UDC 519.7

A. Letichevsky, O. Letychevskyi, V. Peschanenko, A. Guba

DEDUCTIVE VERIFICATION OF REQUIREMENTS FOR EVENT-DRIVEN ARCHITECTURE

The current paper presents the technology of processing of requirements for systems with event-driven architecture. The technology consists of the stages of formalization, formal verification and conversion to design specifications. The formalization is the formal description of events as formal specifications called *basic protocols*. The consistency and completeness of basic protocols, safety properties and user-defined properties are verified. The deductive tools for dynamic and static checking are used for detection of properties violation. The method of enlargement allows reducing the complexity of proving and solving. Formal presentation of requirements allows converting them to SDL\UML specifications and generating the test suite. The technology is realized in IMS system and applied in more than 50 projects of telecommunication, networking, microprocessing and automotive systems.

Requirements capturing stage in software development process

Requirements capturing technology has become as a part of software development process not long ago. The advantages of such stage are the following:

– a detection on the earlier stages of a development of the deep hidden errors that could cause the re-planning or redesign. It could save the efforts of the test group and reduce the probability of financial losses in software projects;

– an automatic generation of a test suit from the formal presentation of requirements for future model;

– an automatic generation of a code or design specifications on the high levels of abstraction;

– near 60-70% of discrepancies, gaps and ambiguities in requirements are detected during the formalization stage.

Last years, special languages for requirements description have been developed. They are such as Promela [1] that allows to describe a system of an interacting automata for SPIN model checking tool [2], and User Requirement Notation (URN) [3] recommended by International Telecommunication Union (ITU) and for which the traversal mechanism of models on UCM language (a functional subset of URN language) has been described [4]. On the other hand, an interest to deductive methods for requirements verification has considerably increased. In this category, the most known tools are Hets [5], which uses common algebraic specification language (CASL) [6], STeP [7], PVS [8]. In 2002, ISO completed a standardization of Z-notation [9], which has been developed since 1974 and proved as a powerful and usable notation for specification [10] and verification of software systems [11, 12].

There are two stages in requirements capturing process:

– a formalization of requirements;

– a verification of formal requirements specifications.

Usually the requirements for system are presented as a set of documentation which contains the informal text with figures, tables, diagrams. It describes the behavior of reactive system as the set of reactions of system or as the interaction between its components. The first stage is the formalization of requirements where the formal specifications are created manually. These specifications are called *basic protocols* [13, 14]. The second stage includes the work with verification tools that accepts the formalized requirements as the input and generates verdict in which the set of findings is described. Every finding is accompanied by counterexample which leads to the violation.

Specific of our approach is the usage of deductive tools and symbolic modeling in verification process. It allows working with

the set of scenarios of a system behavior and with the set of values involved in formalization as opposed to traditional model checking techniques where preference is given to concrete values. With this purpose the deductive tools such as proving and solving machines for different theories are used in this technology. The main previous results of authors are described in [13 – 15].

Basic protocol language

We deal with the set of reactions of system presented as event-driven behavior. The reactive system consists of agents which could be considered as emitters and consumers. Every requirement presents some local description and performs some action. Every agent in system has attributes which define the agent type.

The *model* of a system is defined as a transition system which has the formulas of some typed logic language ***L*** as states. So this is a *symbolic* model. Some functional and predicate symbols of this language are interpreted over their type domains like an arithmetical operations and relations. Other symbols have types defined over fixed domains but are uninterpreted. They are called *attributes* and their values or properties could be changed during the system functioning.

The language ***L*** used in verification system is implemented so far [15] contains integer, real, enumerated, Boolean and symbolic types with linear arithmetic operations and inequalities for an arithmetic types, logical connectors for Booleans and equality for all types. The domain for symbolic type is the set of terms with distinguished set of constructors and access operations. Arrays are considered as functions with restricted domains for indices.

Every reaction of system could be presented by the following entities:

- trigger event;
- waiting state;
- changing of environment state;
- actions caused by trigger event.

