View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Tomsk State University Repository

Applying a Model Based Testing Approach for Testing the Communication Protocol between the Cash Register Software and the Loymax Service

Maria S. Forostyanova[1], Natalia V. Shabaldina[2], Nina V. Yevtushenko[3]
Department of Information Technologies for Studying Discrete Structures,
Tomsk State University,
Tomsk, Russia
[1]mariafors@mail.ru, [2]nataliamailbox@mail.ru, [3]nyevtush@gmail.com

***Abstract* — In this work we apply test derivation methods for (extended) finite state machines for testing the functionality of the Communication Protocol between the cash register software and the Loymax service when conducting cash transactions. The Protocol was provided by Loymax that is the company involved in the development and support of loyalty programs. We analyze the difficulties that occur when we extract a formal model from the description of the system and we suggest different ways for simplify the derived models; we also discuss further directions for this investigation.**

Keywords—web-services, Loymax, EFSM, FSM, model based testing

I. Introduction

Testing of web-services is still a hot topic in spite of the plenty of tools for automatic generation [see, for example, 1-2] and applying test suites. The reason is that most test suites are derived based on intuition and experience of a tester. This approach does not guarantee that the derived test suites are complete. Correspondingly, in this work we study the quality of tests which are derived based on the model of an (extended) finite state machine; such tests show their effectiveness when testing telecommunication protocols [3,4]. Under the web-services we mean applications that are working in WEB in different network nodes and the logic of data exchange between applications is set by some protocol.

As an example for web service testing, we took the one provided by Loymax [5]. Loymax is the company that has been automating loyalty programs which are in turn aimed at implementing a set of marketing events to promote repeat purchases, increase traffic by attracting new customers and contribute to other potentially profitable developments. The Loymax IT-platform is intended for implementation of multi-format loyalty programs and allows one to raise the efficiency in producing important and effective solutions. The system has universal APIs for integration with external systems – e.g., cash register software. By this time, the Loymax platform has been integrated with the major cash software vendors; there are about 15 vendors. In this regard, relevant is the qualitative testing of this web service. The integration solution provides a possibility of processing within the online and offline modes of operating, which ensures preservation of the client's preferences even in the case of a disconnection and a lack of information exchange between the cash register and the processing unit. Thus, Loymax is interested in a strict conformity with the Protocol requirements presented in the specification and this requires the use of formal models and methods for testing.

This paper provides a short description of the Communication Protocol between the cash register software and the Loymax service when conducting cash transactions and presents an extended automata model developed based on this description. For applying methods for constructing tests with the guaranteed fault coverage, we modeled a service using an extended finite state machine (EFSM) taking into account some limitations. Further, a classical finite state machine (FSM)) was derived and we constructed a test by passing over the transition graph and, despite the fact that theoretically such a test guarantees the detection of only output errors, this test enabled to reveal four errors in the web service implementation being tested, one of which was critical. In the future we are planning to construct tests for the finite-state machine developed by methods that guarantee the detection of transition errors [6].

II. The Description of the Communication Protocol between the Cash Register Software and the Loymax Service when Conducting Cash Transactions

Within the five years of existence in the market of IT solutions Loymax has released about six major and five minor versions of the Communication Protocol for cash register software, taking into account specific features of processing in different business sectors. The testing presented in this paper was conducted based on Protocol 3.0. This Protocol is used for communication of the cash equipment/software with the Loymax system where the formation of an XML document and the use of an HTTPS protocol are possible. When using the Loymax processing unit for transactions and payments one must submit a request in a strictly specified format to the cash register. The request is an XML document. The document can contain a list of commands and the server processes each individually, i.e., if an error occurs while processing one of the commands in the document, the rest will be processed independently. If the command is sent to the server again (the command field coincides with the one previously processed), the server will return the result of the previous processing step. This can be used when the communication with the server is lost; in this case, it

978-1-5090-1081-3/17/$31.00 ©2017 IEEE

is enough to repeat the only command. If the command is sent with the fields partially changed, the system will return an error to the changed parameters within the old receipt.

