ISSN 0361-7688, Programming and Computer Software, 2007, Vol. 33, No. 1, pp. 24–33. © Pleiades Publishing, Ltd., 2007.
Original Russian Text © A.N. Terekhov, V.V. Sokolov, 2007, published in Programmirovanie, 2007, Vol. 33, No. 1.

Implementation of the Conformation of MSC and SDL Diagrams in the REAL Technology

A. N. Terekhov and V. V. Sokolov

St. Petersburg State University,
Universitetskii pr. 28, Petrodvorets, St. Petersburg, 198504 Russia
e-mail: ant@tercom.ru, svv@tercom.ru

Received October 13, 2004

Abstract—Various methods of relating MSC and SDL diagrams are considered, including methods that allow one to automate manual conversion from scenarios of the behavior of the entire system (HMSC/MSC) to behavior models of separate objects (SDL), methods for supporting their conformity during the entire life cycle (concerted modification), and extensions of the MSC diagrams for real situations. The existing algorithms are reviewed, their advantages and disadvantages are considered, and our own approaches eliminating the disadvantages are proposed. The proposed algorithms are integrated into one complex.

DOI: 10.1134/S0361768807010045

1. INTRODUCTION

Our team has been engaged in the development of software for telephone exchanges and other real-time systems since the beginning of the 1980s. It turned out that the most difficult aspect is interaction between customers, algorithm creators, programmers, protocol experts, electronics engineers, etc. Moreover, problems arise because of communication gap rather than inadequate qualification of the development partners. To overcome these difficulties, the International Consultative Committee on Telegraphy and Telephony (CCITT, nowadays ITU-T) has designed a series of graphical description methods, in particular, the Specification and Description Language (SDL) [1] and Message Sequence Chart (MSC) [2].

An SDL diagram is very similar to traditional flowcharts, but has several important extensions: the symbol of a state in which the process does not load the processor and is waiting for reception of one or several signals, the signal input symbol, and signal sending symbol (Fig. 1). In the SDL diagrams, a special attention is paid to the signal use logic, whereas the logic of execution of actions not related to the signals is detailed only when required. Therefore, an action can be either atomic, as shown in Fig. 1, or rather complex, consisting of tens or hundreds of operators.

MSC diagrams[1] allow one to describe scenarios of the time behavior of the system. Time runs from top to bottom; vertical lines represent system objects; and arrows designating signals[2] are drawn between them. A primitive MSC diagram is shown in Fig. 2. The MSC diagrams consist of separate scenarios and structures specifying the sequences of execution of these scenarios. They are widely used by various international organizations to describe protocols, for example, standards developed by the ITU-T organization; however, exhaustive descriptions of system behavior, including emergency processing, maintenance, tariffication, etc., are rarely presented. Both MSC and SDL diagrams are applied to describe the dynamic behavior of a system;[3] their comparative analysis is given in the table.

In modern systems, both standards are used, because they provide different information on the dynamic behavior of an object and supplement each other. Similarly, they both are required for technology. Correspondingly, there is a need in their consistency.

In the paper, we briefly substantiate the use of both MSC and SDL diagrams in one tool, describe the existing approaches to their simultaneous use, and point out the problematic aspects. Our technological tool REAL [4] also uses both MSC and SDL diagrams and we describe our approach to the resolution of the problems.

2. DISCUSSION OF THE PROBLEM

We have already mentioned that MSC and SDL give different views of the system. We emphasize that they are not just different representations of the same model,

[1] At present, HMSC diagrams [3], which are extensions of MSC diagrams, are widely used. Inherently, they are similar to the MSC diagrams in many respects; for this reason, we will not consider them separately and will include them into the term MSC.

[2] In the paper, the concepts of the SDL signal and the MSC message are considered to be equivalent.

[3] These models also include tools for describing the static part of the system, but there are no difficulties here.

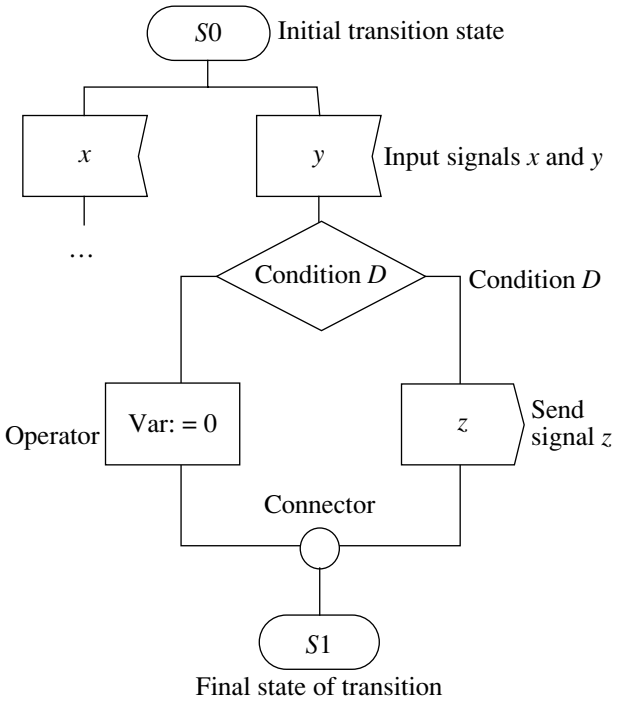

Fig. 1. Example of SDL diagram.

