The Application of Coloured Petri Nets to Verification of Distributed Systems Specified by Message Sequence Charts[1]

S.A. Chernenok <chernenoksergey@gmail.com>,
V.A. Nepomniaschy <vnep@iis.nsk.su>,
A.P. Ershov Institute of Informatics Systems of the Siberian Branch of the RAS,
6 Lavrentjev pr., Novosibirsk, 630090, Russian Federation

Abstract. The language of message sequence charts (MSC) is a scenario-based specification language widely used at the design stage to describe the interaction of components in distributed systems. However, the existing methods and tools for validation of MSC diagrams are underdeveloped. They have such limitations as a small set of supported diagram elements, restrictions on the behavior of elements and on the set of analyzed properties. This paper describes a method for translation of MSC diagrams into coloured Petri nets (CPN), which is applied to the property analysis and verification of these diagrams. The translation method consists of three main stages: generation of the MSC internal representation called a partial order graph, processing of the partial order graph and translation of the graph into CPN. The result of the translation is a hierarchical coloured Petri net in a format compatible with the known CPN Tools system. Besides the basic elements of the MSC standard, the considered set of diagram elements includes diagram elements with data (messages, local actions and conditions with data), the elements of UML sequence diagrams (synchronous messages, combined fragments) and compositional MSC diagrams (partial-defined messages). The translator from MSC diagrams into CPN is implemented on the basis of the translation method. The properties of the resulting CPN are analyzed and verified using the system CPN Tools and the CPN verifier based on the SPIN tool. If an analyzed property is violated during the verification process and a counterexample is generated, then an error can be localized inside the verified MSC. To localize the error, an MSC trace leading to a broken state is constructed, which is a sequence of diagram events and variable states of each process. The application of the translation method and tools for analysis and verification is illustrated with an example of Alternating Bit Protocol (ABP).

Keywords: specification; translation; verification; distributed systems; communication protocols; message sequence charts; UML sequence diagrams; coloured Petri nets

DOI: 10.15514/ISPRAS-2015-27(3)-14

For citation: Chernenok S.A., Nepomniaschy V.A.. The Application of Coloured Petri Nets to Verification of Distributed Systems Specified by Message Sequence Charts. *Trudy*

[1] This work is partially supported by RFBR grant 14-07-00401

ISPRAN/Proc. ISP RAN, 2015, vol. 27, issue 3, pp. 197-218. DOI: 10.15514/ISPRAS-2015-27(3)-14.

1. Introduction

One of the major issues that arise in the process of software development is a validation problem. Over the last few years, a large number of methods and tools have been developed for the analysis and validation of systems at the stages of their design and development. However, these methods are not so powerful as compared to the formal methods of software analysis and verification. Therefore, an important goal of software validation is to improve the existing validation methods used in practice by means of integration of well-studied analysis and verification formalisms.

The scenario-based languages are a popular way to describe program specifications at the design stage of software development. They have an expressive graphical representation and are easy to use. One of the most popular scenario-based languages is the language of Message Sequence Charts (MSC) standardized by the ITU-T [1]. MSC diagrams are widely used for specification of communication protocols. The sequence diagrams of the UML standard (UML SD) [2], inspired by the MSC, made the interaction diagrams popular in the wide fields of software development. The application area of MSCs includes documentation, requirements specification, simulation, test case generation, etc.

Triggered by the increasing popularity of MSC diagrams several new dialects and extensions of the MSC language emerged. One of the important extensions increasing the expressive power of the MSC is Compositional MSC diagrams (CMSC) [3, 4]. The use of CMSC diagrams allows us to cope with the restrictions of the MSC language in order to describe a certain type of interactions, such as sliding window protocols.

It is known that at the early stages of software development the cost of errors is the highest. Therefore, the program models specified by MSCs should be valid and error-free. In practice there are tools for analysis and validation of MSC specifications. Among them are the following.

The UBET system [5, 6] can check the race conditions and timing violations for a created MSC diagram. The system also provides an automatic test case generation feature and a conversion of MSCs into the Promela language code. UBET only supports the elements of the basic MSC diagrams.

The software tools Cinderella MSC [7] and IBM Rational / Telelogic Tau [8] are visual modeling tools for analysis, specification and testing of systems described by the interaction diagrams. The system [7] supports the generation of MSC diagrams from a user application, the generation of test cases from MSCs, and the conversion of diagrams into other analysis systems. The toolkit [8] allows one to create program models based on the UML sequence diagrams, to perform the automated error checking of the UML SD syntax and semantics, and to convert UML SD diagrams into the SDL modeling language for further analysis. These tools are limited by a small set of available verified properties and do not support many of the diagram elements.

The PragmaDev analyzer [9] allows one to analyze the specific properties of MSC diagrams (analysis and comparison of MSC specifications and analysis of time properties) and also some temporal logic properties defined in Property Sequence Charts. The project is under development and currently only a part of MSC elements is supported.

The problem of analysis and verification of interaction diagrams is investigated by several authors.

Papers [10, 11, 12] describe the modeling of UML SD diagrams using high-level Petri nets. The paper [10] deals with the translation of UML SD diagrams into CPN. This paper describes the translation rules for a limited set of diagram elements and element compositions. Also, structural restrictions are imposed on the message elements (i.e. only the synchronous messages and strict sequential composition between structural fragments are considered) and on the interpretation of conditions. The paper [11] provides an extension of SD diagrams for the purpose of simulation and analysis of embedded systems. The authors describe formal translation rules for most standard elements. But some composition constructs are not considered. The paper [12] provides the semantics of SD diagrams in terms of extended Petri nets. This work deals with most of the UML SD standard elements except the elements for scenario composition. Note that the translation of the elements `strict`, `break` and `critical` is not considered in the papers [10, 11, 12].

Papers [13, 14] present the translation of UML SD diagrams into the input languages of the verifiers SPIN [15] and NuSMV. The authors consider most of the diagram elements, including the combined fragments of UML SD. References and high-level MSC diagrams are not considered.

Note that most of the related work imposes restrictions on the diagram elements that do not allow one to specify and analyze the distributed systems with independent components. In addition, these papers do not consider messages and local actions with dynamic data. The translation of CMSC diagram elements into Petri nets in the papers is not considered.

Thus, analysis and verification of MSC and UML SD diagrams is an urgent problem. Our paper is aimed at investigation of this problem.

This paper describes a method for analysis and verification of MSC diagrams of distributed systems based on the translation of diagrams into coloured Petri nets (CPN) [16]. The resulting CPN are analyzed and verified using the well-known formal methods. The choice of coloured Petri nets as a formal semantic model of interaction diagrams based on the fact that the behavioral model of CPN naturally fits the behavioral model of MSC, allowing us to simulate different types of the event composition and expressions in the MSC data language. Also, CPN are well studied and there are methods and tools for analysis and verification of net models.

