Restrictions to time constructions of UCM formal model and their translation into Basic Protocols language

Pavel Drobintsev [1], Igor Nikiforov [1], Nikita Voinov [1], Vsevolod Kotlyarov [1], Alexandr Letichevsky[2]

[1] Saint-Petersburg State Polytechnic University, Russia,
e-mail: igor.nikiforovv@gmail.com
[2] Glushkov Institute of Cybernetic of NAS of Ukraine, Ukraine
e-mail:let@cyfra.net

***Abstract* —** The paper is devoted to approach of semantics adjustment for UCM (Use Case Maps) time and multithread constructions and their translation into Basic Protocols notation. Analysis of specification languages is presented. Based on the analysis UCM language is selected as the most perspective, powerful and user friendly language for development of software systems formal models.

A set of UCM constructions are described in the scope of the paper these are multithreading, delays and interruptions. UCM language is very powerful, but semantically incorrect models still can be created for described constructions. Authors propose a set of extensions and limitations which allow to solve problem of incorrect models creation.

Results of proposed limitation implementation in a set of projects are presented.

***Keywords* —** UCM semantics, timers, delays, behavioral tree, synchronization, threads generation.

I. Introduction

SOFTWARE system development starts with creation of requirements. Documents describing requirements specifications are generally written in natural language and may contain hundreds and thousands of requirements. Initial specifications often contain errors related to discrepant, incomplete and nondeterministic system behavior. Searching and fixing errors in requirements are more effective at early stages of the development [1].

It is almost impossible to manually analyze industrial systems specifications on errors presence without supporting toolset. Existing systems of verification and testing do not work with informal specifications. Thus the actual task is formalization of initial textual requirements using input languages of the tools for verification and testing.

One of the perspective integrated technologies of testing automation and symbolic verification based on formal models is VRS/TAT technology [2]. The technology uses UCM [3] notation for high level description of behavioral application models and tools for automation symbolic verification and test scenarios generation based on basic protocol language [4].

UCM specifications language is standardized, however contains a number of inaccuracies which do not allow displaying the modeled systems semantics unambiguously and correctly.

Proposed in this paper are restrictions on development of multithread models of the systems as well as adjustments of semantics of UCM language constructions for modeling time delays and interruptions.

II. Comparision of Specification Languages

Our comparative analysis of 9 widely used in the industry specification languages for software development considers the following 6 criteria:

- **IF** – Initial Formalization – can the language be used at the earliest stages of system's design development?
- **VI** – Visibility and Intuitiveness – does the language allow the user to easily visualize the system behavior at various levels of abstraction?
- **ESP** – Explicit Support of Parallelism – does the language contain semantic constructs which allow the user to explicitly express parallelism in the system behavior?
- **EST** – Explicit Support of Timing – does the language contain constructs for explicit expressing of timing constraints and dependencies?
- **TL** – Target Language – whether the language may be directly used as the input language for analysis, verification, and code generation by certain tool?
- **TS** – Tools Support – what tools and editors support the language?

The 9 languages considered for analysis were MSC [5], SDL [6], UML [7], Basic Protocols [8], Promela [9], Lotos [10], VDM-SL[11], RSL [12], UCM[3]. Summary results are presented in the Table 1.

Based on the above criteria, UCM was selected as the user-oriented and applicable at early stages of development among the considered alternatives. Using UCM, customers and

developers can achieve maximal mutual understanding of the architecture and design of the system pretty soon.

Although the language of Basic Protocols is enough powerful [13] for automated analysis and test scenarios generation, it is not as transparent as UCM and assumes a lot of low-level details, which make it difficult to use, especially for the customer expected to understand the system design in general.

- help a developer to predict complicated system behavior;
- provide convenient notation for depicting parallel structures, timers, interruption points on the diagram and aspects using.