Almost all commands require confirmation if the result is accepted by the cashier. In other words, after processing (receipt printing) it is necessary to send the command to confirm or cancel the receipt for transferring the operation to the specified status. Each of the lists can contain an unlimited number of commands. The order of the commands being discussed in this work is presented below:

1) *Calculates* – calculation of a direct discount (can be performed without a loyalty card);
2) *AvailableAmounts* – calculation of the available amount on the card account to pay for the current receipt;
3) *Payments* – calculation of the payment with this card for the current receipt;
4) *Discounts* – calculation of a pending discount with the loyalty program card (cashback accrual);
5) *ConfirmPurchases* – confirmation of all operations for this transaction;
6) *CancelPurchases* – cancellation of all operations for this transaction.

Complete test suites generation for the software is possible only in case of using the fault model, thus in this work we are trying classical finite transition models as specifications, namely, EFSM and FSM.

III. EXTRACTING EFSM FROM WEB-SERVICE DESCRIPTION

Finite state machines are studied very well [7], in particular, test suite derivation methods are well known for finite state machines (including non-deterministic machines).

An (initialized) FSM $\mathcal{S}$ is a 5-tuple (S, I, O, T_s, s_0), where S is the set of states with the designated initial state s_0, I is an input alphabet, O is an output alphabet, T is a transition relation, $T \subseteq S \times I \times O \times S$.

However, classical FSMs do not allow the explicit description of different parameters, such as input or output parameters, predicates for transitions. For this reason, extended FSMs are often used for describing the behavior of protocols and services [8]. An EFSM model allows describing the system behavior in a compact way (in terms of transition number). An EFSM model in addition to the sets of state, inputs, outputs and transitions has the set of context variables (the values of these variables determine the current state of the system), the set of input parameters (in order to determine the current input), the set of outputs parameters (in order to determine the current output), and predicates.

More precisely, an EFSM $\mathcal{M}$ is a 5-tuple (M, I, O, V, T) [3], where M is the set of states, I is the set of inputs, O is the set of outputs, V is the set of context variables (this set can be empty), T is the set of transitions. Each transition $t = (m, x, P, op, y, up, m')$ where $m, m' \in M$ are the initial and final states of the transition, $i \in I$ is an input and $D_{inp\text{-}i}$ is the set of vectors with parameters values that correspond to the input i (input parameters). Correspondingly, $o \in O$ is an output and $D_{out\text{-}o}$ is the set of vectors with parameters values that correspond to the output o (output parameters). Functions P, op and up are defined on the Cartesian product of the sets of input parameters and context variables:

- P: $D_{inp\text{-}i} \times D_V \rightarrow$ {true, false} is a predicate where D_V is the set of context vectors, the components of these vectors correspond to the values of context variables;
- op: $D_{inp\text{-}i} \times D_V \rightarrow D_{out\text{-}o}$ is a function for calculating the values of output parameters;
- up: $D_{inp\text{-}i} \times D_V \rightarrow D_V$ is a function for calculating the values of context variables.

For the protocol describing the interaction of cash software with Loymax we determine the following input and output alphabets:

- *AllCheque* is a receipt with mandatory parameters:

  - *ChequeNumber* is the receipt number (must be unique within the cash register);
  - *ChequeDate* is the receipt date;
  - *ChequeLine* is the receipt line including information on the position number, the product quantity, the product ID, the product name, the total value of the position;
  - *Number* is the card number in the Loymax database;
  - *PurchaseID* is the unique transaction ID which is unique within the entire system and must be the same within the receipt at all stages of communication;
- *PayAmount* is the amount to be paid

Output symbols:

- *Error* is the message in the case of an error at the Protocol level;
- *CardHolder* is the Loymax cardholder;
- *list(chequePositionDiscount)* is the direct discount amount for each *ChequeLine*
- *list(chequePositionCashback)* is the pending discount amount for each *ChequeLine;*
- *list(chequePositionPayAmount)* is the payment amount for a position for each *ChequeLine;*
- *BonusAmount* is the number of bonus points in the customer's account available for payment for the current receipt;
- *CashierMessage* is the message to the cashier on the checkout screen;
- *ChequeMessage* is the message print to the receipt.