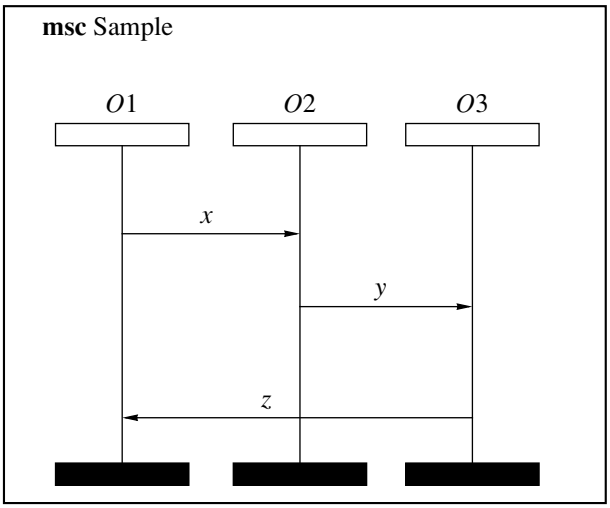

Fig. 2. Example of MSC diagram.

but contain different information. For example, only from the SDL diagrams, we can ascertain that one or another behavior has been selected. The corresponding information is stored in variables that are absent in the MSC diagrams.

To specify the information that is contained only in the MSC diagrams, let us consider a hypothetical situation with a bank, when a teller gives money to some clients immediately and first consults with the bank director before giving money to other clients. Having the SDL models of all participants, it is generally impossible to determine whether the director participates in the procedure of payment to a certain client. It is possible to show that this problem is similar to the problem of self-applicability and is algorithmically unsolvable. The corresponding information is stored in the MSC diagrams, which contain information not only about the way the objects exchange messages, but also about who participates in the communication scenarios.

Therefore, to obtain some kind of information, it is necessary to refer to the MSC diagrams, and to obtain another kind of information, to the SDL diagrams. That is why both diagram types are used in modern technological tools. The process of creating a system in our technological tool REAL at the time when paper [4] was written is shown in Fig. 3. It is seen that there is a gap between the MSC and SDL models, which retards the development, because the system dynamics is doubly created first by MSC and then by SDL, and can result in errors when the models are inconsistent. In our opinion, a gap between the MSC-like model and SDL also occurs in other approaches, such as UML Sequence and Collaboration diagrams [5, 6], as well as in approaches based on MSC without UML, or in approaches based on UseCaseMaps [7].

For a technologist, it would be ideal if there were some universal model, a kind of a three-dimensional cube one face of which shows the SDL diagrams and other face presents the MSC diagrams. As far as we know, such a model does not exist, and we can only refer to several approaches to co-ordination of the MSC and SDL diagrams in the existing CASE systems. They can conditionally be divided into the following groups.

1. MSC and SDL are independent.

(a) MSC and SDL represent separate independent types of diagrams. In this case, their inconsistency results in potential errors.

Comparative characteristics of SDL and MSC

MSC	SDL
Describes interaction of different parts of the system; is external for the object	Describes the interior of each object; is internal for the object
The logic of interaction of several objects is represented. The behavior logic of each object is hidden	The behavior logic of each object is represented. The logic of their interaction is hidden
Information about the choice of a behavior is verbal	The logic of the functioning is detailed up to the variable values
Are created at the design stage	Are created at the programming stage
Are used when discussing the problem with experts in the knowledge domain	Are used by the programmers mainly

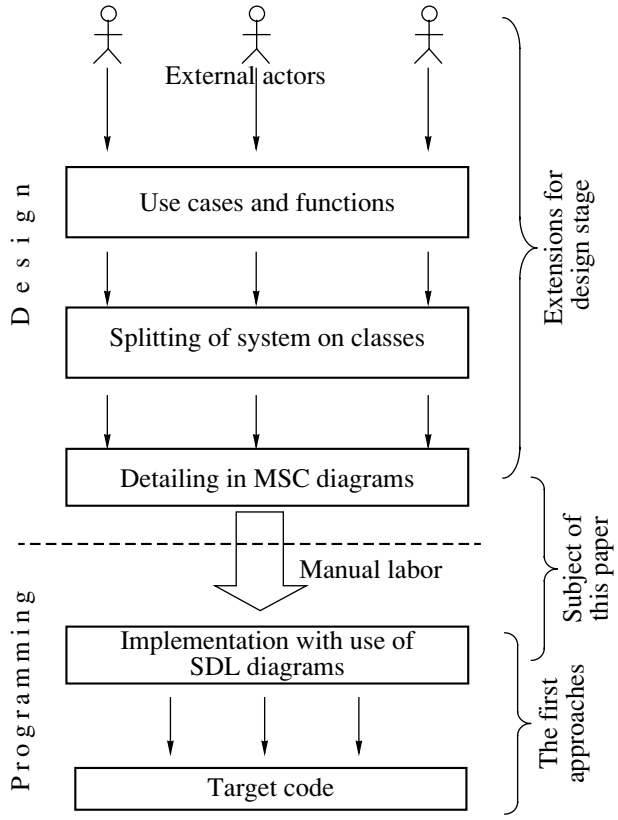

Fig. 3. Place of SDL and MSC diagrams in technology REAL at the moment of writing paper [4].

(b) Conversion of MSC diagrams to SDL diagrams is being performed by a person; this introduces errors and often leads to non-optimal decisions and unnecessary and redundant code, especially in the case of long-term support of the project.

2. Attempts to create systems on the basis of the MSC diagrams with subsequent SDL synthesis.

(a) An attempt to support the entire development cycle entails extension of the MSC standard and transfer of the code, data, and other technical details into the MSC diagrams [8]. After that, there arise problems associated with an increase of the number of scenarios and subsequent inconsistency of code fragments. The MSC structure is hardly applicable for representing a code in it. MSC diagrams answer the question "What can happen?" rather than the question "Why does it happen this way?" As a result, they are redundant and poorly adapted to the use of "arithmetic" data in them.