Table 1. Comparative Analysis of Formal Languages for Specifications

Criteria\ Languages	**IF**	**VI**	**ESP**	**EST**	**TL**	**TS**
MSC	–/+	+	+	+	+	Telelogic TAU G2, TTCN, VRS, TAT
SDL	–/+	–/+	–/+	+	+	Telelogic SDL, TTCN
UML	–/+	–/+	–/+	+	+	Telelogic TAU G2
Basic Protocols	–/+	–/+	–/+	–/+	–/+	VRS/TAT
Promela		–/+	–/+	–/+	–/+	Spin
Lotos	-	-	+	–/+	–/+	LOTOS
VDM-SL	-	-	–/+	–/+	+	VDM, VDM Eclipse
RSL	-	-	–/+	–/+	+	RAISE
UCM	+	+	+	+	–/+	jUCMNav
Legend: "–" – not supported; "–/+"– partially supported; "+"– supported						

This becomes a showstopper for reconciling further work directions between the customers and developers which could lead to waste of efforts and increasing of software development cost.

In contrast to Basic Protocols, UCM notation is intuitively clear for a wide range of users – this simplifies negotiation between the customers and developers; however, its disadvantage is lack of a reliable tool to check correctness of a behavior model in UCM and to automatically derive test scenarios from it.

The above comparison suggests that combining both UCM and Basic Protocols within a single technological chain can achieve the maximal effect in creation and analysis of formal behavior models of complex systems.

III. Use Case Map

Use cases describe sequences of actions performed by a system in response to external impact from users or other software systems (components). Use cases reflect system functionality from system architecture description point of view. They introduce important components in software systems development process [14], namely:

- fill in the gap between textual requirements description and detailed system design;
- allow developing system architecture on high level of abstraction as well as specifying system behavior when architecture is already defined;

System design in UCM language is presented as a set of diagrams interacting between each other. Each diagram in turn focuses on the description of the components (agents, system processes), objects, observers and subsystems interaction. Each component and subsystem contains elements of responsibility (Responsibilities) corresponding to some events in the system as well as strictly defined sequence of their occurrence.

Using elements of UCM notation not only linear behavior can be specified but also parallel scenarios (AndFork) with their further synchronization (AndJoin) can be described. FailurePoint element participates in the description of the interruptions generation and processing mechanism. Timer element is used to specify the system timer behavior both for cases with simple time delay and for cases with complicated logical behavior.

Also is worth noting a structuring element (Stub) which allows creating hierarchical system representation and performing the software development by components from the highest level of abstraction to detailed description of low level diagrams.

Thus the aggregation of components and diagrams provides visible representation of the system behavior and system's components interaction to the user.

UCM diagram is developed using UCM Navigator [15] graphical editor. Fig.1 shows a fragment of UCM diagram for real telecommunication project where high level behavior of the agent modeling automatic telephone station is described.

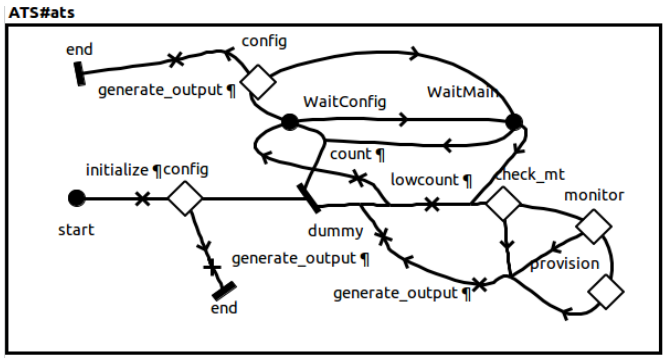

Figure1. UCM diagram with automatic telephone station module behavior.

IV. Restrictions of Multithread Systems Development

Working with the acting UCM standard Z.151 [3] we found that it contains a number of inaccuracies hampering the modeling of multithread systems. In the following subsections we discuss them in details and provide examples.

Brackets balance in parallel threads specification

Consider a case when syntactically correct elements of threads generation and synchronization can cause a violation of parallel threads structure and consequently an incorrect system behavior.

In Fig.2 after *AndFork_A* and *AndFork_E* elements generation of threads B, E and F, G respectively is performed, while on *AndJoin_C* and *AndJoin_D* elements synchronization of threads B, F and C,G respectively is performed.

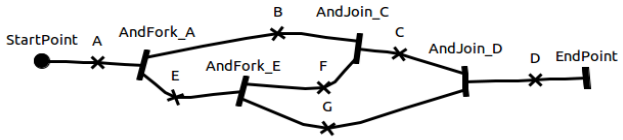

Figure 2. Violation of parallel threads structure.