Context Variables:

- *chequeNumber*, *chequeDate, chequeLine*, *purchaseId, cardNumber* are the receipt parameters; the transaction ID and the card number must not be changed within the receipt, except for the adoption of the Protocol functions at the stage of communication;
- bonusAmount is the number of bonus points in the customer's account available for payment for the current receipt;

States:

- *Open* is the state corresponding to opening the receipt in the cash register, initialization of the receipt and context variables;
- *Calculates* is the state of calculating a direct discount for the receipt;
- *AvailableAmounts* is the calculation of the maximum amount for paying by bonus points;
- *Payments* is the state for debiting the loyalty points from the account with a decrease in total (the proper amount of bonus points is blocked in the Loymax cardholder's account);
- *Discounts* – calculation of a pending discount for the receipt (the proper amount of bonus points is blocked on the cash register account);
- *ConfirmPurchases* is the confirmation of the transaction; all financial flows are reversed and the bonus points are unblocked;
- *CancelPurchases* is the cancellation of the transaction; all financial flows are performed and the bonus points are credited to the accounts;
- *CalculateReqError* is the state corresponded to the Protocol error when calculating a direct discounts for the receipt;
- *AvailableAmountsError* is the state corresponded to the Protocol error when calculating the maximum amount to be paid by the bonus points;
- *PaymentsError* is the state corresponded to the Protocol error when paying by the bonus points;
- *DiscountsError* is the state corresponded to the Protocol error when calculating a pending discount for the receipt.

The list of transitions is as follows.

t1: *AllCheque (ChequeNumber, ChequeDate, ChequeLine, PurchaseID, Number)/ /chequeNumber=Cheque.Number, chequeDate=ChequeDate, chequeLine=ChequeLine, purchaseId = PurchaseID, cardNumber = Number/NULL*

t2: *AllCheque (ChequeNumber, ChequeDate, ChequeLine, PurchaseID, Number)/ ChequeNumber==ChequeNumber AND chequeDate==ChequeDate AND chequeLine==ChequeLine AND purchaseId == PurchaseID AND cardNumber == Number / chequeLine = chequeLine - list(chequePositionDisount) / list(chequePositionDiscount)*

t3: *AllCheque (ChequeNumber, ChequeDate, ChequeLine, PurchaseID, Number)/ ChequeNumber!=ChequeNumber OR chequeDate!=ChequeDate OR chequeLine!=ChequeLine OR purchaseId != PurchaseID OR cardNumber != Number OR check(card)==false OR PurchaseID is not unique in the system/ / Error*

t4: *AllCheque (ChequeNumber, ChequeDate, ChequeLine, PurchaseID, Number)/ ChequeNumber==ChequeNumber AND chequeDate==ChequeDate AND chequeLine==ChequeLine AND purchaseId == PurchaseID, cardNumber == Number / bonusAmount = BonusAmount / BonusAmount*

t5: *AllCheque (ChequeNumber, ChequeDate, ChequeLine, PurchaseID, Number)/ ChequeNumber!=ChequeNumber OR chequeDate!=ChequeDate OR chequeLine!=ChequeLine OR purchaseId != PurchaseID OR (cardNumber != Number AND type(card)!='gift')OR check(card)==false / / Error*

t6: *AllCheque (ChequeNumber, ChequeDate, ChequeLine, PurchaseID, Number, PayAmount) / ChequeNumber==ChequeNumber AND chequeDate==ChequeDate AND chequeLine==ChequeLine AND purchaseId == PurchaseID AND cardNumber == Number AND PayAmount <= bonusAmount / / list(chequePositionPayAmount)*

t7: *AllCheque (ChequeNumber, ChequeDate, ChequeLine, PurchaseID, Number, PayAmount)/ ChequeNumber!=ChequeNumber OR chequeDate!=ChequeDate OR chequeLine!=ChequeLine OR purchaseId != PurchaseID OR (cardNumber != Number AND type(card)!='gift') OR check(card)==false OR PayAmount > bonusAmount/ /Error*

t8: *AllCheque (ChequeNumber, ChequeDate, ChequeLine, PurchaseID, Number) / ChequeNumber==ChequeNumber AND chequeDate==ChequeDate AND chequeLine==ChequeLine AND purchaseId == PurchaseID AND cardNumber == Number / /list(chequePositionCashback)*

t9: *AllCheque (ChequeNumber, ChequeDate, ChequeLine, PurchaseID, Number) / ChequeNumber!=ChequeNumber OR chequeDate!=ChequeDate OR chequeLine!=ChequeLine OR purchaseId != PurchaseID OR cardNumber != Number OR check(card)==false / / Error*

t10: *PurchaseID / / NULL*

t11: *PurchaseID / / NULL*

t12: *PurchaseID / chequeNumber = NULL, chequeDate = NULL, chequeLine = NULL, purchaseId = NULL, cardNumber = NULL / NULL*

t13: *NULL / chequeNumber = NULL, chequeDate = NULL, chequeLine = NULL, purchaseId = NULL, cardNumber = NULL / NULL*