(b) One-way synthesis. Here, SDL diagrams are once synthesized from MSC diagrams, and, then, the derived structures are used [8–10]. In this case, manual modification is very expensive. It often results in total disregard of the MSC diagrams used in the synthesis.

(c) Incremental algorithms. The MSC standard is reduced to the diagrams describing only traces.[4] There are algorithms [11] that allow one to add a new MSC specification (trace) to SDL diagrams. The described approach gives results only on a specific class of problems because of cyclicity in real problems. For the original MSC standard, the problem of development of incremental algorithms was also considered, but it was solved only for a very restricted set of changes [12].

The algorithms used in items 2.c and 2.b [9, 10] are capable of synthesis not from all MSC descriptions. The algorithm of item 2.a can do it always. A side effect of this algorithm is that, in the process of its functioning, the data can be transformed in such a way that the initial MSC diagram will be hardly recognized in the derived SDL diagram. Let us note here that the impossibility to carry out synthesis in many reasonable situations is a very weak point; that is why the authors prefer variants of the algorithm 2.a.

3. Algorithms of comparison of the MSC and SDL diagrams for conformity checking.

(a) Traces are created on the basis of MSC diagrams, and the derived traces are "superposed" onto the SDL diagrams[5] [13–15]. If the trace fits the diagram, then everything is fine. If it does not, then the situation is considered to be erroneous. This method is not sufficiently effective, because addition of a debugging signal somewhere breaks the signal sequence in the trace, and the SDL model is assumed not to conform to the MSC specifications [16]. This method cannot be used to check up a case when an SDL model is an implementation of several roles from MSC diagrams simultaneously.

(b) The authors are aware of the idea of modification of the previous algorithm with permission to skip signals. It was described in [17], but is not widely used because of low reliability of the results.

(c) Usage of one of the algorithms [18] concerned with analysis of the internal states of a protocol, with two models—MSC and SDL—being executed in parallel. In the course of execution, it is checked whether the sent and received signals are identical. The algorithms were initially developed for checking internal consistency of an algorithm and cease to give results when checking conformity of the SDL diagrams to MSC under presence of disagreements between MSC and SDL, such as absence of cyclicity in the MSC documentation and its presence in the SDL model.

In our opinion, both MSC and SDL diagrams should be included into a CASE system to support the possibility of different views on the system dynamics. Practice shows that, in industrial systems, SDL must represent

[4] In this paper, a trace is meant to be a sequence of messages.

[5] In the sense of each-symbol coincidence of the trace signals with the diagram signals.

not only the documentation, but also the program from which the executable code can be generated automatically. To develop an integrated system for the MSC and SDL diagrams, it is necessary to use procedures of transition both from MSC to SDL (generation) and back (verification tools). Here, there are several issues to be elaborated:

• improve the verification procedure so that it allows one to check the conformity of the models when they differ, which is inevitable during the life cycle, more reliably;

• modify the generation algorithm [8] so that it is possible to affect it to obtain a more obvious and convenient SDL model, instead of the single variant produced in generation;

• improve descriptive characteristics of the MSC, so as to enable one to create more complete models of the MSC level; this is necessary for both generation and verification, as well as for more detailed development of the specifications.

The best tool pair existing today in the market is the pair of Telelogic Tau [16] and the facility of diagram generation KLOCWork[6] [20, 21]. This is an integrated approach that uses both generation and verification (algorithms 3.c, 3.a, and 2.b in our classification). This pair is also not free from the above-mentioned difficulties.

3. THE SUGGESTED DECISIONS

On the basis of the technological tool [4], we have developed a number of algorithms that resolve difficulties stated in Section 2. This tool was supplemented with the following innovations:

• an extended MSC language for creation of more complete MSC descriptions of a system;

• customizable generation from the MSC model to the SDL model;

• an original procedure of conformity checking of the SDL program and the MSC documentation.

3.1. A New Verification Method

We have developed a special verification method for checking conformity of the SDL diagrams to the MSC documentation in the case where there are differences between the MSC and SDL diagrams, but both diagram types are known to correspond to the behavior of the same object. In contrast to other methods, which are based on various demonstrations that the SDL and MSC models are "the same," our method is based on the concept of invariant that is preserved under a certain class of modifications of the MSC and SDL models in the life cycle of the development. Here, the MSC and the SDL models may differ from each other and correspond to each other at the same time, but only until they have a common invariant. Speaking simply, an MSC model determining the trace *abc* corresponds to an SDL model determining the trace *abecd*, but does not correspond to the SDL model determining the trace *abbc*. To explain this, let symbols *a*, *b*, and *c* denote "Question," "Answer," and "Confirmation." Then, the first SDL model is characterized by implementation of some additional functionality that does not break the initial one, and the second model specifies an absolutely different exchange algorithm.

When checking MSC–SDL, we leave only symbols from the set $\{a, b, c\}$ in both models. This set contains all common symbols of the two models. Note that the (reduced) MSC trace will be superposed on the reduced first SDL model, but will not go into the second reduced model. In the real checking algorithm, it is required that the whole finite state automaton from the MSC diagrams could be superposed. This requirement corresponds to the existence of a superposition for every trace; moreover, all traces must start from one point rather that simply exist somewhere in the SDL diagram.

Checking algorithm[7]

1. Successively, for all objects of the SDL diagrams to be checked:

2. Choose an SDL diagram of the object and the set of the MSC diagrams being checked that contain this object.

3. Successively, for all MSC diagrams being checked set them as initial ones.

4. Construct finite-state automata for both diagram types. For the SDL, the initial state is taken to be the initial SDL state, and for the MSC, it is the diagram chosen as the initial one on step 3.

5. Each finite-state automaton must generate a set of traces of the chosen object with relation to the given diagram type. Therefore, all states of the finite-state automata are marked as terminal ones.

