул. Политехническая, 29, г. Санкт-Петербург, 195251 Россия, e-mail: vpk@spbstu.ru

Воинов Никита Владимирович , orcid.org/0000-0002-0140-1178, канд. техн. наук, доцент,
Санкт-Петербургский политехнический университет Петра Великого,
ул. Политехническая, 29, г. Санкт-Петербург, 195251 Россия, e-mail: voinov@ics2.ecd.spbstu.ru

Никифоров Игорь Валерьевич , orcid.org/0000-0003-0198-1886, канд. техн. наук, доцент,
Санкт-Петербургский политехнический университет Петра Великого,
ул. Политехническая, 29, г. Санкт-Петербург, 195251 Россия, e-mail: igor.nikiforovv@gmail.com