The paper is organized as follows. Section 2 contains a brief description of interaction diagrams. The translation method from MSCs into CPN is given in Section 3. Section 4 describes the translation of UML SD elements. The translation of MSC elements with data is given in Section 5. In Section 6, a translation

algorithm of CMSC elements is described. Section 7 contains the size estimation of the resulting CPN generated by the translation method. The case study is described in Section 8. Section 9 contains our conclusion.

2. Overview of the MSC language

In 1992, the MSC standard [1] was developed by the ITU-T in order to obtain a simple and expressive scenario-based specification language to describe interactions in distributed systems. The significant update of the standard MSC-2000 brought new diagram elements, and the concepts of data and time. As a result, the current MSC standard can be used for description of system models at a higher level of formalization.

UML 2.0 Sequence Diagrams developed by the OMG [2, 17] are strongly inspired by the MSC. Therefore, the basic ideas, visual representation, and the set of elements in the UML SD language are very similar to MSC. The main difference is that the SD diagrams are an integral part of the UML standard. This means that all objects used in SD diagrams (processes, variables, messages, etc.) are described in various UML diagrams to detail the specific aspects of the objects behavior. On the other hand, the stand-alone MSC standard has its own syntax and can be used independently of other modeling languages in the ITU-T family. Another difference of SD diagrams is that they usually represent the control flow of an object-oriented program, whereas MSCs traditionally describe the behavior of distributed systems.

Interaction diagrams depict communication between system components (*instances*, processes, objects, etc.) by means of messages. Each diagram represents a particular scenario of the system, or a set of scenarios.

All instance events are ordered along the vertical instance axis independently of other instances. The interaction between instances is performed via *messages* which determine the relationships between events of these instances. In the MSC standard all messages are asynchronous. This means that a message output and a message input are two different asynchronous events. The UML SD standard also has a synchronous type of messages. MSCs impose a partial ordering on the set of events.

Besides the message input and output events, there are other basic MSC elements including local actions, conditions, instance creation and termination events, message gates and others [18, 19]. Also, the MSC standard provides structural elements that allow us to determine different kinds of event composition for several instances. So, MSC *inline expressions* (*combined fragments* in UML SD) provide the parallel, alternative or loop composition of events. Reference expressions and *High-level MSC diagrams* (*Interaction Overview Diagrams* in UML SD) allow us to perform the synthesis and composition of several diagrams. Note that the MSC standard defines that the connections of all structural elements within diagrams are made by means of a weak sequential composition.

Consider the example of a UML SD diagram in Fig. 1. This diagram describes the scenario of interaction between the `User` and `Server` instances. All messages

except `sendData` (depicted with a message arrow of different type) are asynchronous. The operations of the user login and interaction with the server are placed in separate operands of the strict sequential composition operator `strict`, which are separated by a dotted line. This means that further interactions with the server are impossible until all events corresponding to the user login operation are executed. After logging in, the user sends the synchronous message `sendData` and executes some local action `localWork`. After receiving message from the user, the server checks a session state. This is made in the `break` operator. If the user session has expired, the `logout` message is sent to the user and then further execution of all events within `strict` operator is terminated. Otherwise, the data transmitted to the server are stored and the user is notified about it.

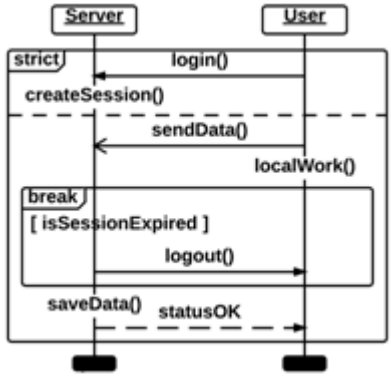

Fig. 1. An example of a UML Sequence diagram which contains the synchronous message `sendData` and two combined fragments `strict` and `break`.

3. A method for translation of MSC diagrams into Coloured Petri Nets

Let us introduce the following definitions which are used in the translation algorithms of this paper.

A *structural fragment* of MSC is a subset of MSC events, which is defined by the following rules:

- a regular MSC diagram and a reference MSC diagram is a structural fragment;
- each inline expression of MSC (a combined fragment of UML SD) is a structural fragment.

Thus, an MSC diagram can be represented as a set of structural fragments connected by means of a weak sequential composition.

We define the *start events* of a structural fragment as MSC events which can be executed first among all events of this structural fragment. By analogy with start

events, we also define the *final events* of a structural fragment. These are the events that can be executed last among all events within this structural fragment.

Then, a *set of MSC traces* is a set of event execution sequences in the diagram, where each event execution sequence begins with a start event. The end of each event execution sequence can be either a final event, or an event after execution of which the MSC will not contain dynamically legal execution traces of events.

Below we present a general method to transform the MSC diagrams into CPN. The input of the translation method is an MSC, HMSC, or MSC document given in the text notation according to the MSC standard. For UML SD and CMSC elements the additional syntax is incorporated to the existing grammar of the MSC language. The output of the algorithm is a coloured Petri net in a format compatible with the CPN Tools system. In this paper we use the CPN definition given in [16]. Note that the algorithm output is a hierarchical CPN if the original specification was defined by HMSC, or if the input MSC contains MSC reference expressions.

It can be considered that the translation method has three main stages.

At the first stage an input MSC is processed to build its internal representation called a *partial order graph*. The graph is generated as follows. For each event in the MSC, a node in the partial order graph is created. This node stores some information about the event. Nodes in the generated graph are connected with each other via directed arcs. The connection between nodes is equal to the connection between the corresponding events in the input diagram.

At the second stage, processing of the partial order graph (creating auxiliary graph nodes, unfolding MSC references, etc.) is performed.

At the third stage, the partial order graph is translated into CPN. The resulting net can be described as follows. Each node of the partial order graph corresponds to a transition of CPN. Each arc connecting two nodes of the partial order graph corresponds to a place and two oriented arcs connecting two transitions of CPN. The orientation of the generated arcs in the resulting Petri net coincides with the arcs orientation in the partial order graph. The places used to transfer control between MSC events are marked by a `UNIT` colour type. The execution of an MSC event corresponds to firing of a transition in the resulting CPN. The start events of MSC correspond to the transitions with `start` input places which have an initial marking `1`()`. The final events of MSC correspond to transitions with the `end` output places and without outgoing arcs.

The translation method described above builds a CPN which simulates all possible event traces of the input MSC. In other words, the set of all possible MSC traces will coincide with the set of all possible event sequences (firing of transitions) of the resulting CPN. An initial transition of each firing sequence in the resulting CPN is a transition that corresponds to a start event of the input diagram.