6. The set of essential messages, which are those that present in each finite-state automaton, is defined.

7. Replace all other messages with ε-transitions in both automata. From this stage, the finite-state automata are referred to as "reduced."

8. Ascertain whether there is at least one such a state of the reduced SDL automaton such that **all** traces of **any length** from the reduced MSC automaton can be laid starting with it.

9. If there are no contradictions, the corresponding nodes must exist for all MSC diagrams of every object. If there are discrepancies, this will be true for some MSC diagrams of some objects.

[6] An official partner of Telelogic; a part of this project was known as MOST [19].

[7] Details of this algorithm can be found in [22].

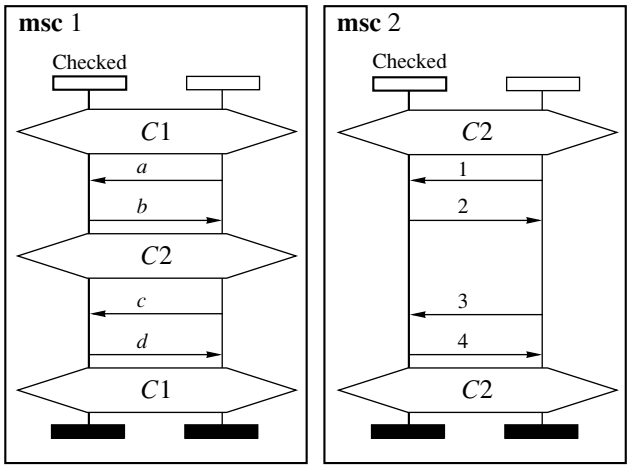

Fig. 4. MSC diagrams 1 and 2 for conformity checking.

Example. Checking for conformity of the SDL diagrams to the MSC documentation under existing differences.

Let MSC specifications 1, 2, 3, and 4 exist. Specifications 1 and 2 are represented in Fig. 4, and specifications 3 and 4, in Fig. 5. Let also an SDL implementation shown in Fig. 6 be given. It is required to analyze conformity of the implementation to various specification sets.

Our method allows one to reveal that the SDL implementation

(1) satisfies MSC specification 1, without taking into consideration specifications 2, 3, and 4;

(2) satisfies MSC specification 2, without taking into consideration specifications 1, 3, and 4;

(3) satisfies specifications 1 and 2 if specification 1 is initial;

(4) satisfies specifications 1 and 2 if specification 2 is initial;

(5) satisfies specification 3; and

(6) does not satisfy specification 4.

3.2. Improvement of the Generation Method for Increasing Readability of the SDL Diagrams

Let us begin this section with an example, which shows different variants of the SDL code that can be generated from the same set of the MSC diagrams. Let the initial MSC diagrams be represented in Figs. 7 and 8, and the SDL results, in Figs. 9 and 10. Using MSC diagrams, we create the SDL diagram that represents the behavior of object 1. Both created SDL diagrams from Figs. 9 and 10 correspond to the initial MSC diagrams. It is obvious that the diagram in Fig. 10 should be chosen between two SDL diagrams as an implementation.

Unfortunately, the automatic generation algorithm [8] will produce another diagram. The cause is the requirement that the automaton must be deterministic.

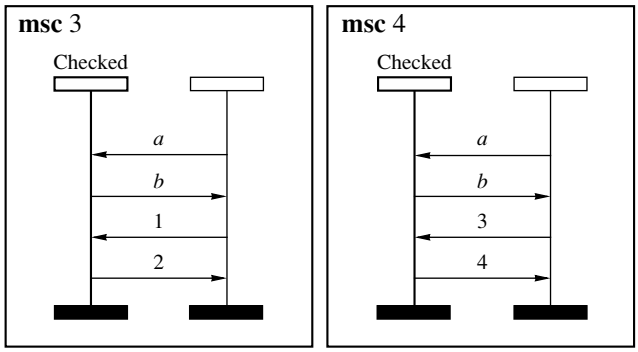

Fig. 5. MSC diagrams 3 and 4 for conformity checking.

This requirement can be weakened: it is sufficient that it is deterministic for incoming signals. From the SDL standpoint, this means that any incoming signal is associated with a certain behavior. However, it is admissible that the same signal was sent at the beginning of two conditional branches, which is shown in Fig. 10 after the uppermost condition.

Let us turn to Fig. 11. This is the finite-state automaton corresponding to the considered example. As can

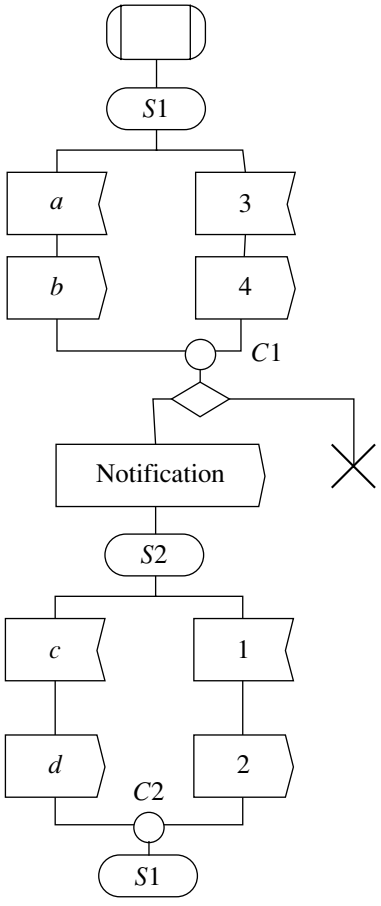

Fig. 6. SDL diagram being checked.