It could be described by means of the basic protocol which contains three components – precondition, action and postcondition. Precondition is the formula in basic language ***L***. Basic protocol is applicable if this formula is true for given state of environment. Postcondition is the changing of attributes. It could consist of the formula in language ***L*** or the imperative statements like assignment. Action is the list of operations performing by agent. Basic protocol with its three components could be presented as MSC-diagrams [16].

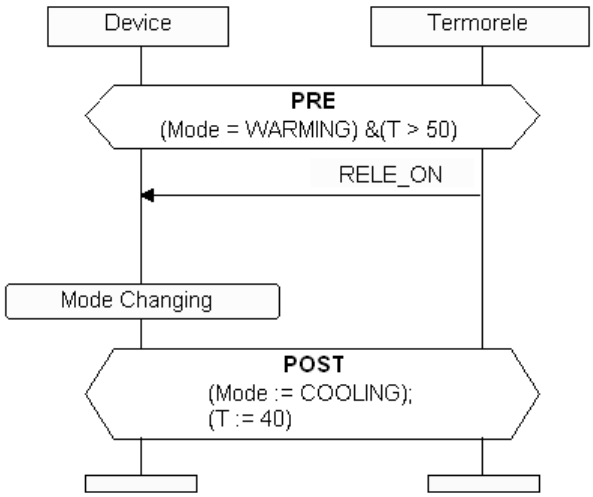

Fig. 1. Example of the basic protocol

The diagram presents the source requirement:

"Upon receiving of signal RELE_ON in warming mode when the temperature exceeds 50 degrees the device should change the mode into cooling and set the temperature in 40 degrees."

The set of actions performing by the agent "Device" is presented by MSC-statements like receiving the message and MSC-action which is titled as "Mode Changing". The formulas of precondition and postcondition define the changing of *symbolic state of the environment* presented by attributes *Mode* and *T*.

The process of formalization of requirements is definition of waiting states of system behavior where the set of triggers is awaited and the changing of environment states caused by trigger event. This approach is very convenient, because software designer typically specify system requirements as a set of possible behavior fragments expressing the system functionalities, and basic protocols resemble such natural language

requirements statementsused in engineering practice. The only difference is the using of the formal language instead of a natural language.

Symbolic modelling of requirements

If we will determine the initial state f_0 of system as some initial formula we can apply basic protocol for this formula by the following way.

– define the applicability of basic protocol or satisfiability of precondition and symbolic state of environment;

– compute the new formula f by means of special function called predicate transformer [15] that has current symbolic state of environment and postcondition as arguments.

Applying the basic protocols we could obtain the set of histories. Each history is presented by a sequence:

$$f_0 \to f_1 \to f_2 \to \dots$$

Every formula f_i is the symbolic state of system and process of generation of such histories presents symbolic modeling of requirements. In that way we can simulate the work of system obtaining different scenario of system behavior [13].

Symbolic modeling [17] of requirements is used for definition of reachability of some property in the system behavior. Property is reachable if we can reach such symbolic state of system which is consistent with this property that is the conjunction of the property and the state is satisfiable. We can check also the reachability of violation of user-defined property. For example, we can check whether some safety property S is violated. If during symbolic modeling we reach the state that is the formula f and $f\& \sim(S)$ is satisfiable then we've detect safety violation.

There are the following properties that could be verified:

– inconsistency. Formula of inconsistency could be defined statically and checking of satisfiability of this formula gives the possibility to detect the non-determinism in requirements;

– incompleteness. Static proving of incompleteness formula detects the possible candidates for deadlocks in the system. Launching of symbolic modeling tools defines the reachability of deadlock with counterexample given as MSC trace;

– safety. The safety violations also could be detected by proving with presentation of counterexample leading to this finding.

