FORMAL VERIFICATION OF THE SEQUENCE DIAGRAM

Vitaliy Lytvynov, Irina Bogdan

Abstract*: The article describes three different approaches to verification of one of the most frequently used UML-diagrams – the sequence diagram. It indicates that these methods allow estimating its correctness only in certain aspects, and complex application of these approaches is the most effective way.*

Keywords*: verification, digital machine, driver, UML-diagram, record, correctness, sequence diagram.*

Introduction

One of the characteristic features of systems of various nature and destination is the mutual interaction of separate elements that these systems are made of. It means that various constituent elements of the systems do not exist in isolation, but have the influence on each other, and that is the fact that distinguishes the system as a unique formation from the simple complex of elements. [1]. The appropriate interaction diagrams are used for modeling the interaction of objects in UML language. The interaction of objects is frequently considered in time, and then the sequence diagram is used to represent the temporal characteristics of the transmission and the receiving of the messages between the objects.

The message sequence diagram shows the exchange of messages between the objects, the object creation and the response to the messages. The sequence diagrams are used at the analysis stage to illustrate the process in the object domain, so how the tasks are solved in general case due to the interaction of various objects within the script [2].

The message sequence diagram is used at the design stage to describe the algorithms, that is which objects are involved in data processing in order to find a particular aspect of the decision and how these objects interact with each other in order to have an impact on this process. It underlines the importance of the verification of sequence diagram. The basic meaning of the verification of sequence diagram is to make sure that there is a correct exchange of the messages between the objects, which classes have already been, verified previously [3].

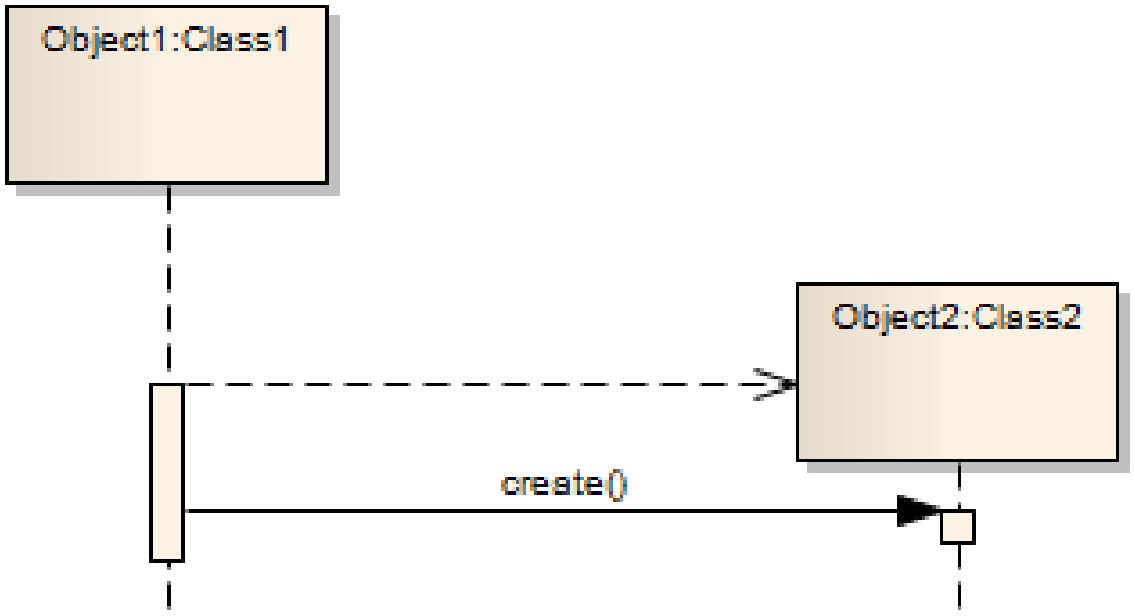
At the moment, there are several different approaches to the verification of sequence diagrams. Most of them allow assessing the correctness of this diagram from different points of view.