Note that in this paper we do not consider the time concept of the MSC and UML SD standards. We also do not consider the following UML SD elements: `neg`, `assert`, `ignore` and `consider`. These elements do not change the set of

diagram traces and hence do not affect the CPN generated by the translation method.

4. Translation of UML SD elements

Since the standard of UML sequence diagrams is based on the MSC standard, most elements were adopted from MSC. In [20], the comparison of UML SD and MSC elements is made.

Several UML SD elements have different names in regard to the MSC standard terminology. For example, the instances in MSC diagrams correspond to *lifelines* in UML SD diagrams; local actions correspond to *execution occurrences*; MSC references correspond to *interaction occurrences*. In the translation algorithms described below, we will use the terminology of the MSC standard.

Note that some UML SD elements which are not in the MSC standard can be modeled by the MSC elements already discussed in [18, 19]. These elements are *continuation* (can be modeled by setting and guarding conditions of the MSC), *interaction constraint* (can be modeled by predicate conditions of the MSC), *state invariant* (can be modeled by the condition MSC element described in [18]), *conditional message* (can be modeled by a regular message within an optional operator `opt`), *operation calls / replies* (can be modeled by synchronous and asynchronous messages).

Below we consider the translation algorithms for the UML SD elements which are not modeled by the MSC elements earlier discussed.

4.1 Synchronous messages

These are the messages for which the output and input events are synchronized. This means that the sender of a synchronous message has to wait for the response from the receiver. This response will indicate what the input message processing is finished by the receiver, and the sender can continue the event execution.

The translation algorithm for the synchronous message `msg` can be described as follows. First, two transitions `Out_msg` and `In_msg` are created in the output CPN. These transitions correspond to the output and input events of `msg`. The transition `Out_msg` is connected to the transition `In_msg` via a place and directed arcs similarly to the translation rules for a regular message. Next, the transition `Reply_msg` is created which means that suspension by the process that sends the message `msg` is finished. The transitions `Out_msg` and `In_msg` are connected with the transition `Reply_msg` through the place and two directed arcs as usual.

Figure 2 shows the CPN which is the result of translation of the UML sequence diagram (see Fig. 1) with the synchronous message `sendData`.

4.2 The `strict` operator

This operator represents a strict sequencing between several sets of diagram events.

We define a *synchronizing event Es* of an MSC diagram for the instances *P1*, *P2*, *..., Pn (n > 1)* as an event which can be executed only when all events from *P1*, *P2*, *..., Pn* located before *Es* have been already executed.

The translation of the `strict` operator is performed as follows.

1. All events within the `strict` operator are translated to a CPN using the common algorithm for MSCs from Section 3.
2. For every `strict` operator with *n (n > 1)* operands, *(n-1)* auxiliary transitions are created in the CPN. Each created transition simulates a synchronizing event between instances involved in the `strict` operator.
3. The synchronizing transitions *Ti (0 < i < n)* created in the previous step are placed at the joint of `strict` operands according to the following rules. All transitions corresponding to final events of the operand *i* are connected via places to the synchronizing transition *Ti*. The synchronizing transition *Ti* is in turn connected to all transitions corresponding to start events of the operand *(i+1)*. Thus, in the resulting CPN, firing of transitions corresponding to events from the operand *(i+1)* of the `strict` operator is possible only after firing of all transitions corresponding to events from the operand *i*.

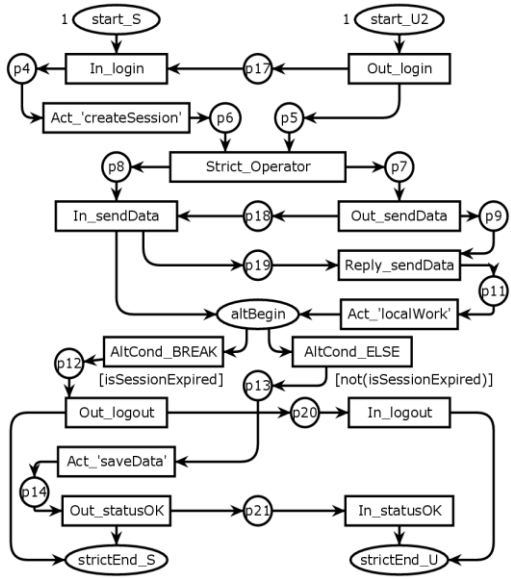

Fig. 2. CPN which is the result of translation of UML SD shown in Fig. 1.

A more detailed description of the translation of synchronizing events is given in [18]. Figure 2 shows the CPN which is the result of translation of the UML SD diagram (see Fig. 1) containing the `strict` operator.

4.3 The `break` operator

Semantics of this operator is similar to that of the break statement in many programming languages. If the `break` operator is performed in a sequence diagram, then execution of all events remaining in the enclosing (parent) structural fragment is skipped. In the UML SD standard structural fragments are called *interaction fragments*. It should be noted that the `break` operator is slightly different from the exceptional case operator `exc` of the MSC language [18]. In the MSC standard, the `exc` operator finishes execution of a current diagram.

The `break` operator belongs to combined fragments of UML SD. This fragment has one operand and should cover all instances of the parent interaction fragment. If the operand has a guard condition and the condition is true, then all events of this operand can be executed, and all remaining events of the parent fragment are ignored. If the guarding condition is false, the `break` operand is ignored and the rest of the enclosing interaction fragment is chosen.

The `break` operator can be represented as the alternative choice expression `alt` of the MSC language, where the first operand is equivalent to a single `break` operand, and the second operand is a part of the diagram that follows the parent fragment of the `break` operator.

Note that in the MSC and UML SD languages the use of the `alt` operator and its special cases (`opt`, `exc`, `break`) attached to several instances can lead to the problem of non-local choice in diagrams [1, 17, 21]. The problem is that the standards do not define which instance checks the guards, and who decides which branch should be chosen if multiple guards are true.

In our work this problem is resolved by creating the synchronizing events for each execution branch of an `alt` operator containing non-local choice. A more detailed description of the translation of an alternative expression with a non-local choice is given in [18]. The same approach is used when translating the `break` operator.

The translation algorithm of the `break` operator consists of the following steps.

1. Input and output auxiliary nodes are created for all structural fragments of a current diagram during the generation of a partial order graph.
2. Identifiers of current and parent fragments are assigned to all nodes in the partial order graph.
3. Each break fragment is translated to the output CPN according to the translation rules for `alt` operators as follows. The `alt` operator has two fixed operands. For each operand the synchronizing nodes are created to simulate a local choice. Final events of the first operand are connected to output auxiliary nodes of the parent fragment in the partial order graph (this simulates an exit from the parent fragment). Start nodes of the second operand of the `alt` operator will be output auxiliary nodes of the break fragment (this simulates the skipping of the `break` operator).

