UDC 004.415.5

TRAVERSAL OF ARBITRARY SEQUENCES OF UCM SYMBOLIC TRANSITIONS FOR TEST GENERATION

A.A. Guba

Glushkov Institute of Cybernetics of NAS of Ukraine,
40 Glushkova ave., Kyiv, Ukraine, 03680.
Tel.: (044) 526 0058,
E-mail: antonguba@ukr.net

The paper proposes a method of traversal of high-level multi-threaded models formalized in UCM language. The pros and cons of this approach comparing to the existing ones are considered. The method that allows state space exploring of UCM models using symbolic solving and proving is presented. The generation of symbolic traces used for software system testing is described.

Запропоновано метод обходу високорівневих багатопотокових моделей, які формалізовані мовою UCM. Розглядаються переваги та недоліки даного підходу у порівнянні з існуючими. Представлений метод дозволяє обходити простір станів моделей UCM, використовуючи символьне розв'язання та доведення. Описана генерація символьних трас, які використовуються для тестування програмних систем.

Introduction

Use Case Map (UCM) is a scenario based language that has associated behavior of the system with its architecture in the formal and visual forms. It has been successfully used in describing real-time systems, with a particular focus on telecommunication system and services. UCM is a part of an approved proposal to ITU-T for a User Requirements Notation (URN) [1]. It can be applied to capture and integrate functional requirements in terms of causal scenarios representing behavioral aspects at a higher level of abstraction, and to provide the reasoning about the system architecture and behavior. UCM is not intended to replace UML, but rather complement it and help to bridge the modeling gap between requirements (use cases) and design (system components and behavior).

UCM semantics began to develop in the early 1990s . The first attempt to construct the semantics of UCM was the introduction of a traversal mechanism for UCM [2, 3], which allows you to select a specific scenario in the language of UCM model and convert it to MSC [4]. Later, UCM semantics was expressed by LOTOS language formalisms [5], abstract state machines (ASM) [6], as well as timed transition systems (CTS) and timed automata (TA) [7]. Recent works on the semantics of UCM based on the modeling of business processes [8] and use the theory of Petri nets [9]. In [10, 11] an insertional model was used as the formalism for describing the semantics of UCM. The traversal of UCM use deductive symbolic simulation tools together with the modeling of the behavior of a system using model checking.

This paper is the continuation of [10, 11] and enhances the usage of sequences of UCM language constructs, as well as proposes an approach for trace generation of high-level multi-threaded UCM with arbitrary UCM constructs connection types. Some methods of implementation of UCM semantics are compared.

UCM specification

UCM specification (UCM project) is a set of UCM maps, which consist of a finite, possibly empty set of UCM paths. Maps are the basic notion of the language and express behavior scenario. Maps are represented as graph structures, which vertices are associated with language constructs. Paths define the order of the vertices. Vertices can be constructs of different types used to express start and end of paths, alternative and parallel branches, merging of paths. Path is located on different maps, which could be linked by vertices of stub type [11]. Vertices can be inside the component. The components are abstract entities that are used to express the structure of the model, and could be nested into another component. UCM specifications are suitable to describe multithreaded systems. Multithreaded system is a system of interacting parallel processes in which interaction takes place via shared memory. An example of such system could be programs for multicore processors. Multithreading in UCM is implemented by parallel paths and parallel branches. In this paper, the following UCM constructs are considered:

- StartPoint (beginning of a path),
- EndPoint (end of a path),
- EmptyPoint (empty construct),
- DirectionArrow (indication of a path direction),
- Responsibility (action, some operator),
- OrFork (alternative branching),
- OrJoin (joining of alternatives without synchronization),
- AndFork (parallel branching),
- AndJoin (joining of parallel alternatives with synchronization),
- WaitingPlace (waiting condition),
- Stub (link from one UCM map to another).