The construction of the driver

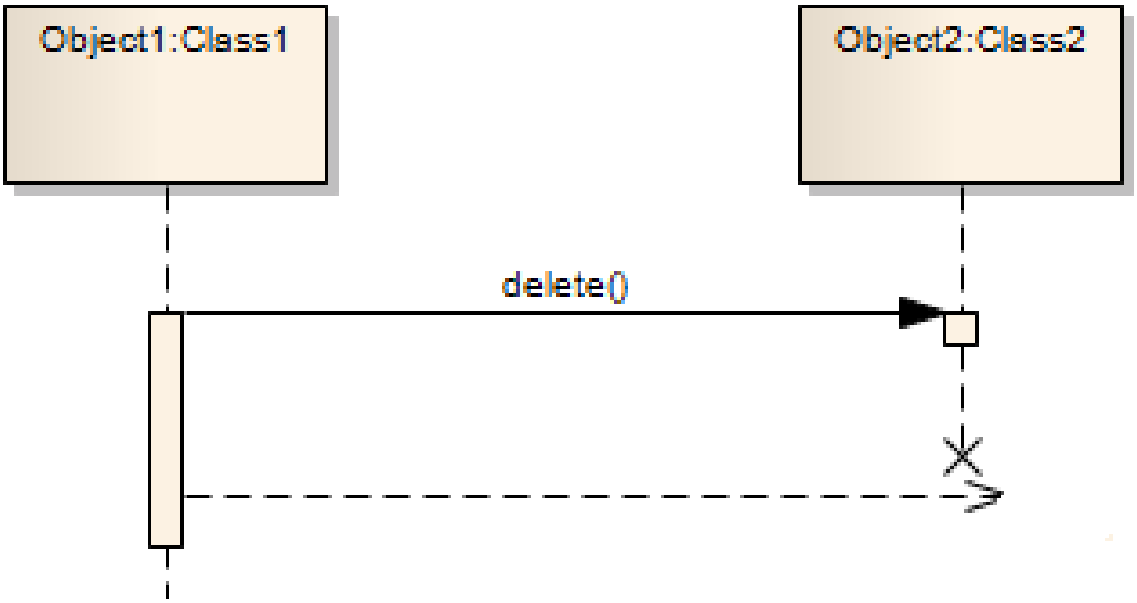
One of the simplest approaches is the one that is associated with the construction of the driver. This approach allows assessing the correctness of the creation of the sequence diagram formally and identifying the most typical mistakes.

As some objects in the sequence diagram will exist permanently, and some - temporary (only during the performance of the required actions), so, first of all, you should assure that the destruction or creation of the objects, that are created for the time of the performance of their actions, is done correctly and explicit messages are provided for them.

Picture 1 shows an example of sending a message to an object before its creation, that is not correct, and Picture 2 shows an example of sending a message to an object after its destruction, that is also not correct.