Figure 2 shows the CPN which is the result of translation of the UML SD (see Fig. 1) containing the `break` operator.

4.4 Critical Region

The `critical` operator is an atomic block of events. The block atomicity is defined by two conditions. Firstly, all events within the critical region cannot be interrupted by other events of the SD diagram which are located on the same instances as this critical region. Secondly, the atomicity of events inside the critical region cannot be broken even if it is contained within the parallel execution operator `par`.

An example of the UML SD diagram containing the critical operator is shown in Fig. 3. In this diagram, when the processes `User1` and `Server` enter the critical region by the first branch of a parallel execution, the interaction with these processes in other parallel branches will not be allowed until the execution of the critical region for these processes has been finished.

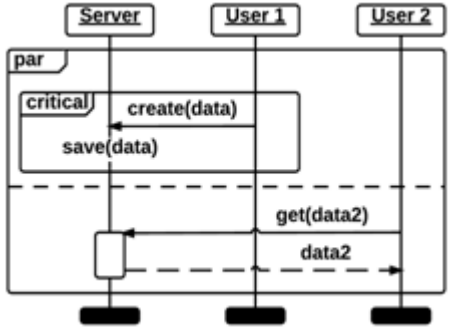

Fig. 3. An example of the UML Sequence Diagram which contains a critical region inside a `par` combined fragment.

To satisfy the first condition, it is necessary to create the synchronizing input and output events for each `critical` operator which are attached to instances involved in the critical region. The second condition is satisfied by introduction of additional places of the output CPN with flags for all events within a parent fragment `par`. Thus, an event of an instance can be executed if the flag for this instance is true. The flags for all instances involved in the critical region will be set to false when an entrance to the critical region occurs. The flags will be set to true when an exit from the critical region occurs. Note that the `critical` operator increases the size of the generated CPN in the case when this operator is placed to a par-expression with a large number of events.

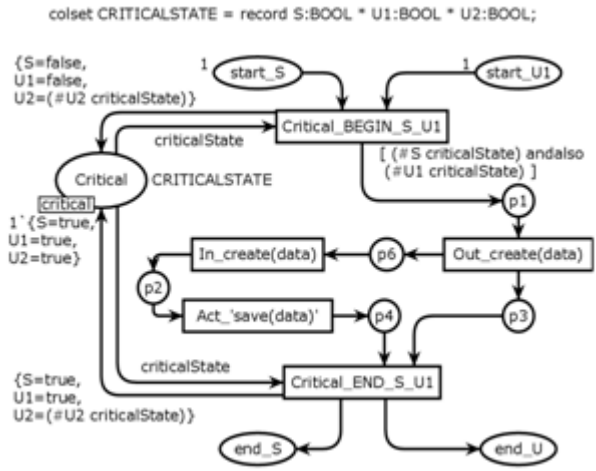

Fig. 4. The fragment of CPN which is the result of translation of critical region from the UML SD shown in Fig. 3.

The detailed translation algorithm of the critical region can be described as follows.

1. Synchronizing transitions are created at the beginning and end of each critical region.
2. If the critical region is not contained within a `par` operator, then the algorithm is finished.
3. If the critical region is contained within a `par` operator (if there are several nested `par` operators then we consider the highest level of nesting), then the next step is performed.
4. The fusion place *Critical* with a special colour type *CRITICALSTATE* is created. The place is defined as a CPN ML record «*record P1: BOOL * ... * Pn: BOOL*», where *P1, ..., Pn* are the names of diagram instances. This place will store the information about flags for each instance, signalizing about entering/finishing the critical region. The place *Critical* has an initial marking «*1` {P1=true, ..., Pn=true}*». If a flag is true for a particular instance, this means that the instance is in a normal mode of execution. Otherwise, it is assumed that the instance has entered a critical region.
5. For each transition corresponding to an event within a higher-level `par` operator with a critical region and belonging only to instances that are involved in this critical region, the next actions are made. A bidirectional arc marked by *criticalState* (the variable *criticalState* has the colour type *CRITICALSTATE*) is created. This arc connects the place *Critical* with the current transition. The transition is marked by the CPN ML guard function «*[(#P1 criticalState) andalso ... andalso (#Pk criticalState)]*», where *P1, ..., Pk* are the instance names to which the current event is attached. If the

transition already has a guard function, then the above guard expression with the prefix «*andalso*» is added at the end of this function.

6. The synchronizing transition which simulates entering the critical section for the instances *P1, ..., Pk, (k <= n)* is connected to the *Critical* place as follows. An incoming arc is marked by *criticalState*. An outgoing arc is marked by the expression *{P1=false, ..., Pk=false, ..., Pn=(#Pn criticalState)}*. This expression means that the flags of the instances involved in the critical region are reset to false, thereby preventing other events of these instances to run outside the region. This synchronizing transition is also marked by the guard function from step 5.
7. The synchronizing transition which simulates the finishing of the critical section for the instances *P1, ..., Pk, (k <= n)* is connected to the *Critical* place as follows. An incoming arc is marked by *criticalState*. An outgoing arc is marked by the expression *{P1=true, ..., Pk=true, ..., Pn=(#Pn criticalState)}*. This expression means that the flags of the instances involved in the critical region are reset to true.

Figure 4 shows the CPN fragment which is the result of translation of the `critical` operator from the UML SD diagram shown in Fig. 3.

5. Translation of diagram elements with data

An important feature of MSC and UML SD diagrams to consider them as precise and formal specifications of software systems is the data concept.

Both standards do not impose restrictions on the data notation, so any data language can be incorporated into MSCs and UML sequence diagrams. In the MSC standard data declarations are placed in the MSC document. In the UML standard data declarations are placed in the Class Diagrams and Communication Diagrams.

In this paper we only consider the case of data declarations in the MSC document [19]. We also assume that the MSC data language allows simple types – `Boolean`, `Integer` and `String` – and the composite type `Enumeration`. An expression in the data language consists of variables, literals, parentheses, arithmetic and assignment operators, and comparisons.

The MSC document in addition to data type and variable declarations also describes the signatures of all messages with data used in the diagrams. The *message signature N(T1, T2, ..., Tn)* is a set of a message name *N* and the ordered set of parameter types *Ti* which defines the data tuples transmitted by this message. For example, the message signature *frame(Integer, Boolean)* means that a diagram contains a message with the name *frame*. This message transmits a data tuple with a content of `Integer` and `Boolean` types.