Verification system

The verification system was developed by authors where the symbolic modeling of formal requirements specifications and proving of mentioned above static properties was provided. Special deductive system has been developed for this purpose. It contains the proving machine and solver for integer and real arithmetic based on the algorithms of Fourier–Motzkin and Pressburger and proving machine with solver for enumerated and symbolic types. The input of system is the formalized requirements as a set of basic protocols. User could input the properties of safety which could be checked. Static checker prove the completeness, consistency and safety and if a violation has been detected then the system is trying to reach the violation by forward and backward symbolic modeling [15]. It also gives the counterexample as a scenario of system behavior in the case of reachability of the finding. There is a scheme of requirements processing technology below:

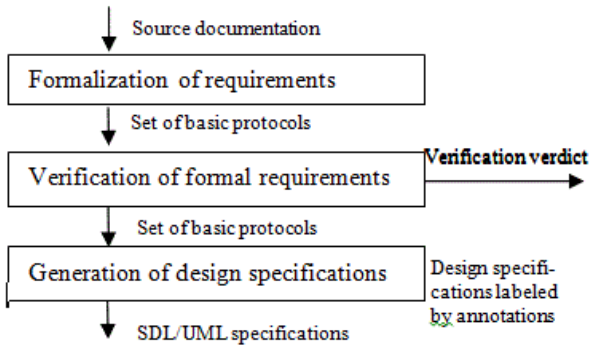

Fig. 2. Requirements processing technology

The input of the system is the source documentation which contains the requirement in a text form. There could be thousands of pages with figures, tables and other

non-formal descriptions. Verification engineer formalizes the information as the set of the system reactions. He defines the environment as the set of agents with its attributes. Then he tries to find descriptions of external triggers for system in documentation which causes the changing of environment and creates the basic protocols. He could use the incremental verification or create the whole set of basic protocols. After launching of verification system the verification verdict could be obtained and results are presented to customer. Customer should correct or refine the requirements and the verification stage will continue until absence of violations.

The other possibility of verification system is to generate the design specification like SDL/UML models. The result of generation could be used in further refinement and detailing of model.

Invariants techniques

Originally basic protocols are not ordered. Therefore after applying basic protocol and transforming current state by predicate transformer, any other protocol can be applied. So we should check the applicability for all basic protocols, and each check requires the use of deductive system. The symbolic technology allows reducing the expensive proving and solving processing during the modeling of a system behavior by means of computing invariant properties.

In real industrial projects there can be thousands of basic protocols, so to reduce the search time for the next applicable basic protocol is an actual problem. To make this search more efficient the succession relation $F \subseteq D^2$ is computed statically for the set of basic protocols. This relation by definition must satisfy the following property: basic protocol d' can be applied after d in some trace starting from the initial state of the system $S(L, D)$ only if $(d, d') \in F$.

The first approximation F_0 of succession relation is the following:

$$(d, d') \in F_0 \Leftrightarrow d(1) \wedge \exists x \alpha'(x) \neq 0$$

Here $\alpha'(x)$ is the precondition of d'. Let us prove that F_0 is a succession relation. Let there is a trace where d' can be applied after d. This trace must contain fragment $s \xrightarrow{a} s' \xrightarrow{a'} s''$ such that $s \xrightarrow{d} s' \xrightarrow{d'} s''$. From the monotonicity of predicate transformer it follows

$$d(s) = s' \Rightarrow d(1), s' \wedge \exists x \alpha'(x) \neq 0 \Rightarrow$$

$$\Rightarrow d(1) \wedge \exists x \alpha'(x) \neq 0.$$

Therefore $(d, d') \in F_0$. Succession relation F_0 can be strengthened using invariant properties of requirements. Formula φ is a preinvariant of a basic protocol d if it is valid each time before application of this protocol. Formula φ is a postinvariant of a basic protocol d if it is valid each time after application of protocol d. A formula $d(1)$ is a postinvariant of d. The succession relation can be strengthened by adding arbitrary invariants as conjunction members to $d(1)$.If we know some postinvariant of a protocol B then we can check the consistency of this invariant with the applicability conditions for all protocols and reduce the set of successors of B to only those protocols for which these conditions are consistent with postinvariant of B (conjunction is satisfiable). Therefore the network of basic protocols can be constructed and the reachability search is reduced from the search in an infinite tree to the search in a graph which is much more efficient.