It is easy to notice that synchronization of threads generated by different elements significantly complicates the mechanism of error detection and fixing in the system, as well as complicates the tracking of parent/child connection in the threads hierarchy. Such connections are useful when child thread keeps executing after parent thread has finished.

System behavioral graph with correct structure of threads generation and synchronization is depicted in Fig.3.

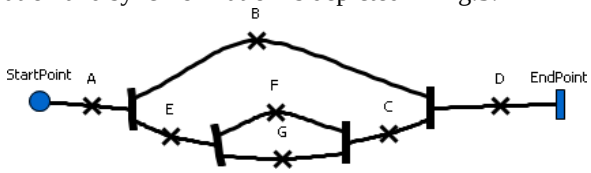

Figure 3. The graph with correct threads structure.

Threads structure analysis can be compared with the analysis of mathematical expressions brackets format. If expression brackets format is violated, it is considered to be syntactically incorrect. This is also valid for parallel threads modeling: if threads structure is violated, the whole system is considered to be syntactically incorrect.

Analyzing threads for errors detection and fixing allows creating syntactically correct system models.

Unlimited generation of threads

Consider the case shown in Fig.4. Threads B and E are generated after D element. Thread B is finished on EndPoint element. Thread E is returned through the cycle, which has no condition of iterations limits, and D element and generates new threads B' and E'. The behavioral scenario is repeated for thread E'.

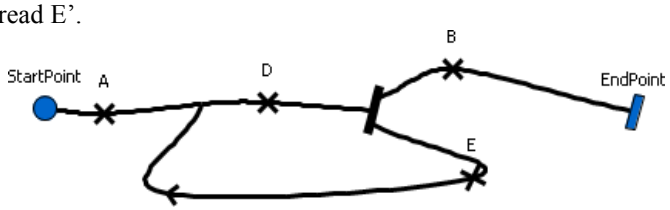

Figure 4. Unlimited generation of threads

Unlimited cycles lead to generation of unlimited number of unfinished threads which leads to shortage of memory and other resources. Thus it is important to introduce restrictions on usage of such constructions in developed models.

Data racing while accessing shared resources by parallel processes

Consider the case when shared resources are used on the parallel branches without synchronization. Fig.5 depicts two parallel threads using "**var**" shared resource without synchronization. Such formalization leads to racing while data accessing [15].

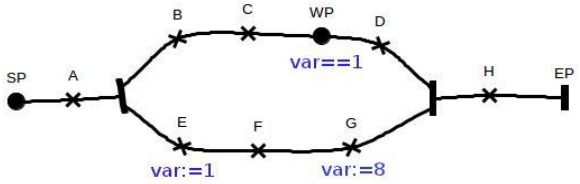

Figure 5. Shared resources without synchronization.

There are two executable scenarios in the model:

1) If "**E**"-"**F**"-"**G**"-"**WP**" scenario is executed, "**D**" element will never be applied. This scenario leads to a deadlock.
2) If "**E**"-"**F**"-"**WP**"-"**G**" scenario is executed, "**D**" element can be applied and this scenario will reach EP end point.

Deadlock can be avoided by introducing synchronization and thus excluding parallel access to shared system resource (Fig.6).

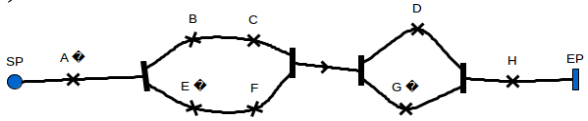

Figure 6. System model with synchronization.

Proposed restrictions on multithread systems development in UCM

- Using parallel constructions with violated threads structure is not allowed.
- Using unlimited recursive threads generation is not allowed.
- Using shared resources on parallel execution paths without synchronization is not allowed.

V. Mathematical Restrictions Model

To automatically check UCM model for restrictions on the parallel threads specified in section IV we propose to use formal specification of a thread as well as mathematical specifications of incorrect cases which are implemented in the algorithm of automatic analyzing of UCM model.

To analyze UCM diagram we shall introduce a formal specification of a thread.

Statement 1. Let thread T be a quaternion

$$T = (S, P, C, E),$$

where S is the element on a diagram which initiates the thread; P is a set of all threads which are parents of this thread; C is a set of all threads which are child of this thread; E is a set of elements on a diagram which are included into this thread.

Statement 2. Let "parents" of the thread be the threads which have led to creation of this thread. The thread may not have parents in case this is the start (main) thread.