The data in diagrams are used in messages, local actions and conditions. Data expressions in messages and local actions can contain only variable assignment operations. A data expression in conditions cannot contain an assignment operator

and can be a statement with a `Boolean` return value. An example of an MSC diagram containing messages with data is shown in Fig. 5.

The translation algorithm of events with data consists of two stages.

At the first stage, the colour type and variable declarations in the CPN ML language are generated from the input MSC document. These declarations will be used in the CPN obtained by translation of MSC with data events.

Generation of the colour types and variables for MSC elements with data is as follows:

1. Data types declared in the `data` block of the MSC document are converted into the corresponding colour types of CPN ML.
2. Local variables declared for each instance in the `inst` block of the MSC document are converted to variables of CPN ML with the same name and with the colour type resulting from the transformation at step 1.
3. Message signatures declared in the `msg` block of the MSC document are used to simulate message buffers in the resulting CPN. The signature *N(T1, T2, ..., Tn)* is translated to a `product` colour type of the CPN ML language: *colset pT1T2...Tn = product T1 * T2 * ... * Tn*. To simulate the buffer which contains messages with the same signature *N(T1, T2, ..., Tk)*, the `list` colour type is used: *pT1T2...TkList = list pT1T2...Tk*.
4. For colour types generated at step 3, auxiliary variables *pT1T2...Tn_var* and *pT1T2...TnList_var* of types *pT1T2...Tn* and *pT1T2...TnList* are created.

At the second stage, the translation of an MSC diagram which uses data declared in the MSC document is performed.

The translation of local actions and conditions with data is described in [19]. Below we describe the translation of messages with data. The MSC and UML SD standards imply that communicating instances send messages through the buffer which is local regarding to messages. This means that there is one FIFO buffer for every message in a diagram. Buffers which contain MSC messages with data are modeled by places of the `list` colour type in the resulting CPN. The list is a queue of records (CPN `product` types), where each record contains the set of transmitted data values. Thus, the translation algorithm for messages with data is as follows:

1. For each message *msg_i(T1, T2, ..., Tn)* in the diagram, a place in the resulting CPN is created to simulate the message buffer as follows. The name *msg_i* and the colour type *pT1T2...TnList* are assigned to the place. The initial marking for this place is set up to the value 1`[], which indicates that the buffer is empty.
2. The input and output events of the message *msg_i* are translated into the corresponding transitions of the CPN.
3. Each transition corresponding to the input/output events of the message *msg_i* is connected to fusion places modeling the variable states. The details of variable state simulation in the resulting CPN are given in [19].

4. For a transition corresponding to an output event of the message *msg_i*, an input arc from the place *msg_i* is created with the inscription *pT1T2...TnList_var*. Also the output arc is created with the inscription *pT1T2...TnList_var ^^ [(VarT1, VarT2, ..., VarTn)]*, where *VarTi* are the variable names with data transmitted from the sender instance. This expression describes the addition of a tuple with a message content into the buffer.
5. For a transition corresponding to an input event of the message *msg_i*, an input arc from the place *msg_i* is created with the inscription «*pT1T2...Tn_var :: pT1T2...TnList_var*». This expression means that a head element and a tail part of the buffer are got and saved to the specified variables. Also, the output arc is created for this transition with the inscription *pT1T2...TnList_var*, which is used to simulate the removal of the upper buffer element.
6. The process of obtaining and saving the transmitted data by the receiver instance is modeled in the resulting CPN as follows. The fusion places are created for each variable listed in the actual parameters of the message signature *msg_i*. These places are used to store the transmitted data of the message *msg_i* into the local variables of the receiver process (see the translation of local actions with the data for full details [19]). The transition corresponding to the input event of the message *msg_i* is connected to the created fusion places. The outgoing arcs from each fusion place are marked by the corresponding variable names. The arcs coming into the fusion places are marked by the inscription «*Tj_var = #j pT1T2...Tn_var*», where *Tj_var* is the *j*-th variable name of the receiver in the signature *msg_i*, and the expression «*#j pT1T2 ... Tn_var*» means that the *j*-th element from the tuple variable *pT1T2 ... Tn_var* is got.

Figure 6 shows the CPN which is the result of translation of the MSC from Fig. 5 containing non-regular messages with data introduced in the next section.

6. Translation of compositional MSC elements

The non-standard extension of MSC diagrams called *Compositional Message Sequence Charts (CMSCs)* [3, 4] has been developed to increase the expressive power of the MSC language and to describe scenarios with complex parallel communication of processes.

In [3, 4], the authors show that the expressiveness of MSC diagrams is not sufficient for the specification of a certain type of interactions, such as sliding window protocols. In the CMSC language it is possible to describe this kind of protocols using partial-defined messages. The use of this type of messages, on the one hand, allows messages to be decomposed into several diagrams. On the other hand, such messages use a different buffer type which is similar to the buffer model in the communicating finite-state machines or SDL language.

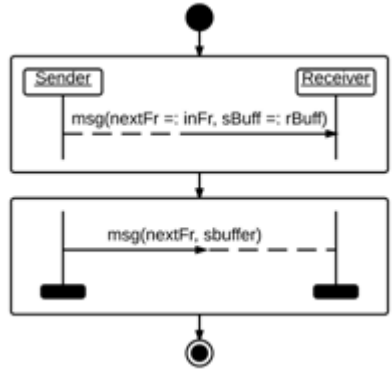

Fig. 5. The HMSC diagram with two MSCs which contain the unmatched message `msg`.

The CMSC language is defined as the MSC language, except for the definition of messages. In Compositional MSC diagrams, the input and output message events are partially defined. This means that for the partial-defined message there are multiple input events for a single output event and vice versa. Such messages in a CMSC are called *unmatched messages*.

Unmatched send message events and unmatched receive message events use a new buffer model. This buffer is local relative to the two instances involved in the message exchange (this is a so-called pair buffer).

An example of the CMSC diagram is shown in Fig. 5. Unmatched messages are shown as arrows with a dotted part. The CMSC shows the decomposition of the unmatched message `msg` which is contained in two different reference MSC diagrams.

Below we describe the translation algorithm for unmatched messages.

1. Each input and output event of the unmatched message *umsg_i(T1, T2, ..., Tn)* is converted to the corresponding transition of the CPN.
2. If the message does not contain any data then the following steps are made.
    2.1 The fusion place simulating a buffer is created with the `UNIT` colour type and the name «*CMSC P1-to-P2*», where *P1* is the name of the instance that sends the message *umsg_i* and *P2* is the name of the instance that receives this message. Note that the name of the created place is unique for the couple of instances *P1* and *P2* which communicate in the direction from the first to the second instance.
    2.2 For each transition corresponding to the output unmatched message event from *P1* to *P2*, an output arc is created. This arc is connected to the place «*CMSC P1-to-P2*».