A protocol B is called initial, if the condition of its applicability is consistent with the initial state. If preinvariant of a protocol B is 0, and it is not initial, then B is not reachable from the initial state.

The computation of the strongest invariants gives the possibility to define the strongest succession relation preliminary and avoid the expensive redundant proving and solving during simulation. In this case the reachability of a protocol coincides with the existence of a path from the initial protocol. The reachability of a property also can be computed without trace generating. For this purpose one must check the consistency of this property with the postcondition of all reachable protocols.

Unfortunately, it is possible that the strongest invariant does not exist (not expressible in the logic language). In this case we can be satisfied by computing finite approximations of invariants. This computation is performed solving corresponding equations for minimal fixed points.

The invariants that were computed during verification could be used on the step of design. For continuation of work with design specifications it is necessary to move to the lower level of abstraction and the refinement of specifications shall be provided on the level of design specifications.

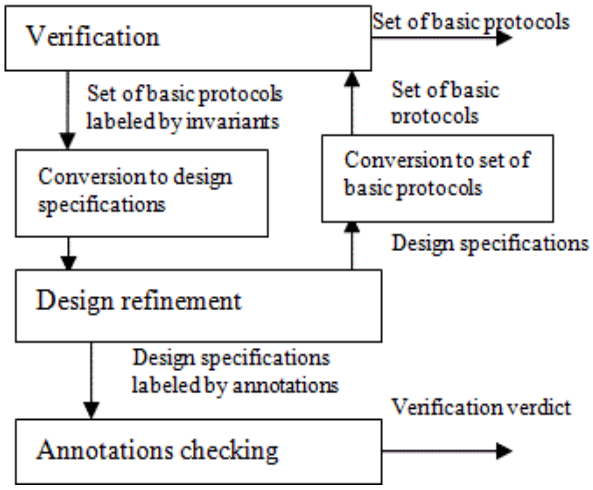

Fig. 3. Checking of integrity of requirements and design specifications

The design specifications could be labeled by invariants and it will give the possibility to check whether the refining specification correspond to source requirements. There are two options. These invariants that are called ***annotations*** could be checked during modeling of design specifications if such simulation tools exist. From the other side it could be prove by tool which is developing by authors – Insertion Modeling System [18], where annotations could be proved during symbolic modeling of design specifications. The other option is to decompose updated design specification into basic protocols and repeat verification. During this stage the invariants will be updated and the new set of annotations for design specifications could be formed.

Enlargement technique

The set of basic protocols could be encapsulated as enlarged basic protocol if it composes the connected component in an oriented graph which presents the succession relation of basic protocols. If this relation has been strengthened by invariants it is possible to define pre- and postcondition for enlarged basic protocol. If the folded connected component will be encapsulated to single node in graph then the set of input and output arrows could be defined for it. We can consider the set of preinvariants $(P_1, P_2, \ldots)$ for successors as the input arrows and correspondingly the set of postinvariant $(Q_1, Q_2, \ldots)$ for predecessors as output arrow. So the precondition for enlarged basic protocol could be defined as disjunction $P_1 \vee P_2 \vee \ldots$P1. The postcondition could be defined as disjunction $P_1 \wedge Q_1 \vee P_2 \wedge Q_2 \vee \ldots$P1.

If we hide the set of basic protocols into the enlargement basic protocol it is possible to reduce the significant interleaving and verify the properties for different subsets of basic protocols. There are the following possibilities:

– enlarging of an agent behavior. If we have some agents interacting one with another then we can encapsulate the behavior of a single agent into the enlarged basic protocol. It gives the possibility to detect the properties violations on the high level abstraction that reduces the complexity of computations;

– incremental verification. Some initial part of basic protocols could be verified separately. After verification of this part it could be inserted into the enlarged basic protocol. It gives the possibility to avoid exponential explosion in the number of projects;

– features interaction. After verification of feature it is possible to encapsulate it into the enlarged basic protocol and continue processing with the other features. It could give the possibility to avoid repetition of verification of common parts of system;

– enlarging of the set of agents. The group of agent could be folded into one entity that presents the enlarged agent. The interaction between agents could be reduced to the

interaction between enlarged agents and prove the absence of violation on such level of abstraction.