t14: *NULL / chequeNumber = NULL, chequeDate = NULL, chequeLine = NULL, purchaseId = NULL, cardNumber = NULL / NULL*

t15: *NULL / chequeNumber = NULL, chequeDate = NULL, chequeLine = NULL, purchaseId = NULL, cardNumber = NULL / NULL*

t16: *Cheque, PurchaseID, Number, PayAmount / ChequeNumber==ChequeNumber AND chequeDate==ChequeDate AND chequeLine==ChequeLine AND purchaseId == PurchaseID AND cardNumber == Number AND PayAmount == 0 / chequeList = ChequeList - list(chequePositionDisount) / list(chequePositionDiscount*

An EFSM extracted based on the above description of exchange protocol software cash with Loymax is shown in Figure 1.

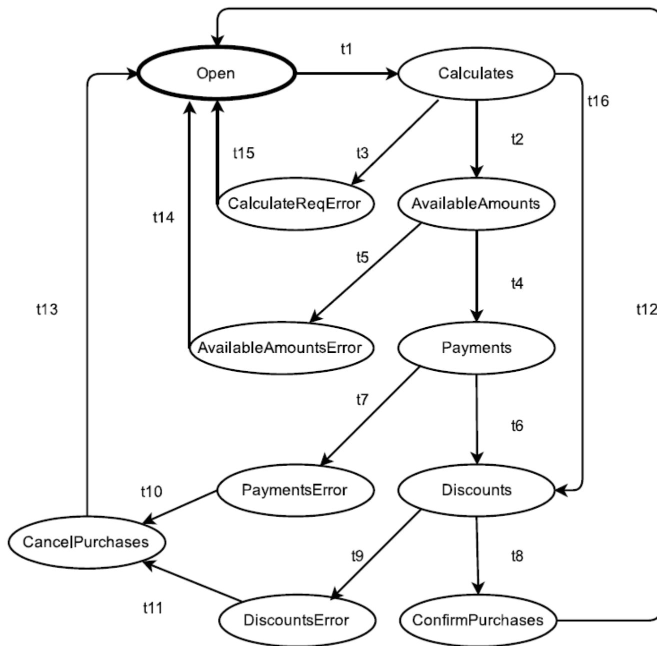

Fig. 1. Extended finite state machine for Loymax protocol

IV. Constructing FSM Based on EFSM and Deriving test Suite by Traversing the Transition Graph

After extraction the model with finite number of transitions from the description of the system we can construct a test suite using one of the classical methods that guarantee the fault coverage. One of such methods is a transition tour for the finite state machine model [9].

When we construct an FSM based on the given EFSM, we match an input symbol in FSM to the pair: input symbol in EFSM and the values of the input parameters vector. So the number of inputs in FSM depends on the number of input parameters values. Similar, each output symbol in FSM corresponds to the pair: output symbol in EFSM and the values of the output parameters vector. And each state in FSM corresponds to the pair: state in EFSM and the values of context variables. In order to construct finite state machine for the given extended finite state machine we need to use some restrictions. In this work, for the system at hand we propose the following restrictions.

For the input parameter values:

- ‘1’ corresponds to the case when the value is the same as at the moment of service initialization, or the value satisfies the requirements of the processing (for example, PurchaseId is unique);
- ‘0’ corresponds to the case when the value is different from the value at the moment of service initialization, or the value does not satisfy the requirements of the processing.

Consider the following transition:

t1: *AllCheque (ChequeNumber, ChequeDate, ChequeLine, PurchaseID, Number)/ /chequeNumber=Cheque.Number, chequeDate=ChequeDate, chequeLine=ChequeLine, purchaseId = PurchaseID, cardNumber = Number/NULL.*

This transition corresponds to the transition from the initial state; there are no predicates for this transition. In finite state machine, it will be the input configuration (1, 1, 1, 1, 1). All context variables are equal to 1. The next transition contains the request to the server:

t2: *AllCheque (ChequeNumber, ChequeDate, ChequeLine, PurchaseID, Number)/ ChequeNumber==ChequeNumber AND chequeDate==ChequeDate AND chequeLine==ChequeLine AND purchaseId == PurchaseID AND cardNumber == Number / chequeLine = chequeLine - list(chequePositionDisount) / list(chequePositionDiscount).*

In this input vector we need to consider all input parameters and check predicates:

- (0, 1, 1, 1, 1) – this transition corresponds to the case when the receipt number that is sending to the server is different from the receipt number that has been sent before;
- (1, 0, 1, 1, 1) – the receipt time is different from the receipt time that has been sent before;
- (1, 1, 0, 1, 1) –the receipt positions are different from the receipt positions that has been sent before (in this work we consider only the case when the number of receipt positions are different);
- (1, 1, 1, 0, 1) – the transaction identifier is different from the identifier that has been sent before;
- (1, 1, 1, 1, 0) – the card number is different from the card number that has been sent before.