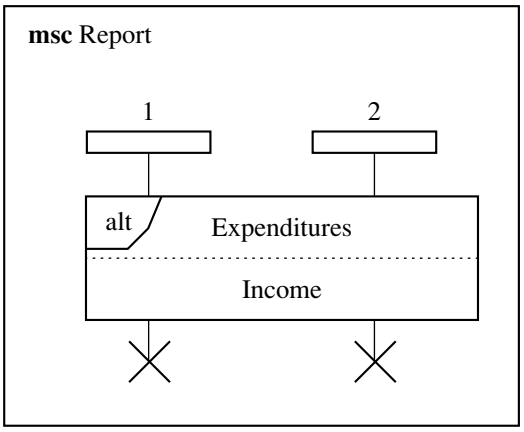

Fig. 7. Main MSC diagram for the example of generation.

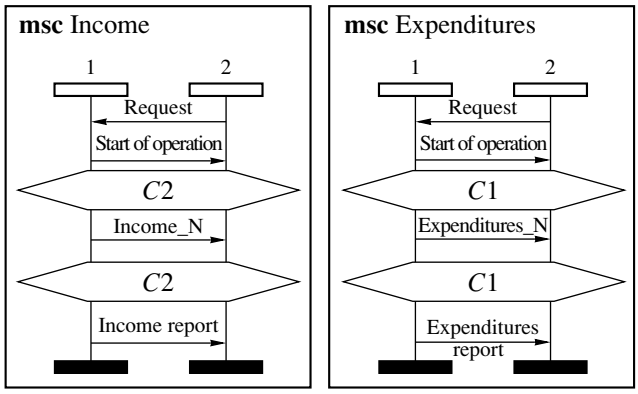

Fig. 8. Detailing MSC diagrams for the example of generation.

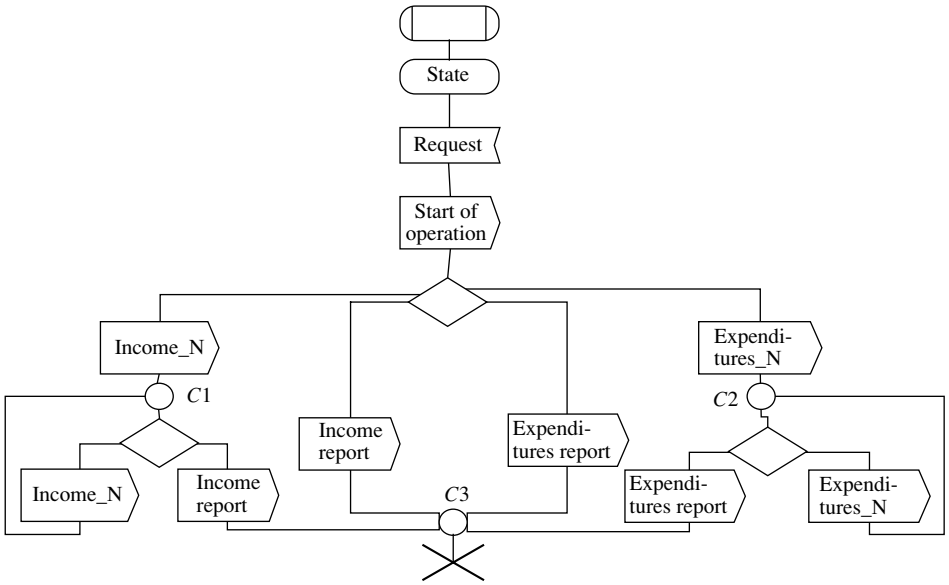

Fig. 9. Classical result of generation.

be seen, it is more compact than the MSC and SDL diagrams. This is because it contains no information about other objects from the MSC diagrams and no information about the decision-making logic of the SDL diagram.

We propose to give this automaton to a technologist as an intermediate model between the MSC and SDL diagrams. Owning to its compactness, it is possible to introduce necessary modifications in it: to "redistribute" the signals, to merge some sections together, to unglues others, to specify procedures, etc. Then, one can check the automaton for possibility of generation according to the criterion formulated in the beginning of the section and develop the SDL code. This gives us opportunity to create interactively several variants of SDL models and choose the most suitable one.

The distinction of the proposed method from the existing generation algorithms is that we managed to save the opportunity of creation of the SDL from all possible MSC diagrams. At the same time, a technologist gets not the single SDL model but selects one from the set of possible models that better suits the internal logic of the system and possesses better "readability."

3.3. Extension of the MSC Language for Improving Its Descriptive Characteristics

The experience of practical use of the MSC diagrams in the development of telephone exchanges shows that a great defect of the MSC diagrams is their orientation toward description of direct branches only. After the structure describing the ideal behavior of the system is designed, it is very difficult to add any new