Applications

There are more than 50 industrial examples which were processed by verification system [19]. According to the rules of customers, the details of these projects couldn't be published. The examples of verification of several commonly known algorithms are presented below. These are telecommunication systems and protocols, telephony, automotive systems, networking, microprocessing and other projects. Process of verification started from formalization of requirements as a set of basic protocols. It is interesting that 70% of errors, discrepancies and gaps in requirements were detected during formalization. Verification engineer should not be as a specialist in subject domain which belongs to verified system. He considers the requirements as abstract statements which should be consistent. Such methodic could detect missed logic content and avoid the ambiguity in understanding of requirements by developers.

The more deep hidden errors have been detected after launching of verification system. The findings detected during verification are presented by counterexamples which show the scenario leading to violation.

Usage of enlargement techniques in verification of Hard Handoff feature in telecommunication protocol.

The requirements for the feature of telecommunication protocol present 1150 pages in event-driven style. Each requirement presents the consuming of messages by agent, processing of its parameters and sending of messages to other agents. There are 4 different types of agents that are considered in this protocol – Mobile Station, Base Station, Mobile Manager and Mobile Station Communicator**.** Actually the requirements were prepared for Mobile Manager component. Hard Handoff feature is considered as transition of mobile phone from one cell to another and all messages from phones and bases are processed by Mobile Manager. There is the number of features such as parsing of message, calculation of some parameters which could be implemented by programming with statements like nested cycles, non-linear functions. It is hard and unnatural to implement it by basic protocols. These features could be presented as folded entities which could be formalized as enlarged basic protocols and be refined on the next level of abstraction which is design specifications.

All these requirements were formalized as 245 basic protocols. 114 findings have been detected during formalization. After verification 36 findings have been detected as safety violations. There were 14 findings of incompleteness in systems presented by the deadlocks with corresponding counterexamples. 4 inconsistencies presented as non-determinism were detected.

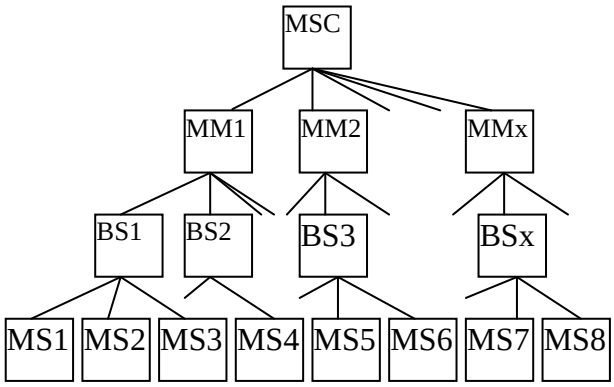

Fig. 4. The set of agents interacting in Hard Handoff feature of telecommunication protocol

But the total verification of feature without methods of enlarging entailed exponential explosion even with usage of only 3 Mobile Managers. The following enlargement of agents was used:

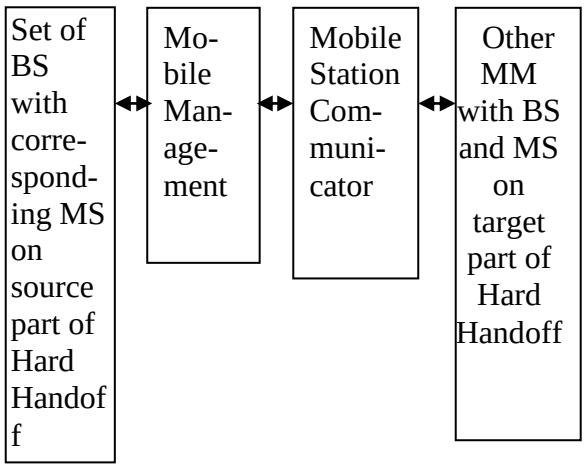

Fig. 5. Enlargement in Hard Handoff feature

Such formalization was verified during one hour and allowed to detect all findings for Mobile Manager requirements.