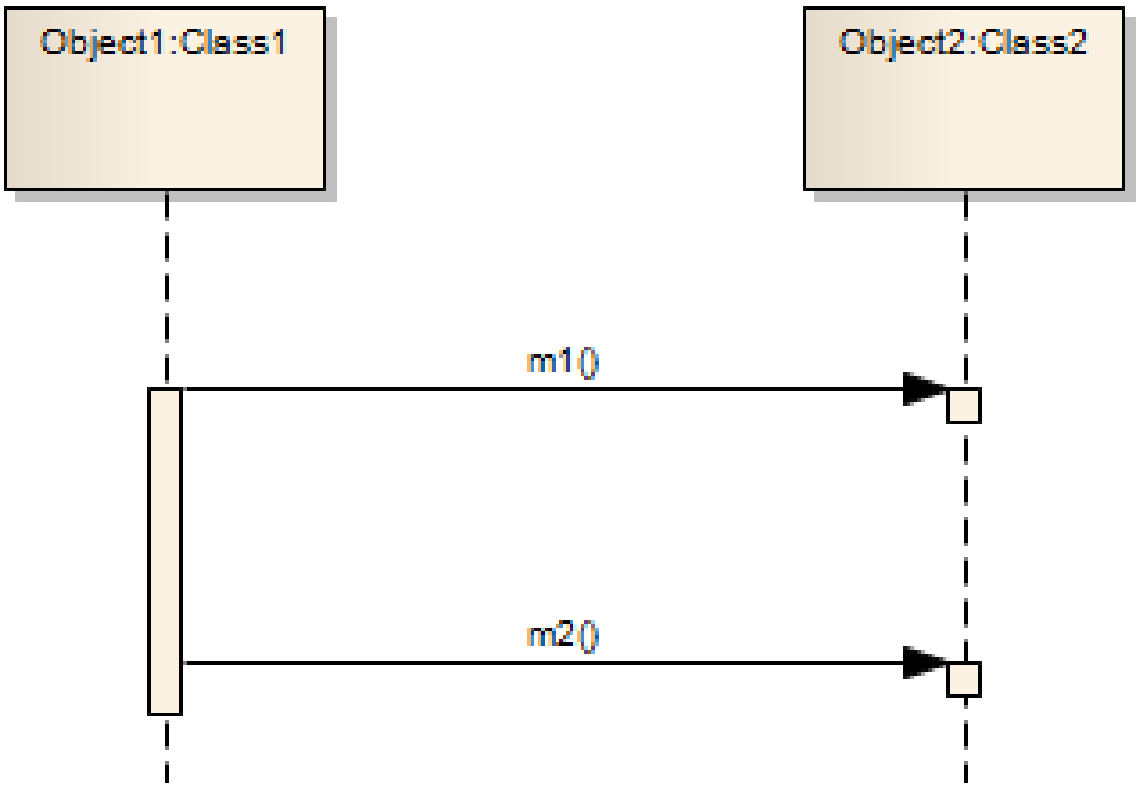
Picture 1. An example of typical error: sending a message to an object before its creation



Picture 2. An example of typical error: sending a message to an object after its destruction

The next step is to perform the actual verification of interaction between the objects. During the verification of interaction under the conditions of contact approach, the attention should be paid to checking, whether the preconditions of methods of the object-recipient are performed by the object-sender. The verification does not take place if these preconditions are violated. The meaning of such checking is to determine whether the object-sender executes the testing of the preconditions of the object-recipient, before posting an unacceptable from the

very beginning message. At the same time, it is checked whether the object-sender stops its activity correctly, possibly, generating an exception. So, for example, according to the definition of synchronous message, the actions of the sender are blocked until it receives the reply. Picture 3 shows an example, according to which the actions of the sender are not blocked after the sending of synchronous message before getting the response, which is not correct.

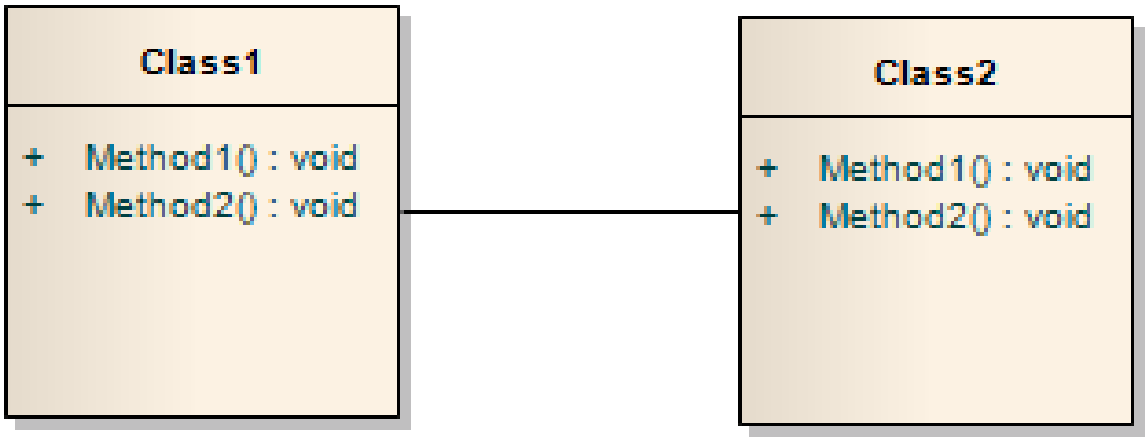


Picture 3. An example of typical error: the actions of the sender are not blocked after sending of synchronous message before getting the response.