Statement 3. Let "children" of the thread be the threads which are initiated of this thread. The thread does not have children in case this is the end thread which leads to EndPoint.

Each restriction can be specified as mathematical formula. If this formula is true for all system threads, the UCM diagram is considered to be incorrect.

Introduced specification of a thread as well as following specifications are used for mathematical specifications of restrictions:

$T_i \in T_a$ — the i-th thread, contained in all T_a threads of UCM model, where $i \in 1:N$;

$T_i^{S_j}$ — the i-th thread, created on the element S_j, where $S_j \in S$;

$S_i \xrightarrow{n} S_j^{next}$ — UCM element S_j^{next}, which is reachable from element S_i in n steps;

$T_i \to C_j$ — the thread T_i creates the child thread C_j;

$T_i \perp T_j$ - the thread T_i is synchronized with the thread T_j;

$R \in T_i$ - the thread T_i uses the resource R.

Introduce mathematical specifications for incorrect conditions.

Unlimited recursive creation of threads is specified by the following formula

$$(\forall T_i^{S_j} \in T_a) \to (C_k^{S_m} \in T_a) \& (S_j \xrightarrow{N \neq 0} S_m) \& (S_i = S_m),$$

which is translated into natural language as follows:

- let T_i be a thread from the set of all threads T_a and created on the element S_j; if true that T_i creates child thread C_k on the element S_k given that S_k is reachable in N steps from S_j and elements S_k and S_j are the same element of the diagram, then such diagram is considered to be incorrect..

Usage of parallel constructions with violation of threads structure is specified by the following formula:

$$(\forall T_i^{S_j} \in T_a) \perp (C_k^{S_m} \in T_a) \& (T_i \to C_k),$$

which is translated into natural language as follows:

- let T_i be a thread from the set of all threads T_a and created on the element S_j; if true that T_i is synchronized with the thread C_k created on the element S_k given that C_k is a child thread for T_i, then such diagram is considered to be incorrect.

Mathematical specification of shared resources usage without synchronization on parallel execution paths is the following:

$$(T_i \in T_a) \| (T_j \in T_a) \& (R \in T_i) \& (R \in T_j),$$

which is translated into natural language as follows:

- if two parallel threads T_i and T_j from the set of all threads T_a modify shared resource R, then such diagram is considered to be incorrect.

Implemented is the library for analyze of parallel threads on UCM diagram based on the algorithm of search and specification of incorrect situations. The scheme of the algorithm is presented on Fig.7.

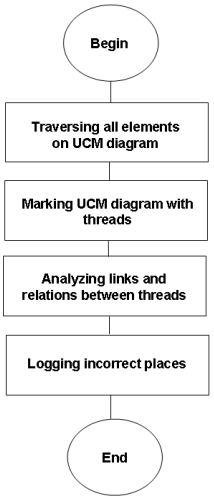

Figure 7. Algorithm of UCM model analyze

The algorithm contains 4 steps:

— traversing all elements on UCM diagram. On this step all elements on UCM diagram are traversed and lists of global start points, threads creation points and synchronization points are created. Global start point is StartPoint element which is not connected with start points of Stub elements (i.e. it is not contained in paths hierarchy);

— marking UCM diagram with threads. UCM diagram is traversed from each global start point to next elements until EndPoint is reached. While traversing the parts of the paths are marked with threads. Initially the algorithm marks the path with the thread created on start point. If thread creating element or threads synchronization were met during traversal, a new thread is added into threads array and path marking proceeds with new thread;

— analyze of links and relations between threads. The analyzing and selection of places on UCM diagram where corresponding specifications of incorrect situations are performed based on formulas specifying incorrect situations and paths array generated on this step;

— logging the information about potentially dangerous places detected on the step.

The research proved that proposed specifications of restricted constructions and the library for analyzing implementing the algorithm of automatic detection of incorrect UCM constructions allow to find potentially dangerous places and errors in UCM model.

VI. Features of Time Delays Modeling

Requirements of time delays often occur in the industrial systems. In this case it is about the modeling of the relative time – the time between events. Events are the change in the system attributes values.

Features of timer usage

According to the standard [3] two outgoing paths are connected with Timer element (Fig.8): regular path (RP) and timeout path (TOP). For selecting each path there are conditions CRP and CTOP respectively. Also there is a trigger path (or trigger counter) which affects timer behavior and allows to cancel the delay.