Feature interaction in Plain Old Telephone System (POTS)

The requirements for POTS presented as use-cases where every use-case presents the single feature. There are 10 different features which are intersected and the verification of all features together could entail exponential explosion. It was possible to encapsulate the already verified parts of system into enlarged entity and continue verification only with other parts of system. All features presents interaction of such agents as Switch, Service Point Control and Operation System. For example the feature "IN Freephone Billing" repeats part of the main feature "Basic Call". The intersected parts could be folded into enlarged entity and verification was performed only for parts of new feature.

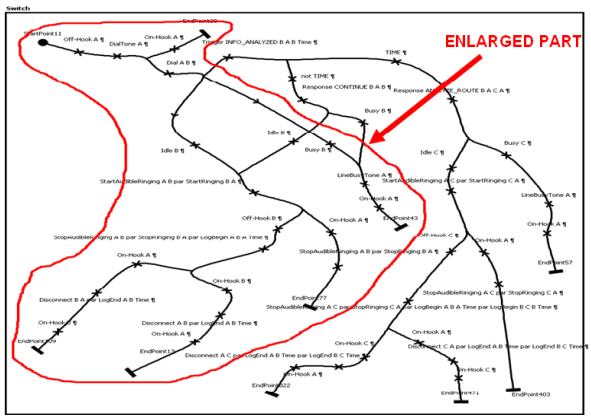

Fig. 6. Enlarging in POTS

UCM (Use Case Map) diagram presents the two features that were decompose to basic protocols and separated for two sets. One set contained the basic protocols which belongs to "Basic Call" feature and to the both ones. The second set belongs to the "IN Freephone Billing" feature.

The first set was presented as enlarged basic protocol. The verification was provided for enlarged basic protocol and the second set of basic protocols. The other features could be also verified by such way with those features with which the intersection exists.

Enlarging of agents in Generic Attribute Registration Protocol (GARP)

GARP protocol is a kind of multicast protocol where the processors are registered into groups in network. The network is presented by the set of domains which contain the certain number of processors. Every processor could create, join or leave the group. It should send the signal in the network by the corresponding path. The problem is to prove the absence of deadlocks in described algorithm for arbitrary topology of network and arbitrary number of agents. Every processor could be in some state and its transitions are defined by special state machine. The following enlargement was proposed for avoiding of exponential explosion.

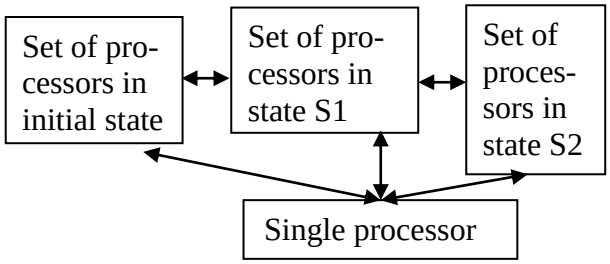

Fig. 7. Enlargement of agents in GARP

We consider every enlarged entity as the set of agents in some state. When any processor performed transition it becomes the member of the other enlarged entity. Such abstraction allowed proving the absence of deadlocks statically inasmuch as it is equivalent to formalization without enlarging and corresponding theorem is proved.

1. *Holzmann G.J.* The Spin Model Checker, Primer and Reference Manual. Addison-Wesley, 2003, 596 p.
2. Holzmann *G.* "The Model Checker SPIN" // IEEE Trans. on Software Engineering. – Vol. 23, N 5, – 1997. – P. 279–295.
3. *International* Telecommunications Union.Recommendation Z.151 – User Requirements Notation (URN). – 2008.
4. *Gunter Mussbacher.* Aspect-Oriented User Requirements Notation. Thesis submitted to the Faculty of Graduate and Postdoctoral Studies in partial fulfillment of the requirements for the degree of Ph.D. in Computer Science. University of Ottawa, Ottawa, Canada, September 2010.
5. *David Aspinall.* Proof general: A generic tool for proof development. In Susanne Graf and