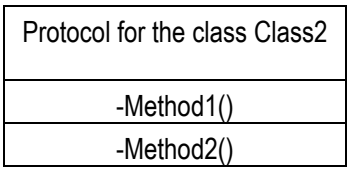
Method of protocols

Verification of the sequence diagrams can also be operated with the method of protocols [2]. As some object starts interaction with other objects the number of messages it gets increases. All these messages are regulated into an intended sequence. Verification on protocol controls fitting of installation into this sequence. Different protocols, that one or another object is involved in, can be inferred out of preconditions and post-conditions which institutionalize the performance of alone operations declared in the class of this object. Identifying sequence of method calls, in which the methods whose post-conditions satisfy the preconditions of the next method are aggregated, build a protocol. Such sequences are picked out on the diagram of classes by analysis of the methods of every specific class. In the case of this approach every alone protocol contains methods of alone class.

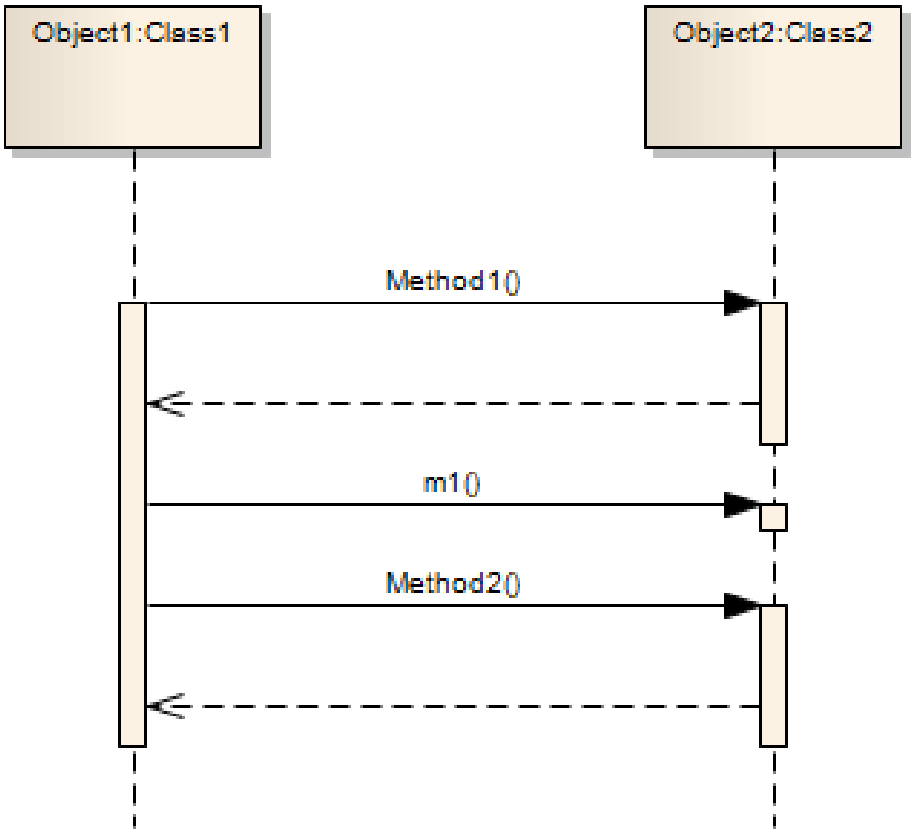
Basically this method of verification is a special form of verification of a life cycle. Every protocol builds a life cycle of verified objects of classes in combination with other classes.



Picture 4. Straightforward diagram of classes



Picture 5. Protocol for the class Class2



Picture 6. Sequence diagram of interaction for the object of class Class2

In picture 4 we are presented an example of straightforward diagram of classes which consists of two classes each of which has two methods. In picture 5 protocol for the class Class2 is presented, and in picture 6 a diagram of the sequence of interaction for the object of class Class2 is presented. According to this diagram the protocol for class Class2 must contain Method1(), m1() and Method2(), but on the assumption of picture 5 m1() does not belong to protocol for class Class2, as a result this sequence diagram is not correct.

