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 Leitchevsky 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. COMPARISON 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.

<table>
<thead>
<tr>
<th>Language</th>
<th>IF</th>
<th>VI</th>
<th>ESP</th>
<th>EST</th>
<th>TL</th>
<th>TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic Protocols</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Promela</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lotos</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDM-SL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UCM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Criteria\Languages</th>
<th>IF</th>
<th>VI</th>
<th>ESP</th>
<th>EST</th>
<th>TL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSC</td>
<td>–/+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>SDL</td>
<td>–/+</td>
<td>–/+</td>
<td>–/+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>UML</td>
<td>–/+</td>
<td>–/+</td>
<td>–/+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Basic Protocols</td>
<td>–/+</td>
<td>–/+</td>
<td>–/+</td>
<td>–/+</td>
<td>–/+</td>
</tr>
<tr>
<td>Promela</td>
<td>–/+</td>
<td>–/+</td>
<td>–/+</td>
<td>–/+</td>
<td>–/+</td>
</tr>
<tr>
<td>Lotos</td>
<td>–</td>
<td>–</td>
<td>+</td>
<td>–/+</td>
<td>–/+</td>
</tr>
<tr>
<td>VDM-SL</td>
<td>–</td>
<td>–</td>
<td>–/+</td>
<td>–/+</td>
<td>+</td>
</tr>
<tr>
<td>RSL</td>
<td>–</td>
<td>–</td>
<td>–/+</td>
<td>–/+</td>
<td>+</td>
</tr>
<tr>
<td>UCM</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>–/+</td>
</tr>
</tbody>
</table>

Telelogic TAU G2, TTCN, VRS, TAT
Telelogic SDL, TTCN
Telelogic TAU G2
VRS/TAT
Spin
LOTOS
VDM, VDM Eclipse
RAISE
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.
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.

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'.

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].

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).
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 may 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.

Introduce mathematical specifications for incorrect situations. The scheme of the algorithm of search and specifications of incorrect situations is specified by the following formula:

\[ \forall T^s_i \in T^a \rightarrow (C_j^s \in T^a) \land ((S_j \rightarrow \text{next} S_j^\text{next}) \land (S_j = S_i)) \]

which is translated into natural language as follows:

- let \( T^s_i \) be a thread from the set of all threads \( T^a \) and created on the element \( S_j \); if true that \( T^s_i \) creates child thread \( C_j \) on the element \( S_j^\text{next} \), then such diagram is considered to be incorrect.

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

\[ \forall T^s_i \in T^a \perp (C_j^s \in T^a) \land (T^s_j \rightarrow C_j) \]

which is translated into natural language as follows:

- let \( T^s_i \) be a thread from the set of all threads \( T^a \) and created on the element \( S_j \); if true that \( T^s_i \) is synchronized with \( C_j \) created on the element \( S_j^\text{next} \), then such diagram is considered to be incorrect.

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

\[ (T^s_i \in T^a) \parallel (T^s_j \in T^a) \land (R \in T^s_i) \land (R \in T^s_j) \]

which is translated into natural language as follows:

- if two parallel threads \( T^s_i \) and \( T^s_j \) 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.
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.

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

As soon as \textit{FailureFlag} equals \textit{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.

\section{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\textemdash 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 \textit{timer\_var} attribute, which is responsible for timer state and assigned two possible values: \textit{true} — if the timer is set, \textit{false} — if the timer is reset. By default the value is \textit{false}.

In basic protocol for Timer element responsible for timer set \textit{timer\_var=}true expression is generated in postcondition while \textit{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]:

\[
\text{timer\_var=}true \lor \text{timer\_var=}true \land \text{trigger} \land \text{CTOP}
\]

\[
\text{timer\_var=}true \land \text{trigger} \lor \text{timer\_var=}true \land \text{CTOP}
\]

where \text{trigger} is a logic expression for trigger event. \text{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: \textit{timer\_var=}false.

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

\[
\text{timer\_var=}true \land \neg \text{CRP} \lor \text{CTOP} \lor \text{timer\_var=}true \land \neg \text{CRP}
\]

\[
\text{timer\_var=}true \land \neg \text{trigger} \lor \neg \text{CTOP}
\]

\[
\text{TIMER\_EXPRESS} \land \text{TIMER\_RESET}
\]

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.

\section{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 \(\neg \text{FailureCondition}\) in precondition.

The second protocol contains checking of interruption occurrence \textit{FailureCondition} in precondition and a flag in postcondition signaling that the interruption has occurred – \textit{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 \textit{HandlingCondition}.

\[
\text{FailureFlag}_1 = \text{true} \lor \text{FailureFlag}_2 = \text{true} \lor \ldots \lor \text{FailureFlag}_n = \text{true} \lor \text{HandlingCondition}
\]

For all protocols generated for elements of AbortScope set \(\neg \text{FailureFlag} = \text{true}\) 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;
modes 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.

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

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.

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.
REFERENCES


[10] Salaun, G., Serwe, W., Translating hardware process algebras into standard process algebras - illustration with CHP and LOTOS.


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 (SPbSU) 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.Petersburg 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).