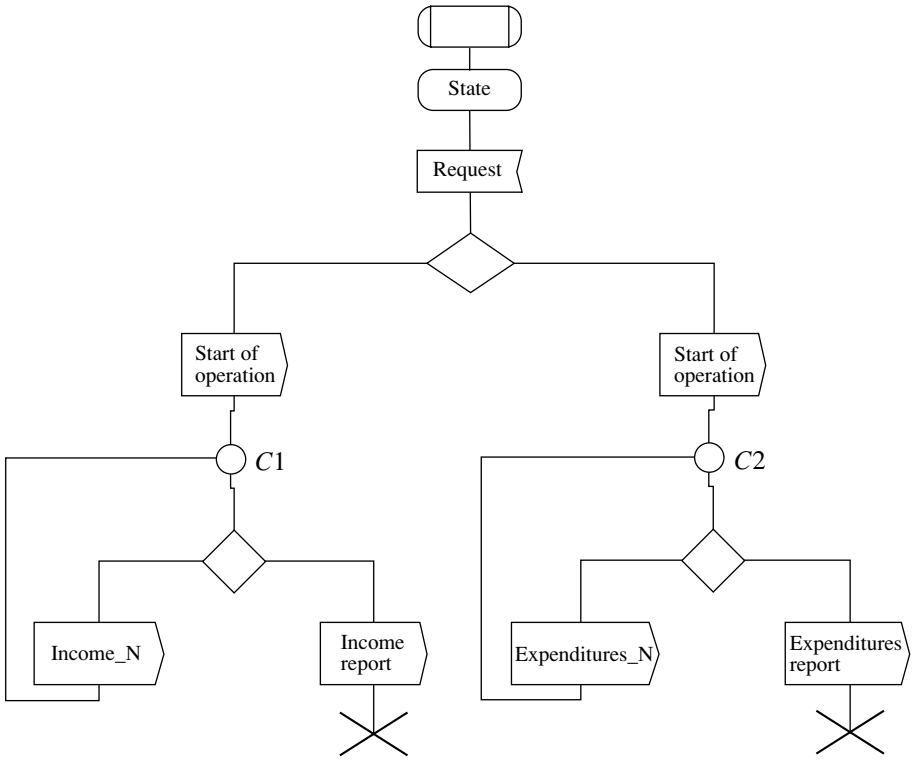

Fig. 10. Our result of generation.

branches implementing processing of various errors. The description of a small section implementing an error situation entails addition of one more interaction variant to those already existing on this level. After that, it is necessary to add a processing variant on the previous level (which was worked out in detail). The error processing often does not terminate on this level, which gives rise to redrawing higher levels as well. Therefore, the description of an additional variant entails much redrawing, and the resulting diagrams grow rapidly as the number of the behavior variants grows. Compared with the programming language Pascal, in the MSC diagrams, it is possible to call a procedure, but there is no opportunity to call a function and obtain the result of its execution. However, in real applications, it is required to foresee the situation where a behavior block can return a number of errors upon its execution. The MSC language has been extended with appropriate possibilities. Along with graphical descriptions, it is proposed to use text descriptions of the following type:

```
function scenario_1(Object1, Obj.2, Obj.3)
{
    if     (scenario_2(Object1,     Obj.2,
Obj.3)!=1) {
            scenario_3(Obj.2, Obj.3);
            scenario_4(Obj.2, Obj.3);
    } else {
            while    (scenario_5(Object1,
Obj.2)==7) {
                if (! scenario_6(Object1,
Obj.2))
                        return 0;
            }
    }
    return 1;
}
```

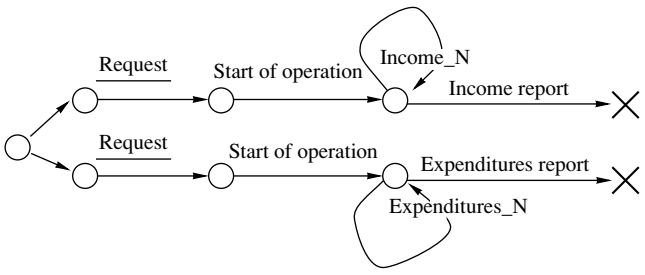

Fig. 11. Automaton derived from the MSC diagrams in Figs. 7 and 8 for object 1.

The idea of construction consists in the following. Suppose that we have a scenario with three execution variants. For each variant, we create the corresponding scenario and number them from 1 to 3. The construct of the form

```
if (scenario(…)!=1)
    A
else
    B;
```

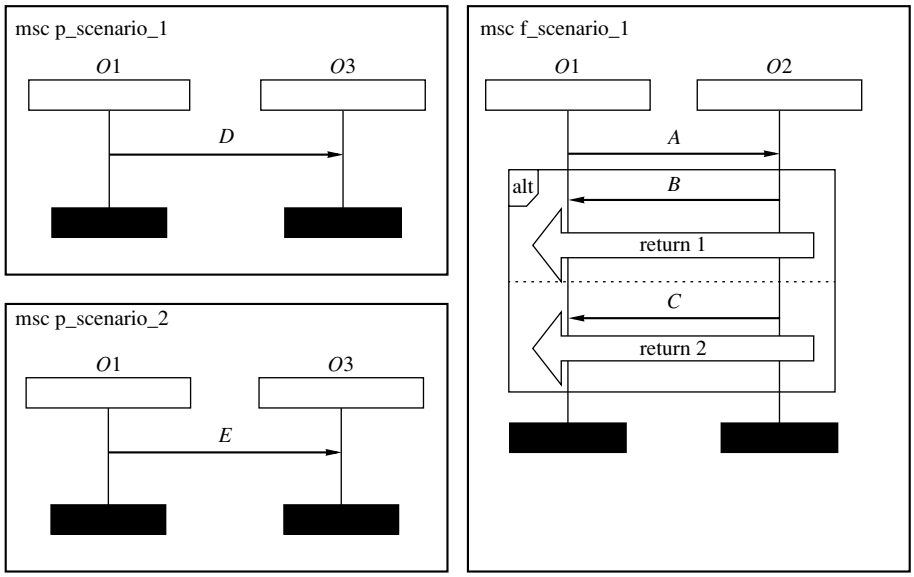

Fig. 12. Two graphic procedures and one graphic function.

can be represented as an ordinary MSC diagram consisting of scenarios 1, 2, 3, *A*, and *B*. Similar constructions can be generated for other operators and scenarios. It is asserted that such an approach corresponds better to the logic of the communication protocols. A man better perceives exchange of messages in a graphic form, whereas the logic of integration of various scenarios is better described by text constructions.

Description of the suggested extension

For text descriptions, we introduce the following operators:

• **if else**,

• **switch**,

• **for**-definite (repeat *N* times),

• **for**-indefinite (repeat 0…∞ times),

• **while**,

• **do while**

with C-like syntax. Neither variables nor numerical operations are introduced.