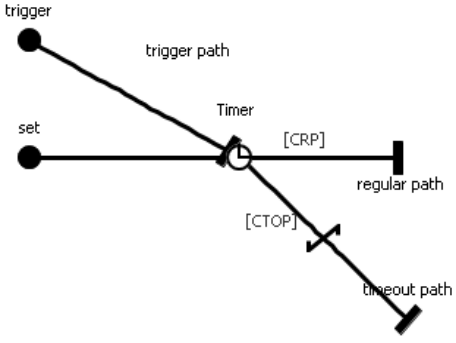

Figure 8. UCM diagram with Timer element

In Z.151 standard semantics of the elements modeling time delays contains cases description of model possible behaviors depending on the occurred events, but does not describe which types of events are associated with timers and does not specify types of some events specific for telecommunication applications specification.

Extend timer semantics description

Extend UCM timer semantics description with the following events:

- Timer set: TIMER_SET <timer name>. The event occurs when Timer element is reached.
- Timer expiration: TIMER_EXPIRE <timer name>. The event occurs after CTOP condition has been executed.
- Timer reset: TIMER_RESET <timer name>. The event occurs after RP or TOP path has started execution or at trigger event occurrence.

Using semantics of Timer element and associated events three types of time delays can be stated:

1) simple delay, whose modeling feature is strictly specified conditions of outgoing paths (false) and absence of trigger event;
2) interruption delay, whose modeling feature is presence of trigger event;
3) interrupted execution delay, whose modeling feature is presence of FailurePoint interruption on timeout path.

Proposed extension of UCM timer semantics by timer set, expiration and reset allowed solving the delay description problem for telecommunication projects.

VII. Features of Interruptions Modeling

There are two types of interruptions in requirements: local interruptions, affecting behavior of a specific function or object, and interruptions, affecting behavior of other system threads. Each interruption shall have a corresponding handler.

Interruptions are modeled by the group of elements [3]: FailurePoint, AbortStartPoint and FailureStartPoint. Fig.9 depicts a simple UCM diagram modeling an interruption with the handler.

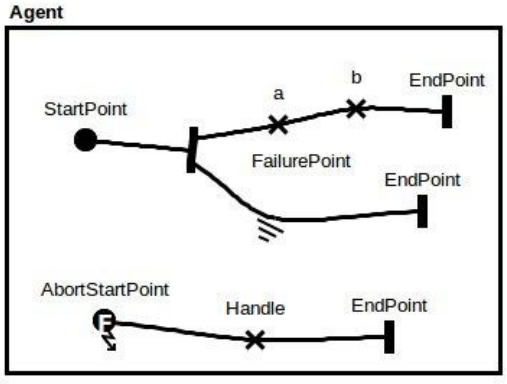

Figure 9. UCM diagram modeling an interruption with handler

When FailurePoint (grounding symbol) is reached, interruption occurrence condition is calculated. Call it *FailureCondition*. If calculation result is true, then the execution flow with FailurePoint will be interrupted and a flag of interruption occurrence (call it *FailureFlag*) will be enabled.

As soon as *FailureFlag* equals **true**, conditions calculation on all interruptions handlers is performed: FailureStartPoint and AbortStartPoint. Handler types behavior is described in the standard [3].

The main difference between two types of handlers is the impact on parallel threads, affected by the interruption. In case of FailureStartPoint only interruption of the thread which reached FailurePoint is performed. In case of AbortStartPoint interruption of all threads which belong to FailurePoint activity area is performed. Call this set AbortScope.

Usage of any of the considered elements introduces the enormous number of system behaviors, which need to be checked during verification. In general case checking of all possible execution variants is impossible due to states explosion problem.

For all elements from the AbortScope set it is required to check interruption occurrence which means to perform interleaving of all cases where interruption can occur.

Proposed are three approaches which can be used either separately or supplement each other:

1. Checking behaviors on the bounds of linear parts of paths and in the points of common resources sharing. For this purpose the analysis of the paths and elements set from FailurePoint activity area is performed as well as key points where behavior shall be checked are specified. Combination of all possible behaviors is performed for these points only.
2. User check. User marks his check points on the diagram with a marker.
3. Default check, i.e. verification will be performed for all elements of the set. This case can be only used after manual introduction of restrictions on the set of verified elements [17], otherwise states explosion is inevitable.