2.3 For each transition corresponding to the input unmatched message event from *P1* to *P2*, an input arc is created. This arc connects the place «*CMSC P1-to-P2*» with the current transition.

3. If the message contains data then the following steps are made.

3.1 The fusion place simulating a buffer is created as follows. The place type is set to *pT1T2...TnList*. The place name is set to «*CMSC P1-to-P2-umsg_i*», where *P1* is the name of the instance that sent the message with data, *P2* is the name of the instance that receives this message, and *umsg_i* is the message name. The place is marked by 1`[]. Note that the name of the created place is unique for the couple of instances *P1* and *P2* with a given type of the message signature. Thus, the unmatched messages with the same signature will be sent by *P1* through a common buffer. The same is true for the receiving of unmatched messages.

3.2 The processing of transitions corresponding to the output events of unmatched messages with data is carried out by the translation rules of step 4 of the previous section.

3.3 The processing of transitions corresponding to the input events of unmatched messages with data is carried out by the translation rules of steps 5 and 6 of the previous section.

Figure 6 shows the CPN which is the result of translation of the CMSC (see Fig. 5) with the unmatched message `msg`.

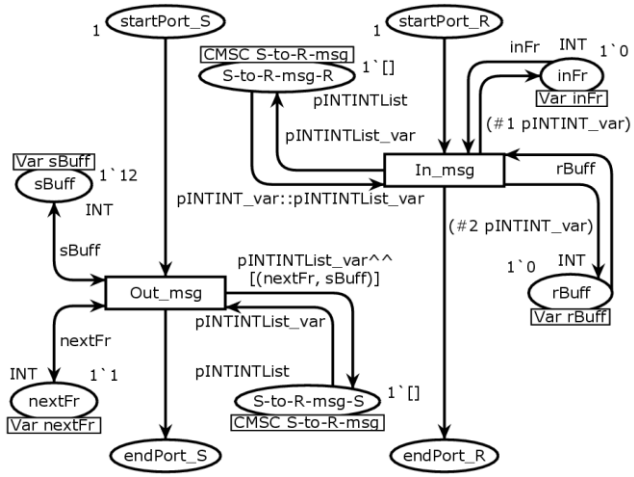

Fig. 6. The CPN which is the result of translation of the HMSC shown in Fig. 5.

7. Size estimate of the resulting CPN

Below we consider the estimate of the number of transitions, places, and arcs in the CPN, given as the result of translation algorithms described in our paper.

Let us consider the MSC diagram with N events, M messages and the number P of instances containing events.

Introduce the following notation: S is the number of start and final MSC events; AC is the number of local actions and conditions; IP is the number of parallel operators `par`; IL is the number of `loop` operators; N_{IP} is the maximum number of events among `par` operators of the diagram; BR is the number of `break` operators; ST is the number of `strict` operators; OP_{ST} is the maximum number of operands among `strict` operators of the diagram; CR is the number of `critical` operators within `par` operators; VAR is the number of variables defined in the MSC.

Then the upper bound T of the number of transitions in the resulting CPN will be:

$$T \le N + 2P \cdot (IP + IL) + ST \cdot (OP_{ST} - 1) + P \cdot BR + 2CR.$$

The upper bound P of the number of places in the resulting CPN has the following form:

$$P \le N + M + S + VAR + 2P \cdot (IP + IL) + ST \cdot (OP_{ST} - 1) \cdot P + P \cdot BR + 2CR.$$

The upper bound A of the number of arc in the resulting CPN has the following form:

$$A \le 2N + 4M + 2 \cdot VAR \cdot (AC + 2M) + 4P \cdot (IP + IL) + 2ST \cdot (OP_{ST} - 1) \cdot P + 2P \cdot BR + 2CR \cdot N_{IP}.$$

As we can see, a significant contribution to the size estimate of the resulting CPN is made by the operators `par`, `loop`, `break` and `critical`.

8. Case study: Alternating Bit Protocol

Let us consider an example of the property verification for the MSC specification of a protocol known as the *Alternating Bit Protocol* (ABP) [22].

This protocol is bidirectional. This means that the data between the two communicating machines are transmitted in both directions. The protocol operates as follows. The sender sends a sequence of data frames to the receiver. Each data frame consists of two parts: a one-bit frame number and a portion of data. When a data frame arrives to the receiver, it sends to the sender an acknowledgment frame that contains the number of the received frame. Both processes use a timer to wait for the next frame. Thus, the sender is sending a current data frame continuously until it receives an acknowledgment from the receiver with the current frame number. On the other hand, after getting a data frame, the receiver is sending an acknowledgment frame to the sender continuously until it receives a new data frame from the sender.

The MSC specification of the ABP protocol is presented in [23]. In the specification, the `par` operator and CMSC elements are used to describe the distributed interaction between two machines. The timer execution events of communicating processes are modeled in the resulting CPN by firing of transitions

corresponding to these timer events. The transmitted data in the protocol are a sequence of integers from 1 to 4.

To reduce the state space of the resulting CPN and apply the CPN verifier based on SPIN [24], the initial MSC specification should be rewritten into a quasi-regular form in which diagrams do not contain unlimited loops [19]. To do this, we introduced additional restrictions on the protocol model without loss of generality: the frame number that can be lost during transmission is limited by a constant.

For analysis and verification of the ABP model, the following properties of a proper behavior are formulated:

1. The sequence of the received data is equal to the sequence of the sent data.
2. The receiver does not accept the same message twice.
3. The sender does not send a new message before a previous one was acknowledged.
4. The sequence of the received frames is a prefix of the sequence of the sent frames.

The property 1 is a postcondition. For the protocol model, it means that if the event execution of the MSC specification ends at its endpoint, then this property is satisfied. For the CPN model of the protocol, it means that the resulting net should not have dead markings except the markings corresponding to the endpoint of the MSC specification. Properties 2, 3 and 4 are specified by linear temporal logic (LTL) formulas [23].

The analysis of the model properties was made in the CPN Tools (property 1) and in the automated verification system developed in IIS SB RAS on the basis of SPIN (properties 2, 3 and 4). Verification of the properties described above showed that they are satisfied for the ABP protocol model.

The property validation was also made for the ABP protocol model containing errors. In the first case, we considered a protocol model in which one of the processes can send a new message non-deterministically, without waiting for reception of the previous one. In the second case, we considered a protocol model in which the sender can send non-deterministically a frame with incorrect data. During verification of these ABP models, the following property violations were detected. In the first case, property 3 was violated (and property 4, consequently). In the second case, property 4 was violated.

For the violated properties, the counterexamples were generated which contain traces in the MSC specification leading to a broken state. The file with a counterexample is a sequence of CPN transitions and net markings.