There can be also the following combinations from (0, 0, 0, 0, 1) to (1, 1, 1, 0, 0), except combinations that were described above, however, we will not consider this combinations in finite state machine model since for all of them the output ‘exception’ will be returned.

Now we describe the list of additional output responses. These responses differ from each other because of the different error codes and error messages.

The list of additional output symbols contains the following outputs:

- *Er1* is an error message for a different receipt number;
- *Er2* is an error message for the different receipt time;
- *Er3* is an error message for different receipt positions;
- *Er3* is an error message for a different transaction identifier;
- *Er4* is an error message for a different card number.

For the simplifying the visual representation we consider these output reactions as different outputs, since there are

different response models at each stage of interactions. For example, if there is an error in calculating the discount, then the response model is *<CalculateRequest> Er1 </CalculateRequest>*; if there is an error at the payment stage then the response model is *<PaymentRequest> Er1 </PaymentRequest >.*

And we also use the following restriction: if the value of the context variable is updated at the transition, as for example at *t*2 where in the case of correct transitions of all the data the discount has been calculated, then the value of the parameter *ChequeLine* will be changed according to the discount value. Thus «1» for all further requests corresponds to the execution of the operation *chequeLine = chequeLine - list(chequePositionDisount.*

Consider the following transition:

t6: *AllCheque (ChequeNumber, ChequeDate, ChequeLine, PurchaseID, Number, PayAmount) / ChequeNumber==ChequeNumber AND chequeDate==ChequeDate AND chequeLine==ChequeLine AND purchaseId == PurchaseID AND cardNumber == Number AND PayAmount <= bonusAmount / / list(chequePositionPayAmount).*

In this transition, the new parameter *PayAmount* is added and for this parameter, the rules and restrictions described above take place:

- 1 – the amount to be paid is available for this account;
- 0 – the amount to be paid is more than the receipt sum or is not available for this account.

Additional outputs

- *Er6* means the message of incorrect amount for paying by bonus points for the receipt under processing taking into account the state of card account.

Based on the above rules, we construct an FSM with 26 states and 69 transitions and then derive a transition tour for this FSM. The total length of all test sequences (test cases) is 221 symbols. The minimum length of a test case is three, the maximum length of a test case is eight. We apply the test suite half-automatically using Microsoft Visual Studio and NUnit (version 3.0.6, the last version).

All requests were emulated in the same device within the test environment and processed by the processing unit sequentially. The marker of change in the contextual variables, e.g., generation of a new *PurchaseId*, change in the card number, the receipt number, etc., was the input vector (-, -, -, -, -). When testing, two types of preferences were installed in the emulator: a direct discount and a pending discount (bonuses). As an option added to one of the cards, there was an action condition for a personal offer.

V. Analysis of the Testinf Results

When testing the IT-solutions of loyalty programs, the following errors and inconsistencies in implementations of the Communication Protocol with the cash register software were detected:

1) The error of the system if the request *<CalculateRequest>* was not accompanied with the loyalty card.

Criticality of the error is high, as the system can also handle requests to calculate direct discounts without loyalty cards.

This error was caused by the personal offer added to the system and an available access to the context variable without checking it for presence:

Exception Message from system after request: Object reference is not set to an instance of an object.

The error was detected in the new version which is only being prepared for release, and will be eliminated before putting it on production.

2) No checking for change in the receipt date and number in the requests *<AvailableAmountRequest>* and *< PaymentRequest >.*

Criticality of the error: medium.

In a similar request of <DiscountRequest> (by its processing principle) such a procedure is implemented and the given exception returns. According to the business requirements, as well as the formal description of the Protocol, it is necessary to track changes in two variables and return an error at the Protocol level.

3) An incorrect principle of operating when sending different cards within the same *PurchaseID* with paying for the receipt.

Criticality of the error: medium.