Method based on the construction of an abstract digital automat

Another approach to verification of the sequence diagrams is based on the introduction of the sequence diagrams as an abstract digital automaton.

Abstract theory of automatons, digressing from the structure of the automaton and taking no interest in the manner of its construction, studies the way of behavior of the automaton towards the external environment.

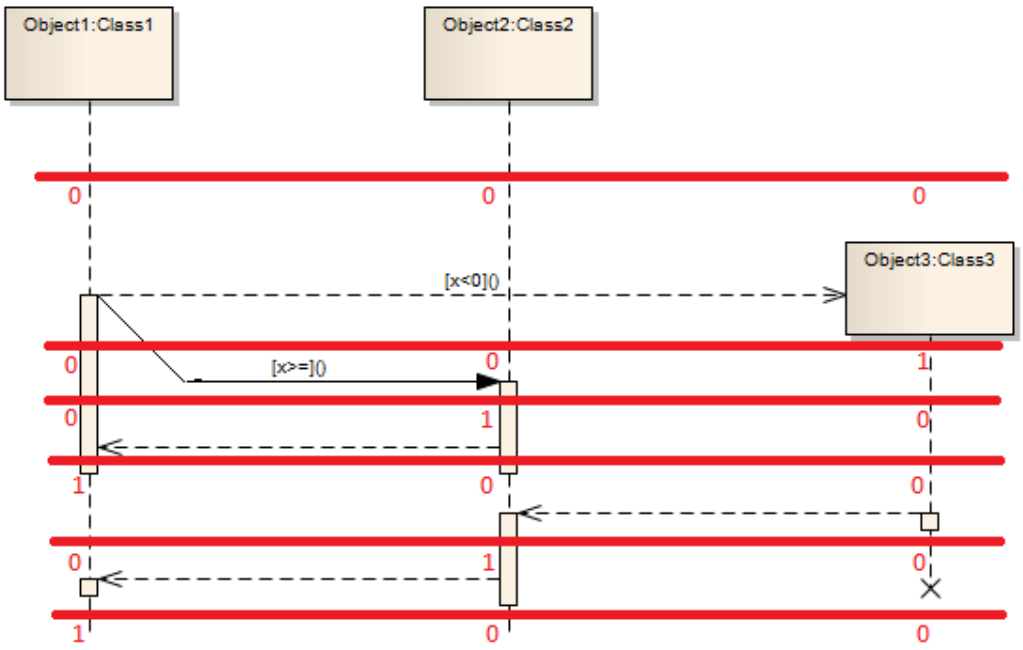
Abstract digital automaton A is defined by a total of five objects {X, S, Y, φ, λ}, where X = { x_i }, i$\in \overline{1,m}$ - is a set of input signals of automaton A (input alphabet of automaton A); S = { s_j }, j$\in \overline{1,n}$ - is a set of states of automaton A (states alphabet of automaton A); Y = { y_k }, k$\in \overline{1,l}$ - is a set of output signals of automaton A (output alphabet of automaton A); φ - is a function of transition of automaton A which indicates the display (X $\times$ S)$\rightarrow$ S, i.e. assigning to any pair of elements of a Cartesian product of sets (X $\times$ S) element of set S; λ - function of outputs of automaton A which either indicates display (X $\times$ S)$\rightarrow$ Y, or display S $\rightarrow$ Y [4].

In this case one automaton describes the sequence of interaction of the objects in one diagram of sequence, where the state means a total of states of all the objects present in the given diagram. Activeness or passiveness of one or another object can be defined (1 or 0) depending on its involvement in the interaction at a certain moment of time.