Worth noticing, that the first two approaches are enough to balance the time of verification and required coverage level. In general case usage of proposed approaches allowed to effectively solving the problem of interruption description in telecommunication projects.

VIII. Conversion of time delays into basic protocols

For VRS/TAT toolset for verification and testing a tool for translation of models in UCM language into models in basic protocols language was developed [18,19,20]. UCM→BP translator implements the conception of time delays and interruption conversion as well as checking of formulated restrictions on multithread systems development.

Consider the features of some constructions translation important for specification of real-time applications.

For Timer element there is *timer_var* attribute, which is responsible for timer state and assigned two possible values: **true** — if the timer is set, **false** – if the timer is reset. By default the value is **false**.

In basic protocol for Timer element responsible for timer set *timer_var:=true* expression is generated in postcondition while TIMER_SET expression is generated in the process field of basic protocol.

For each outgoing path (RP and TOP) from Timer element a single basic protocol is generated.

Precondition of the basic protocol for RP path (expression for selection of regular path) is generated in accordance with logic formula derived based on [13]:

(timer_var=true)&(CRP)∨
(timer_var=true)&(trigger)&(CTOP) ***(1)***,

where trigger is a logic expression for trigger event.

TIMER_RESET action is generated in the process field of basic protocol.

In postcondition of this basic protocol the expression modeling timer reset is generated: *timer_var:=false.*

Consider a basic protocol for timeout path (TOP). In general case the following expression is generated in precondition:

(timer_var=true)&(~CRP)&[(CTOP∨
(~trigger)&(~CTOP)] ***(2)***

TIMER_EXPIRE and TIMER_RESET operations are generated in the process field of the basic protocol for TOP path.

Thus, conversion of time delays implies generation of three basic protocols with different logical expressions in precondition.

For each considered case of timer modeling optimization of logical expressions is possible as the values of used conjuncts and disjuncts is known beforehand.

IX. Conversion of interruptions in UCM language into Basic Protocols

Two basic protocols are generated in basic protocols notation for elements modeling interruptions (FailurePoint). The first protocol is for regular execution path with negation of interruption occurrence *~(FailureCondition)* in precondition.

The second protocol contains checking of interruption occurrence *FailureCondition* in precondition and a flag in postcondition signaling that the interruption has occurred – *FailureFlag:=true*.

For all handlers of interruptions: FailureStartPoint and AbortStartPoint a new execution flow will be created, the first protocol for each of them will contain checking of interruption occurrence in the system in precondition as well as expression for this handler enabling, call it *HandlingCondition*.

(FailureFlag_1 = true)∨ (FailureFlag_2 = true∨...∨ (FailureFlag_n = true) & (HandlingCondition) ***(3)***

For all protocols generated for elements of AbortScope set *~(FailureFlag=true)* expression is added to precondition which means that flow execution will continue until exception will occur.

Using approaches to translation of time delays and

interruptions a conversion principle of time delay with FailurePoint element can be described. For this case generated are a basic protocol for timer set, two basic protocols for timer exit and two protocols for FailurePoint element.

Herewith the condition of interruption event occurrence is added to protocols which lead to FailurePoint.

X. Results

Manual creation of basic protocols for multithread UCM

When translator is used for formalization, basic protocols describing the structure and behavior of the system are generated automatically.

Implemented rules to obtain fields of basic protocols allow avoiding syntax and semantics errors while generating basic protocols.

Automatically generated fields are the following:

— basic protocols names;
— key agents;
— instances description;
— agents states in pre- and postconditions;
— control variables required to save control flow and synchronization;

models containing time delays and interruptions is very time-consuming and laborious process which demands accuracy and deep experience in specific language of basic protocols.

Average time on a single basic protocol creation is 20 minutes but this is an optimistic estimation for those cases when there are no uncertainties in requirements while creating a formal models.

— process fields;
— signals;
— local variables declaration fields;
— local variables initialization fields;
— comments.

UCM2FM tool was applied in number of experimental projects. Table 2 contains time characteristics of formal model creation from UCM specifications using manual and automatic formalization methods in three projects.

Table 2. Manual and automatic approach comparing.