An exception for the error that within one *PurchaseID* some cards were handed and they do not meet the requirements that only one card can be the primary while the others are gift cards for which multiple payments are acceptable, only at the stage of confirmation of the purchase when the customer concerned may not already be at the point of sale.

Requirements for the development: move the module with the card checking for multiple payments to the stage of calculation of a direct discount.

4) An incorrect principle of operating when sending different cards within the same *PurchaseID* without paying for the receipt.

Criticality of the error: above medium.

Based on the requirements for integration with the external cash register software, one of the points requires that the card used in the requests should be the same (except for gift cards). Meanwhile, within the system there is a reversed situation of operating with data which is incorrect according to the formal requirements. Among other faults, there is an optional web interface which specifies the card number for each transaction and instead of the card sent in the request *<CalculateRequest>* it displays the card of *<DiscountRequest>* which is an incorrect operation. This error also allows the loyalty program customers to find a loophole for obtaining the highest level of preference by using more cards that are advantageous at each stage of

processing the receipt; this may result in financial losses for the company.

Therefore, deriving test suites based on formal model allow us to find 1 critical error and 3 not critical errors in implementations of the exchange protocol of cash software with Loymax which is using almost everywhere in more than 20 companies and operated with about 600 000 receipts per day. Software developments are now working at the fixing these mistakes.

VI. CONCLUSIONS

In this paper, we have studied the applicability of formal models, such as extended finite state machine and classical finite state machine, for describing the behavior of a proper web-service in order to derive tests with the guaranteed fault coverage. We considered the communication protocol of cash software with Loymax. We extracted an EFSM from the description of this web-service with 11 states and 16 transitions. Then based on this EFSM an FSM was constructed excluding some EFSM configurations. In order to avoid the enumeration of all possible values from database for checking of its uniqueness in the whole system, we suggested to restrict input parameter values to «0» or «1» which are related to the conditions «coincide» and «not coincide» in the current receipt. Thus, using the rules and properties of the system, we restricted the input alphabet, and the size of the corresponding FSM. A transition tour of the FSM allowed to find one critical error and two not-critical errors and one error that may may result in financial losses for the company. In our future work we are planning to derive the test suite using HSI-method that guarantees the detection not only of output faults but also of latent transition faults.

REFERENCES

[1] L. Bentakouk, P. Poizat, F. Zaïdi, “A Formal Framework for Service Orchestration Testing Based on Symbolic Transition Systems”, In: Núñez M., Baker P., Merayo M.G. (eds) Testing of Software and Communication Systems. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, vol 5826, pp.16-32, 2009.

[2] F. Zaidi, Ana Cavalli and Emmanuel Bayse, “NetworkProtocol Interoperability Testing based on Contextual Signatures”, The 24th Annual ACM Symposium on Applied Computing SAC'09Hawaii, USA, pp. 2-7, March 9-12 2009.

[3] A.V. Kolomeets “Algoritmy sinteza proveryayushchikh testov dlya upravlyayushchikh sistem na osnove rasshirennykh avtomatov”: dis. ... kand. tekhn. nauk. Tomskii gosudrastvennyi universitet, [PhD dissertation, Tomsk state university], 129 p., 2010.

[4] N. Kushik, M. Forostyanova, S. Prokopenko, N. Yevtushenko, “Studying the optimal height of the EFSM equivalent for testing elecommunication protocols”, Proc. of the Second Intl. Conf. on Advances In Computing, Communication and Information Technology – CCIT. 2014, pp. 159-163.

[5] Loymax: [official site]. URL: http://loymax.ru

[6] R. Dorofeeva, “Experimental evaluation of FSM-based testing methods”, In Proc. of the IEEE International Conference on Software Engineering and Formal Methods (SEFM05), Pp. 23-32, 2005.

[7] A. Gill, Introduction to the Theory of Finite-state Machines. M. Science, 272 p., 1966.

[8] A. Ermakov, N. Yevtushenko, “Increasing the fault coverage of tests derived against Extended Finite State Machines, System informatics”, № 7, pp. 23-32, 2016.

[9] K. El-Fakih, S. Prokopenko, N. Yevtushenko, G. Bochmann, “Fault Diagnosis in Extended Finite State Machines. Lecture Notes in Computer Science”, vol. 2644, Pp 197-210, 2003.

ACKNOWLEDGMENT

This work is supported by the grant for the basic research №16-49-03012 of Russian Scientific Fund.