Michael I. Schwartzbach, editors, TACAS, volume 1785 of Lecture Notes in Computer Science, pages 38–42. Springer, 2000.
6. *Autexier S., Hutter D., Mantel H., and Schairer A.* Towards an evolutionary formal software-development using Casl. In C. Choppy and D. Bert, editors, Recent Trends in Algebraic Development Techniques // 14th International Workshop, WADT'99, Bonas, France, Vol. 1827 of Lecture Notes in Computer Science. – Springer-Verlag, 2000. P. 73–88.
7. *Nikolaj Bjørner, Anca Browne, Eddie Chang, Michael Colón, Arjun Kapur, Zohar Manna, Henny B. Sipma and Tomás E.* Uribe. STeP: Deductive-Algorithmic Verification of Reactive and Real-time Systems.
8. *Sam Owre, John Rushby, N. Shankar and David Stringer-Calvert.* PVS: An Experience Report. Applied Formal Methods, FM-Trends 98, Boppard, Germany, October 1998.
9. *Information* Technology — Z Formal Specification Notation — Syntax, Type System and Semantics (ISO/IEC 13568:2002 ed.). 2002 – 07-01. P. 196.
10. *Spivey J.M.* The Z Notation: A Reference Manual. Prentice-Hall International, New Jersey, second edition, 1992.
11. *Brucker A.D., Rittinger F., and Wolff B.* HOL-Z 2.0: A proof environment for Z-specifications. Journal of Universal Computer Science. – Feb. 2003. – 9(2). – P. 152–172.
12. *Nipkow T., Paulson L.C., and Wenzel M.* Isabelle/HOL — A Proof Assistant for Higher-Order Logic, volume 2283 of Lecture Notes in Computer Science. Springer, 2002.
13. *Letichevsky A., Kapitonova J., Letichevsky A. jr., Volkov V., Baranov S., Kotlyarov V., Weigert T.* Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications // Computer Networks. – 2005. – Vol. 47. – P. 662–675.
14. *Letichevsky A., Kapitonova J., Volkov V., Letichevsky A. jr., Baranov S., Kotlyarov V., and Weigert T.* System Specification with Basic Protocols // Cybernetics and System Analyses. – 2005. – Vol. 4. – P. 3–21.
15. *Letichevsky A.A., Godlevsky A.B., Letichevsky A.A. Jr., Potienko S.V., Peschanenko V.S.* Properties of Predicate Transformer of VRS System // Cybernetics and System Analyses. – 2010. – Vol. 4. – p. 3–16.
16. *International* Telecommunications Union. Recommendation Z. 120 – Message Sequence Chart (MSC), 84 p.
17. *Symbolic* modeling, http://en.wikipedia.org/wiki/Model_checking
18. *Letichevsky A., Letychevskyi O., Peschanenko V.* Insertion Modeling System. In: Clarke E.M., Virbitskaite I., Voronkov A. (eds.) PSI 2011. LNCS, Vol. 7162, P. 262–274. *Springer*, Heidelberg (2011)
19. *APS&IMS* Systems, http://apsystem.org.ua

Data received 24.10.2012

About the authors:

Letichevsky Alexander Adolfovich,
Glushkov Institute of
cybernetics NAS Ukraine,
Head of dep. 100,
Kijv-187, Glushkov av., 50, 03680-CCP,

Letichevskyi Olexandr Olexandrovich,
Glushkov Institute
of cybernetics NAS Ukraine,
Mathematician of dep. 100,
Kijv-187, Glushkov av., 50, 03680-CCP,

Vladimir Peschanenko,
Kherson state university,
Associate professor of Department of Informatics of Kherson State University.
73000 Ukraine, Kherson,
40 Rokiv Zovtna street, 27,

Anton Guba,
Glushkov Institute of
cybernetics NAS Ukraine,
PhD-student of dep. 100,
Kijv-187, Glushkov av., 50, 03680-CCP.