Using the counterexamples, the errors were localized in the original MSC specification. Since each transition corresponds to a concrete event in an MSC, and the MSC variables state is calculated by the values of places with the same name as original variables, the localization of errors in a diagram by a counterexample is straightforward.

9. Conclusion

The scenario-based specification languages are a convenient and expressive way to describe a system behavior during the design and development stages. The most popular in practice among them are the MSC and UML SD languages. Despite a wide application of these notations, the methods of analysis and verification are still underdeveloped.

In this paper we describe the method for translation of MSC diagrams into coloured Petri nets. To the best of our knowledge, our method is the first to cover a large set of the MSC and UML SD diagram elements with minimal restrictions on the considered elements. Unlike the related papers, the translation method fully supports the diagram elements with dynamic data and elements of compositional MSC diagrams. The consideration of all elements listed above, on the one hand, allows us to apply the translation method for most interaction diagrams used in practice. On the other hand, this allows us to use the method for verification of distributed systems with complex object interactions.

A CPN given as a result of the translation method can be analyzed and verified by the known verification methods and program tools. In particular, one can analyze some properties of MSC diagrams using the CPNTools, and verify properties specified by LTL formulas using the method [24].

The software tool was implemented on the basis of the translation algorithms. The translator has been tested on various examples of communication protocols. In particular, the alternating bit protocol specified by MSCs has been considered. For the protocol, the CPN model was generated. Some properties of the resulting CPN was analyzed by the CPN Tools and verified by the CPN verifier [24].

In our further work we plan to develop the approach for formal justification of correctness of the translation algorithms. We will study other MSC extensions intended for specification of distributed systems. Also, we plan to use the translator for verification of other examples of distributed systems and communication protocols.

References

[1]. ITU-T Recommendation Z.120 (02/2011): Message Sequence Charts (MSC), 2011.
[2]. Unified Modeling Language (UML) 2.5. Object Management Group, 2013. (http://www.omg.org/spec/UML/2.5/Beta2/)
[3]. Genest B. Compositional Message Sequence Charts (CMSCs) Are Better to Implement Than MSCs. TACAS 2005, LNCS 3440, 2005. P. 429-444.
[4]. Genest B., Muscholl A., Peled D. Message Sequence Charts. Lectures on Concurrency and Petri Nets, LNCS 3098, 2003. P. 537-558.
[5]. Rajeev Alur, Holzmann G.J., Peled D. An Analyzer for Message Sequence Charts. TACAS 96, LNCS 1055, 1996. P. 35-48.
[6]. UBET (MSC/POGA) toolset — http://cm.bell-labs.com/cm/cs/what/ubet/index.html
[7]. Cinderella MSC computer tool — http://www.cinderella.dk/msc.htm
[8]. IBM Rational Tau system — www.ibm.com/software/products/en/ratitau

[9]. Gaudin E., Brunel E. Property Verification with MSC. SDL 2013, LNCS 7916, 2013. P. 19-35.
[10]. Fernandes J.M., Tjell S., Jorgensen J.B., Ribeiro O. Designing Tool Support for Translating Use Cases and UML 2.0 Sequence Diagrams into a Coloured Petri Net. SCESM '07: Proc. of the Sixth International Workshop on Scenarios and State Machines, Washington, DC, USA, 2007. P. 2.
[11]. Yang N., Yu H., Sun H., Qian Z. Modeling UML sequence diagrams using extended Petri nets, Telecommunication Systems, Springer, 2012. V. 51, N. 2-3, P. 147-158.
[12]. Eichner C., Fleischhack H., Meyer R., Schrimpf U., Stehno S. Compositional Semantics for UML 2.0 Sequence Diagrams Using Petri Nets. SDL-Forum 2005, LNCS 3530, 2005. P. 133-148.
[13]. Lima V., Talhi C., Mouheb D., Debbabi M., Wang L., Pourzandi M. Formal verification and validation of UML 2.0 Sequence Diagrams using source and destination of messages. Electron. Notes Theor. Comput. Sci., 2009. V. 254, P. 143–160.
[14]. Shen H., Robinson M., Niu J. Model Checking Combined Fragments of Sequence Diagrams. Software and Data Technologies, Springer, 2013. V. 411, P. 96-111.
[15]. Holzmann G. The Spin model checker: primer and reference manual. Addison Wesley, 2003. 608 p.
[16]. Jensen K., Kristensen L.M. Coloured Petri Nets: Modeling and Validation of Concurrent Systems, Springer, 2009. 384 p.
[17]. Micskei Z., Waeselynck H. The many meanings of UML 2 Sequence Diagrams: a survey. Software and Systems Modeling, Springer, 2011. V. 10, N. 4, P. 489-514.
[18]. Chernenok S.A., Nepomniaschy V.A. Analysis of Message Sequence Charts of Distributed Systems Using Coloured Petri Nets, Preprint 171, Institute of Informatics Systems, Novosibirsk, 2013 (in Russian). http://www.iis.nsk.su/files/preprints/171.pdf
[19]. Chernenok S.A., Nepomniaschy V.A. Analysis and Verification of Message Sequence Charts of Distributed Systems with the Help of Coloured Petri Nets. Modeling and Analysis of Information Systems, 2014. V. 21, N. 6, P. 94-106 (in Russian).
[20]. Haugen O. Comparing UML 2.0 Interactions and MSC-2000. 4th International SDL and MSC Workshop, LNCS 3319, 2005. P. 65-79.
[21]. Abdallah R., Gotlieb A., Helouet L., Jard C. Scenario Realizability with Constraint Optimization. FASE 2013, LNCS 7793, 2013. P. 194-209.
[22]. Tel G. Introduction to distributed algorithms. Cambridge University Press New York, USA, 2000. 612 p.
[23]. Chernenok S. A. Examples of Analysis and Verification of Message Sequence Charts. Appendix, 2015. (http://bitbucket.org/chernenok/msc-verification)
[24]. Stenenko A.A., Nepomniaschy V.A. Model Checking Approach to Verification of Coloured Petri Nets, Preprint 178, Institute of Informatics Systems SB RAS, Novosibirsk, 2015 (in Russian). http://www.iis.nsk.su/files/preprints/178.pdf

Применение раскрашенных сетей Петри для верификации распределенных систем, специфицированных MSC-диаграммами

Сергей Черненок <chernenoksergey@gmail.com>,
Валерий Непомнящий <vnep@iis.nsk.su>,
Институт систем информатики им. А.П. Ершова СО РАН,
630090, Россия, г. Новосибирск, ул. Лаврентьева, д. 6.