To describe a return operator, we use the word **return**. The result of this operator is a number. This number is analyzed in the checking operators **if**, **switch**, and **while**. It must be known by the beginning of the compilation, and the result of execution of only one scenario is used.

In the resulting extension, we have the following structure blocks:

• Blocks that return nothing. These are

– ordinary MSC diagrams, e.g., in Fig. 12 (left panel);

– text insertions that do not contain word **return**, e.g., `t_scenario` described below. The headings of such insertions contain word **procedure** with a list of objects participating in this scenario.

We will term them as MSC procedures. The text insertions can be used as an analogue of the MSC text descriptions.

• Blocks that return a result. These are

– MSC diagrams supplemented with a return arrow (such a diagram is represented in Fig. 12, right panel);

– text insertions that contain word **return**, e.g., the scenario mentioned above. The headings of such insertions contain word **function** with a list of objects participating in this scenario.

We will term them as MSC functions. Their distinction from the procedures is that they are supplemented with a tag that contains information about what has happened in the scenario. This information is incomplete and divides all possible variants of the behavior into several classes.

Consider an example. Suppose that we have two ordinary MSC diagrams depicted in Fig. 12 and the following text insertion (which is an MSC procedure):

```
procedure t_scenario(O1, O2, O3) {
  if (f_scenario_1(O1, O2)==1) {
        while (f_scenario_1(O1, O2)>1)
            p_scenario_1(O1, O3);
  } else
        p_scenario_2(O1, O3);
}
```

An analogue of the text description of `t_scenario` in the MSC standard [2] is shown in Fig. 13. The proposed extension should be treated similarly: it gives us opportunity to use text extensions with operators, and a certain MSC diagram will correspond to it. Although compatibility with the standards is important, the pri-

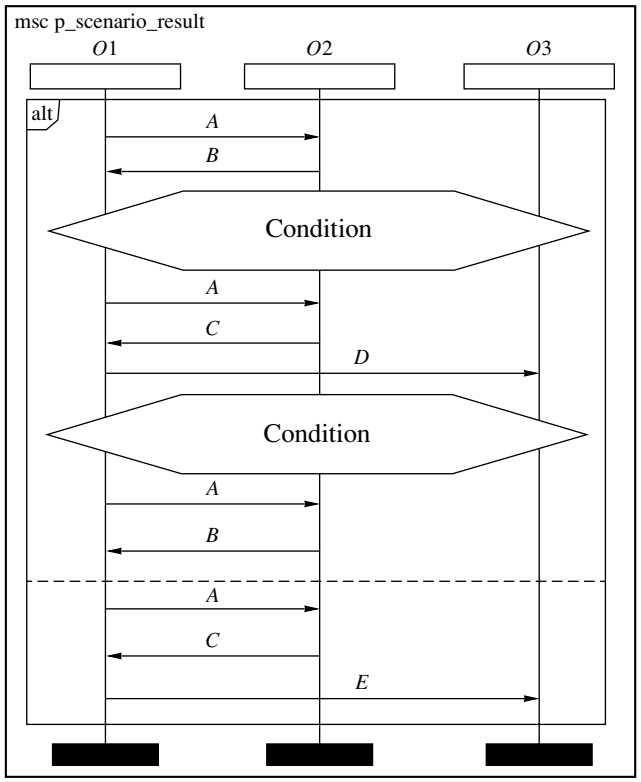

Fig. 13. Result of representation of text description in the MSC standard.

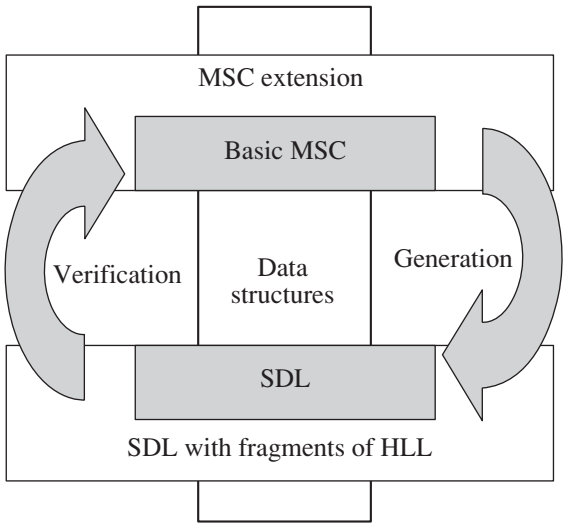

Fig. 14. Integrated approach to conformation of MSC and SDL in our technology today.

mary application of this method of description is generation and verification of the SDL models; therefore, some inconvenience of representation in the MSC standard is allowed. It is clear that the proposed method is designed for working with high-level MSC descriptions without considering what correspond to them in the MSC standard. Representation according to the standard is required only for using the solution in the existing tools.

Comparison with the existing models

Considering the suggested approach in the class of means of description of a system containing interacting objects, we should exclude a number of graphic models, such as SDL [1] and StateChart [5], and the text languages (a review of the latter can be found in [24]), because they provide view from only one object rather than from several ones. Compared to the UML Sequence [5], UML Collaboration [5], UML Activity [5], and Use Case Maps [7] diagrams, our approach is preferable owning to its higher flexibility and, as a result, the possibility of deriving more complete descriptions.

The most similar model is another MSC extension called Life Sequence Charts (LSCs) [23], which is an attempt to adapt the MSC scenarios for description of real systems. Our approach gives us opportunity to describe various behavior variants more conveniently, because LSCs allows one to choose only two basic variants—normal and erroneous—of the scenario completion. It should be mentioned also that the disadvantage of the LSCs model is that this approach deviates greatly from the standard, which requires creating special tools for this model.