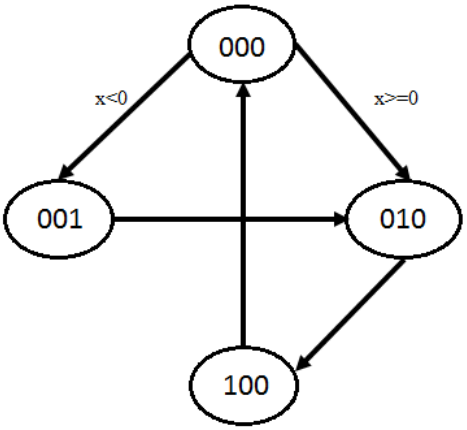
Thus, a set of states of automaton S provides a set of total states of every object shown in the diagram of sequences represented in the binary system. Both sets of input and output signals of automaton X and Y consequently compile a set of messages in the diagram of sequence which make the object either active or passive.

In picture 7 we can see an example of a diagram of the sequence of interaction of three objects with the display of their states in different time intervals.

On graphical method of assigning, as any abstract automaton, the given automaton is represented as a direct graph where the states of the automaton are shown as the points of the graph and the transitions between states as arcs between the corresponding points. Transitions can be both conditional and unconditional. At the moments of time when the objects in the sequence diagram are not yet created or already deleted, they are considered as not active on default (0). Thus, the abstract automaton for the sequence diagram from picture 7 is presented in picture 8.



Picture 7. Sequence diagram of interaction of the objects with the display of their states



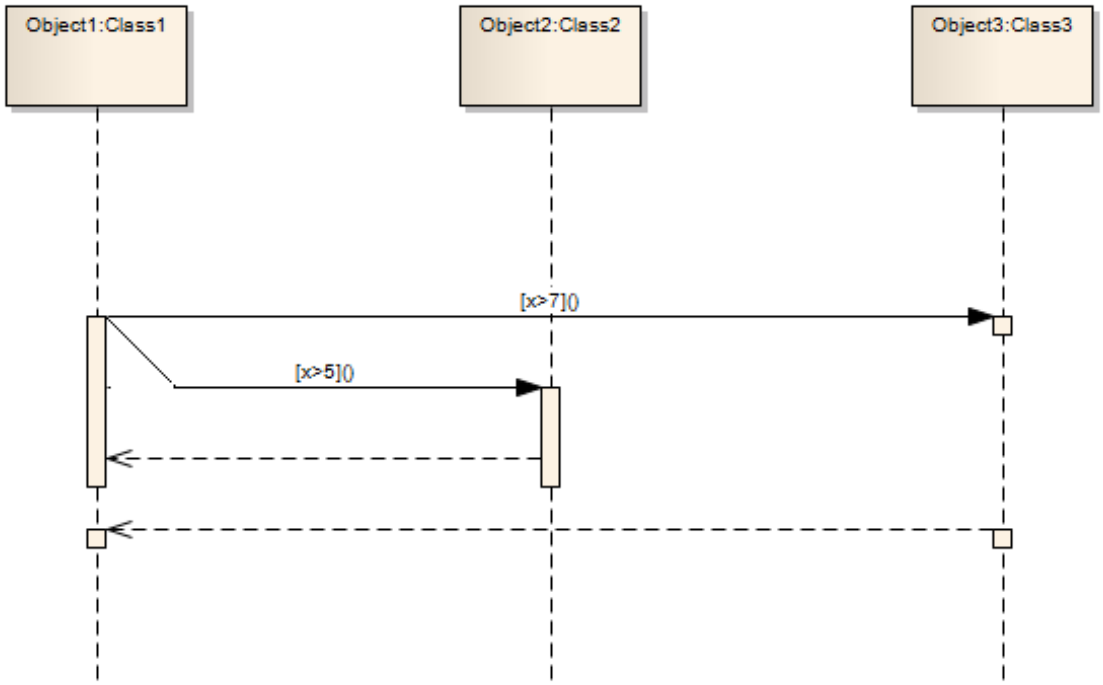
Picture 8. Abstract automaton for the sequence diagram from picture 7

As this approach is based on the introduction of the sequence diagram as digital automaton so the requirements preferred are the same as to any digital automaton:

- At every moment of time the object, which is introduced as a set of states of every object of the sequence diagram, can be in one and only in one of its states. Herewith, the object can be in a certain state as long as it is necessary if no event objects happen that is until new messages come;
- The number of states of an object must be definitely finite (in UML language consideration is given only to finite state automatons) and all of them must be specified in explicit manner;
- An automaton cannot contain isolated states and transitions. This requirement means that for every state, except for the source state, previous state must be defined. Every transition must connect two states of the automaton. Transition from the state into itself is permitted. This kind of transition is also called “self loop”;

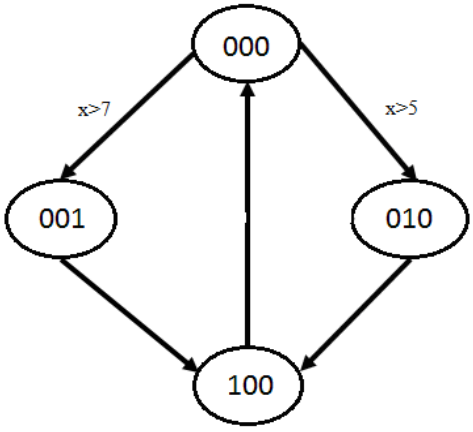
– An automaton cannot contain conflicting transitions that is such kind of transition from the same state when the object can simultaneously transfer into two or more sequent states (except the case of simultaneous sub-automatons). In UML language exclusion of collisions is possible based on typing so called guard conditions (conditional state transitions).

Thus, in particular, conflicting transition can become the branch with ill-conditioned requirement. In picture 9 we can see the sequence diagram and in picture 10 – abstract digital automaton to this diagram. This automaton has conflicting transition: for example, if x=10 then from the state (000) the automaton transfers simultaneously into two states (010) and (001) which contradicts the requirements to construction of the digital automaton. Thus, this sequence diagram is not correct.



Picture 9. The sequence diagram with the branch

The sequence diagram from picture 7 does not have conflicting transitions as the requirement of the branch x<0 and x>=0 do not contradict each other.



Picture 10. Abstract automaton for the sequence diagram from picture 9

Conclusion

The article describes several different approaches to verification of the sequence diagrams. All of them allow us to estimate the correctness of the sequence diagrams from a variety of angles. Thus, the method which is based on the construction of the driver allows us to expose only the most distinctive mistakes in the diagram. Method of protocols allows estimating the correctness of the sequence diagrams formally, observing those messages in it which do not correspond the methods in the diagram of classes. Suggested method is based on the introduction of the sequence diagram as abstract digital automaton allows exposing mistakes in the selection and other conflicting messages. Therefore, the most effective is the complex usage of the given methods

Bibliography

1. Leonenkov A. UML tutorial. Effective instrument of modelling of information systems / Leonenkov A. – Spb.: BHV – St. Petersburg, 2001. – 304p.
2. McGregor J. Testing Object-Based Software: practical guide. / J. McGregor, D. Sykes; translated from English – K.: LLC "TID"DC"", 2002. – 432p.
3. Lytvynov V.V. Formal verification of the class diagrams / V.V. Lytvynov, I.V. Bohdan// Academic periodical "Mathematical machines and systems" № 2. – 2013. – P. 41-47.
4. Samofalov K.G. Applicable theory of digital automatons / K.G. Samofalov, A.M. Romankevych< V.N. Valuiskyi, Y.S. Kanevskyi, M.M. Pynevych – K.: Head publishing house of the publishing association " Vyscha shkola", 1987. – 374p.

Authors' Information

Irirna Bogdan *– Postgraduate, the assistant lecturer, Chernihiv National Technological University, 95, Shevchenko street, Chernihiv-27, Ukraine, 14027; e-mail: irakirienko@gamil.com*

Major Fields of Scientific Research: design of object-oriented software, management software projects and their risks, expert systems

Vitaliy Lytvynov *– Dr. Sc. Prof. Chernihiv National Technological University, 95, Shevchenko street, Chernihiv-27, Ukraine, 14027; e-mail: vlitvin@ukrsoft.ua*

Major Fields of Scientific Research: modeling of complicated systems, computer-aided management systems, decision support systems