Аннотация. Язык диаграмм последовательностей сообщений (MSC-диаграмм) является сценарно-ориентированным языком спецификаций, который широко используется на этапе проектирования для описания взаимодействия компонент в распределенных системах. Однако, существующие методы и средства проверки корректности MSC-диаграмм недостаточно развиты. К их основным недостаткам относятся небольшой набор поддерживаемых конструкций MSC-диаграмм, ограничения на поведение элементов диаграмм и на набор анализируемых свойств. Данная статья описывает метод трансляции MSC-диаграмм в раскрашенные сети Петри (CPN), который используется для анализа и верификации свойств MSC-диаграмм. Метод трансляции состоит из трех основных этапов: построение внутреннего представления MSC-диаграммы в виде графа частичного порядка, обработка узлов графа и преобразование графа в CPN. Результатом трансляции является иерархическая раскрашенная сеть Петри в формате, совместимом с известной системой моделирования и анализа CPN Tools. Кроме элементов из основного стандарта MSC рассматриваются следующие конструкции MSC-диаграмм: элементы языка данных MSC (сообщения, локальные действия и условия с данными), элементы диаграмм взаимодействий стандарта UML (синхронные сообщения, комбинированные фрагменты) и конструкции композиционных MSC-диаграмм (частично-определенные сообщения). На основе этого метода трансляции реализован транслятор из MSC-диаграмм в CPN. Свойства результирующих CPN анализируются и верифицируются при помощи системы CPN Tools и верификатора CPN на основе системы SPIN. Если в результате верификации проверяемое свойство оказывается ложным и найден контрпример, то место ошибки может быть локализовано в исходной MSC-спецификации. Для этого на основе контрпримера генерируется трасса в MSC до места ошибки, представляющая собой последовательность событий диаграммы и состояний переменных каждого процесса. Применение метода трансляции и средств анализа и верификации продемонстрировано на примере сетевого протокола ABP (Alternating Bit Protocol).

Keywords: specification; translation; verification; distributed systems; communication protocols; message sequence charts; UML sequence diagrams; coloured Petri nets

DOI: 10.15514/ISPRAS-2015-27(3)-14

Для цитирования: Черненок Сергей. Применение раскрашенных сетей Петри для верификации распределенных систем, специфицированных MSC-диаграммами. Труды ИСП РАН, том 27, вып. 3, 2015 г., стр. 197-218 (на английском языке). DOI: 10.15514/ISPRAS-2015-27(3)-14.

Список литературы

[1]. ITU-T Recommendation Z.120 (02/2011): Message Sequence Charts (MSC), 2011.
[2]. Unified Modeling Language (UML) 2.5. Object Management Group, 2013. (http://www.omg.org/spec/UML/2.5/Beta2/)

[3]. Genest B. Compositional Message Sequence Charts (CMSCs) Are Better to Implement Than MSCs. TACAS 2005, LNCS 3440, 2005. P. 429-444.
[4]. Genest B., Muscholl A., Peled D. Message Sequence Charts. Lectures on Concurrency and Petri Nets, LNCS 3098, 2003. P. 537-558.
[5]. Rajeev Alur, Holzmann G.J., Peled D. An Analyzer for Message Sequence Charts. TACAS 96, LNCS 1055, 1996. P. 35-48.
[6]. UBET (MSC/POGA) toolset — http://cm.bell-labs.com/cm/cs/what/ubet/index.html
[7]. Cinderella MSC computer tool — http://www.cinderella.dk/msc.htm
[8]. IBM Rational Tau system — www.ibm.com/software/products/en/ratitau
[9]. Gaudin E., Brunel E. Property Verification with MSC. SDL 2013, LNCS 7916, 2013. P. 19-35.
[10]. Fernandes J.M., Tjell S., Jorgensen J.B., Ribeiro O. Designing Tool Support for Translating Use Cases and UML 2.0 Sequence Diagrams into a Coloured Petri Net. SCESM '07: Proc. of the Sixth International Workshop on Scenarios and State Machines, Washington, DC, USA, 2007. P. 2.
[11]. Yang N., Yu H., Sun H., Qian Z. Modeling UML sequence diagrams using extended Petri nets, Telecommunication Systems, Springer, 2012. V. 51, N. 2-3, P. 147-158.
[12]. Eichner C., Fleischhack H., Meyer R., Schrimpf U., Stehno S. Compositional Semantics for UML 2.0 Sequence Diagrams Using Petri Nets. SDL-Forum 2005, LNCS 3530, 2005. P. 133-148.
[13]. Lima V., Talhi C., Mouheb D., Debbabi M., Wang L., Pourzandi M. Formal verification and validation of UML 2.0 Sequence Diagrams using source and destination of messages. Electron. Notes Theor. Comput. Sci., 2009. V. 254, P. 143–160.
[14]. Shen H., Robinson M., Niu J. Model Checking Combined Fragments of Sequence Diagrams. Software and Data Technologies, Springer, 2013. V. 411, P. 96-111.
[15]. Holzmann G. The Spin model checker: primer and reference manual. Addison Wesley, 2003. 608 p.
[16]. Jensen K., Kristensen L.M. Coloured Petri Nets: Modeling and Validation of Concurrent Systems, Springer, 2009. 384 p.
[17]. Micskei Z., Waeselynck H. The many meanings of UML 2 Sequence Diagrams: a survey. Software and Systems Modeling, Springer, 2011. V. 10, N. 4, P. 489-514.
[18]. С.А. Черненок, В.А. Непомнящий. Анализ MSC-диаграмм распределенных систем с помощью раскрашенных сетей Петри // Препринт 171, ИСИ СО РАН, Новосибирск, 2013. http://www.iis.nsk.su/files/preprints/171.pdf
[19]. С.А. Черненок, В.А. Непомнящий. Анализ и верификация MSC-диаграмм распределённых систем с помощью раскрашенных сетей Петри // Моделирование и анализ информационных систем, 2014 г., Т. 21, N 6, с. 94-106.
[20]. Haugen O. Comparing UML 2.0 Interactions and MSC-2000. 4th International SDL and MSC Workshop, LNCS 3319, 2005. P. 65-79.
[21]. Abdallah R., Gotlieb A., Helouet L., Jard C. Scenario Realizability with Constraint Optimization. FASE 2013, LNCS 7793, 2013. P. 194-209.
[22]. Tel G. Introduction to distributed algorithms. Cambridge University Press New York, USA, 2000. 612 p.
[23]. Chernenok S. A. Examples of Analysis and Verification of Message Sequence Charts. Appendix, 2015. (http://bitbucket.org/chernenok/msc-verification)
[24]. А.А. Стененко, В.А. Непомнящий. Верификация раскрашенных сетей Петри методом проверки моделей // Препринт 178, ИСИ СО РАН, Новосибирск, 2015. http://www.iis.nsk.su/files/preprints/178.pdf