Project	BP number	Manual BP model development time (staff day)	Manual UCM formalization and BP model generation time (staff day)	Gain in means of time
SMTP	30	1,2	0,5	2,4
CDMA	205	10	3	3,3
Satellite terminal	392	20	9	2,2

Results obtained with VRS/TAT technology usage after integration of the tool for automated translation of UCM model into basic protocols model show that the time has decreased in 2,5 times in comparing with manual creation of basic protocols. Fig.10 contains values of spent time reduction in miscellaneous projects achieved by usage of innovative technology for basic protocols automated generation.

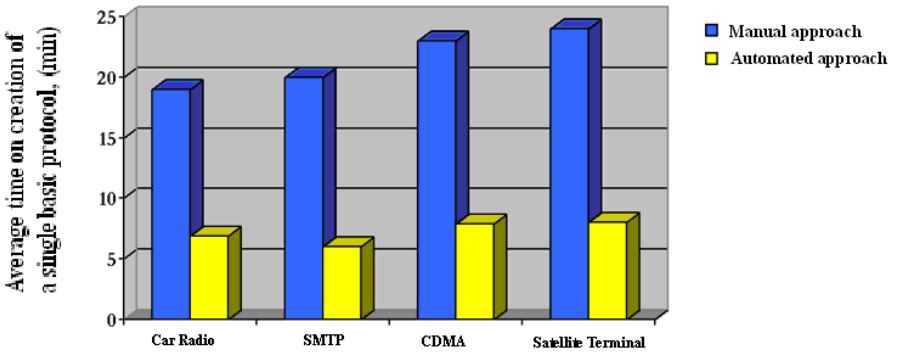

Figure 10. Time saving on creation of a single basic protocol

XI. Usage example

Using of proposed UCM semantics adjustments in the project for telecommunication project presented on Fig.11 (the project was obfuscated due to business requirements) allowed translating of the set of UCM behavioral diagrams into 392 basic protocols, performing of model verification, generating about 11 000 test scenarios and testing which reduced the efforts on 26% in comparison with traditional approach of manual testing.

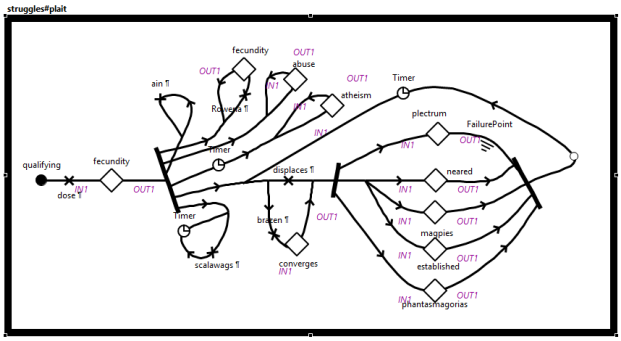

Figure 11. UCM diagram for telecommunication project

XII. Conclusion

Considered in this work methods of semantics adjustments of UCM standard elements, modeling time delays and interruptions as well as restrictions on multithread systems development allow modeling of complex telecommunication systems reducing possibility to create semantically incorrect model behaviors.

Methods are implemented in UCM→BP translator which allows using of VRS/TAT technological chain more convenient and effective for projects of middle and high complexity.

Proposed translator together with supporting toolset of VRS/TAT technology was applied in modules development of telecommunications applications and has shown a significant reduction of efforts on quality industrial software project development.

References

[1] Booch Gr., Maksimchuk R., Engel M., Young B., Conallen J., Houston K. Object-Oriented Analysis and Design with Applications. Addison-Wesley Professional; 3rd edition, 2007. 720 p.

[2] Veselov A.O, Kotlyarov V.P. Test automation in telecommunication area // "Scientific and technical sheets of SpbSTU". №4(103). Spb.: SpbSTU publishing, -2010. - pp. 180-185.

[3] Recommendation ITU-T Z.151. User requirements notation (URN), 11/2008.

[4] Letichevsky A.A., Kapitonova Yu.V., Letichevsky A.A. (Jr) and others. Systems specification using basic protocols // Cybernetics and system analysis, 2005, №4. pp.3-21.

[5] ITU Recommendation Z.120. Message Sequence Charts (MSC), 11/99.