4. CONCLUSIONS

In our technology, we have combined the proposed approaches to linking-up the MSC and SDL diagrams. This is represented graphically in Fig. 14.

There, the following points should be emphasized:

(1) one editor for the data structures, which ensures consistency of static parts of the models;

(2) the proposed extension of the MSC diagrams, which enables creation of the MSC models representing real-life situations;

(3) generation functions that allow one to obtain a skeleton of the SDL code from the MSC model, which considerably reduces the development time;

(4) the possibility to extend SDL with fragments of a high-level language for the subsequent generation of the executable code;

(5) the possibility of checking conformity of the SDL diagrams to the MSC documentation upon modifications of any of these models.

In the context of our technology, the approach described above makes it possible to speed up the process of software development and to improve quality of the resulting product due to the maximal coordination of all technology models. We wrote about linking-up of the other models in [4]. The models are chosen in such a way that they basically cover the area of telecommunication systems and real-time systems. Our approach

solves the problems posed in Section 2, and, in the authors' opinion, the proposed integrated solution allows one to successfully develop the MSC and SDL models and ensure their consistency.

The authors believe that such an integrated approach is also applicable in a number of other technologies for filling the gap between an MSC-like model and the SDL model. To this end, the following aspects should be elaborated in a similar way:

• the static parts of the models are coordinated if needed;

• the generation procedure is modified;

• the verification procedure is modified.

The condition imposed on the MSC-like model is that it should be reducible to a finite-state-automaton. In particular, this approach is applicable to other technologies mentioned in Section 2.

REFERENCES

1. *ITU-T Recommendation Z.100: Specification and description language (SDL)*, 1999.
2. *ITU-T Recommendation Z.120: Message Sequence Chart (MSC)*, 1999.
3. *ITU-T MSC2000R3 Draft Z.120: Message Sequence Charts ITU-T Recommendation Z.120*, 1999.
4. Terekhov, A.N. et al., RTST++: Methodology and a CASE Tool for the Development of Information Systems and Software For Real-Time Systems, *Programmirovanie*, 1999, no. 5, pp. 45–51 [*Programming and Computer Software* (Engl. Transl.), 1999, no. 5, pp. 276–281].
5. Rumbaugh, J., Jacobson, I., and Booch, G., *The Unified Modeling Language Reference Manual*, Reading, Mass.: Addison-Wesley, 1999.
6. Server of OMG group engaged in development of UML methodology, *Standards, Publications, and Tools*, http://www.omg.com.
7. *Use Case Maps (UCMs) notation. Definitions, Publications, and Tools*, http://www.useCaseMaps.org/index.shtml.
8. Mansurov, N. and Zhukov, D., Automatic Synthesis of SDL Models in Use Case Methodology, *Proc. of the 9th SDL Forum* (Montreal, Canada, 1999), Amsterdam: Elsevier, 1999, pp. 225–240.
9. Robert, G., Khendek, F., and Grogono, P., Deriving an SDL Specification with a Given Architecture from a Set of MSCs, *Proc. of the 8th SDL Forum* (Evry, France, 1997), Amsterdam: Elsevier, 1997, pp. 197–212.
10. Abdalla, M., Khendek, F., and Butler, G., New Results on Deriving SDL Specifications from MSCs, *Proc. of the 9th SDL Forum* (Montreal, Canada, 1999), Amsterdam: Elsevier, 1999, pp. 55–67.
11. Li, J. and Horgan, J., Applying Formal Description Techniques to Software Architectural Design, *Comput. Commun.*, 2000, vol. 23, no. 12, pp. 1169–1178.
12. Khendek, F. and Vincent, D., Enriching SDL Specifications with MSCs, *Proc. of the 2nd Workshop of the SDL Forum Society on SDL and MSC (SAM2000)* (Grenoble, France, 2000), pp. 305–319.
13. *ETR 184. Methods for Testing and Specification (MTS). Overview of validation techniques for European Telecommunication Standards (ETSs) containing SDL*, 1995.
14. *EG 201 015. Version 1.2.1. Methods for Testing and Specification (MTS); Specification of Protocols and Services; Validation Methodology for Standards Using Specification and Description Language (SDL). Handbook*, 1999.
15. *ETS 300 414. Methods for Testing and Specification (MTS); Use of SDL in European Telecommunication Standards Rules for Testability and Facilitating Validation*, 1999.
16. *Telelogic Tau 4.2 Documentation*, 2001.
17. Sinclair, D., Stone, B., and Clynch, G., An Object Oriented Methodology from Requirements to Validation, *Proc. of the 2nd Object Oriented Information Systems Conference (OOIS'95)* (Dublin, Ireland, 1995), pp. 265–286.
18. Holzmann, G., *Design and Validation of Computer Protocols*, Englewood Cliffs, N.J.: Prentice-Hall, 1991.
19. *Description, Publications, and Contact Information of the Tool MOST*, http://www.ispras.ru/ groups/case/ projects.html?2#2.
20. Site of the company producing the tool KLOCWork, http://www.klocwork.com/.
21. Annotation of the tool KLOCWork on the site of SDL Forum, http://www.sdl-forum.org/Tools/klocwork.htm.
22. Sokolov, V.V., Checking for Conformity of SDL Diagrams to MSC Documentation under Existing Differences, in *Sistemnoe Programmirovanie* (System Programming), St. Petersburg: St. Petersburg State Univ., 2004, pp. 366–390.
23. Damm, W. and Harel, D., LSCs: Breathing Life into Message Sequence Charts, *Proc. of the 3rd IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS'99)* (Florence, Italy, 1999), Dordrecht: Kluwer, 1999, pp. 293–312.
24. Concurrent Lanquages review http://mint.cs.man.ac.uk/ Projects/UPC/Languages/ConcurrentLanguages.html