[6] M. Berger, J. Soler, L. Brewka, H. Yu, M. Tsagkaropoulos, Y. Leclerc, C. Olma Methodology and Toolset for Model Verification, Hardware/Software co-simulation, Performance Optimisation and Customisable Source-code generation WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATIONS E-ISSN: 2224-3402 Issue 6, Volume 10, June 2013, pp 169-178.

[7] Diem P. G., Hien N. V., Khanh N. P. An Object-Oriented Analysis and Design Model to Implement Controllers for Quadrotor UAVs by Specializing MDA's Features with Hybrid Automata and Real-Time UML // WSEAS TRANSACTIONS on SYSTEMS Issue 10, Volume 12, October 2013, pp.483-496

[8] 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, ISSRE 2004, WITUL (Workshop on Integrated reliability with Telecommunications and UML Languages), Rennes, 4 November. 2005. P. 30-38.

[9] Ka L. Man Formal Verification of SystemC Specifications Using SPIN Proceedings of the 5th WSEAS Int. Conf. on Microelectronics, Nanoelectronics, Optoelectronics, Prague, Czech Republic, March 12-14, 2006 (pp.80-85).

[10] Salaun, G., Serwe, W.,Translating hardware process algebras into standard process algebras - illustration with CHP and LOTOS , Proceedings of the International Conference on Integrated Formal Methods, Eindhoven, The Netherlands, 2005.

[11] VDM // http://www.vienna.cc/e/evdm.htm .

[12] Milne R. The Proof Theory for the RAISE specification language. RAISE Report REM/12, STC Technology Ltd, 1990.

[13] Yusupov Y. V. Integrated methodic of automated creation of C-applications formal behavioral models from the source code. A thesis for the title of the degree of technical sciences candidate. SPb.: SPbSTU, 2009. 176 p.

[14] Buhr R. J. A., Casselman R. S., "Use Case Maps for Object-Oriented Systems." Prentice Hall, 1995.

[15] UCM Navigator - http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/WebHome

[16] Gergel V.P. High-performance calculations for multiprocessor multicore systems. Nizhny Novgorod: University of Nizhny Novgorod, 2010. – 544 p.

[17] P.Drobintsev, V.Kotlyarov, I.Chernorutsky. Test automation based on user scenarios coverage. "Scientific and technical sheets", St.Petersburg university, vol.4(152)-2012, pp.123-126.

[18] Nikiforov I.V., Petrov A.V., Yusupov Yu.V. Generating formal model of the system based on requirements specified in USE CASE MAP notation // "Scientific and technical sheets of SpbSTU". №4(103). Spb.: SpbSTU publishing, -2010. - pp. 191-195.

[19] I.Anureev, S.Baranov, D.Beloglazov, E.Bodin, P.Drobintsev, A.Kolchin, V. Kotlyarov, A. Letichevsky, A. Letichevsky Jr., V.Nepomniaschy, I.Nikiforov, S. Potienko, L.Pryima, B.Tyutin. Tools for supporting integrated technology of analysis and verification of specifications for telecommunication applications // SPIIRAN works- 2013-№1-28P.

[20] I.Nikiforov, A.Petrov, V.Kotlyarov. Static method of test scenarios adjustment generated from guides // "Scientific and technical sheets", SpbSTU, vol.4(152)-2012, pp. 114-119.

Vsevolod Kotlyarov - was born in Stavropol region of Russia on the 14 July 1944. Hold a master degree with specialty «Mathematical and computing instruments and devices» of Saint-Petersburg State Polytechnic University (SPbSPU) in 1968. Defended PhD thesis with specialty "Software engineering" in 1972. Main areas of interests - «Software engineering», «Technologies and tools of automated verification and testing».

Since 1972 he is working as associated professor in SPbSPU, since 1995 as senior researcher in St.Peterburg software development department of Motorola, since 2008 as full time professor of SPbSPU. He is scientific adviser of 20 PHD dissertations of post-graduate students. His scientific school of "Software Engineering" was included in the list of top schools of St.Petersburg.

Prof. Kotlyarov became a M of IEEE and ACM in 1993, M of SABA (Science Advisory Board Association) of Motorola Company in 2005. He is a member of the program committees of the following conferences: Microsoft Technology in Software theory and practice, SYRCOSE, Workshops of Ershov informatics conference (PSI).