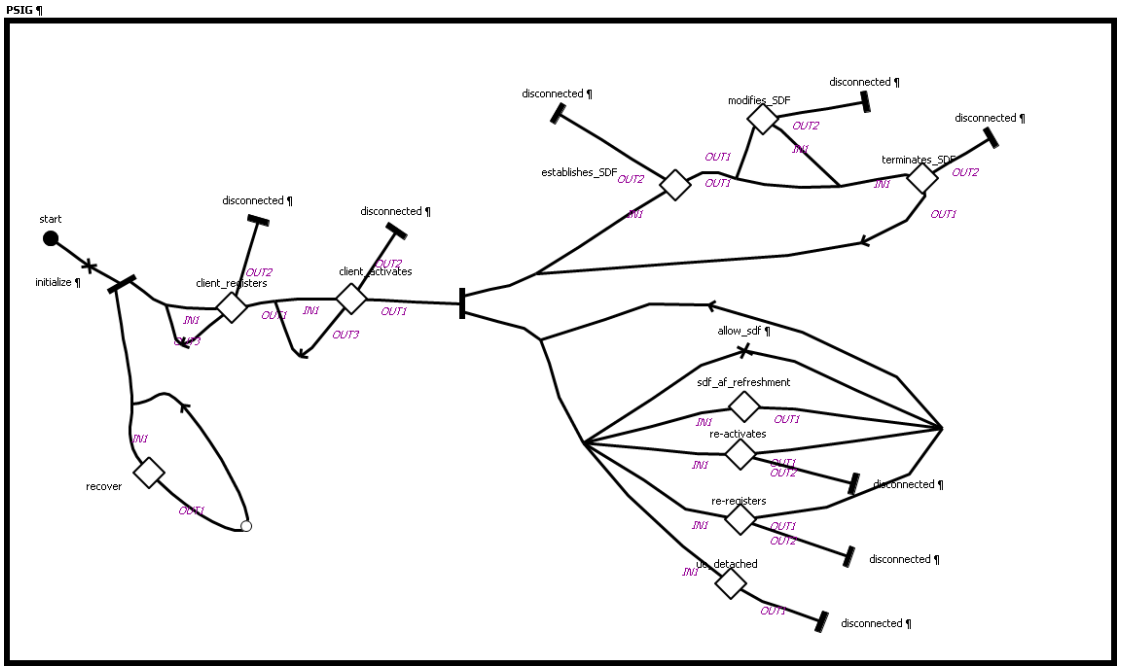

Fig.1. Example of UCM map

The meaning of UCM project is a transition system represented as a composition of multilevel UCM environment and agents inserted into this environment at different levels. The structure of the UCM environment is defined by the environment description. Agents correspond to components. Environment is specified by attributes definition. There are global and local attributes. Besides attributes classification with respect to level of localization, it is considered their division into two disjoint classes of attributes: user-defined (data flow) and control (control flow). User-defined attributes are introduced by UCM developer and are used in Responsibility and WaitingPlace. Control attributes are formed at the stage of model construction from UCM project – translation of the project to the system of environment and agents in order to provide synchronization of processes corresponding to the semantics defined in [10, 11], and could be efficiently realized in C++ code as it was shown in [12].

Basic protocols language

The language, which is used for describing formulas with user-defined attributes (conditions, assignments, etc.) is called Basic Protocols Language. A basic protocol is a formula of dynamic logic $\forall x(\alpha(x,r) \rightarrow < P(x,r) > \beta(x,r))$ and describes the local properties of a system in terms of pre- and postconditions α and β. Both are formulae of first order multisorted logic interpreted on a data domain, P is a process, represented by means of an MSC diagram, and describes the reaction of a system triggered by the precondition, x is a set of typed data variables, and r is a set of environment and agent attributes. The general theory of basic protocols is presented in [13].

At transitions in the space of formulas, a postcondition is considered as an operator. As the operator transforms one formula to another, in [14] a term "predicate transformer" was used. Thus, to compute transitions between the states of such models, basic protocols are interpreted as predicate transformers: for a given symbolic state of a system and a given basic protocol, the direct predicate transformer generates the next symbolic state as its strongest postcondition and the backward predicate transformer generates the previous symbolic state as its weakest precondition. These concepts have been implemented in VRS (Verification of Requirement Specifications) and IMS (Insertion Modeling System) systems [15]. The following data types in the models are supported:

- simple data types: Boolean, integer, real, enumerated, and symbolic,
- structured data types: uninterpreted functional symbols, arrays, and lists of simple date types.

Symbolic type is represented by expression in algebra of free terms. Enumerated type is a user-defined one contains symbolic constants (atomic expressions in algebra of free terms). Parameters and returned value of uninterpreted functional symbols are simple data types only. Let denote that if we are talking about functional, then we mean uninterpreted functional symbol. Arrays are considered as uninterpreted functional symbols with bound restrictions for values of parameters.

Let E(r) be an environment state. Application of a basic protocol to an environment state looks like the following formula:

$$(\exists(x,r)(E(r) \wedge \alpha(x,r)) \neq \varnothing) \Rightarrow \exists x\, pt(E(r) \wedge \alpha(x,r), \beta(x,r)),$$

where pt is a function of predicate transformer. In general, a postcondition of basic protocol can be considered as following conjunction:

$$\beta(x,r) = A(x,r) \wedge L(x,r) \wedge C(x,r),$$

where $A(x,r)$ is a conjunction of assignments, $L(x,r)$ is a conjunction of list operators *add_to_tail, add_to_head,* $C(x,r)$ is a formula part of postcondition.

Automatic test generation

UCM language makes development of projects user-convenient. It is easy to add new maps and paths to existing project. Such method of incremental system design provides rich opportunities for developers, but at the same time, it may lead to errors in the early stages of system development. Detection of errors in early stages of the project development greatly reduces the number of product corrections and total cost of software development. Therefore, methods of testing and model-based testing tools have been actively developed.

This paper is devoted to the enhancement of automatic test technology for VRS/TAT [16] toolkit that supports automatic generation of test scenarios for requirements specifications specified in the notation of basic protocols. In [17] an approach of automatic translation of UCM to Basic Protocols was proposed. This approach proves to be a good engineering solution, as it does not require a lot of time of development and ensures the correct behavior of the model obtained after translation. The cons of this approach are generation of additional basic protocols, which cause:

- additional calls of solver due to checks of satisfiability of pre-condition of generated protocol and environment state (performance issue),
- additional calls of predicate transformer due to applying post-condition of generated protocol (performance issue),
- additional stored states after application of generated protocols for visited and cycle detection (memory issue)
- increase the explosion of states number if generated protocols are occurred on parallel branches (performance issue)

Therefore, the traversal of UCM maps without generation of additional protocols becomes an actual task. From the standpoint of system behavior analyzing the notion of a path is important for testing and verification. Path *p* is a sequence of protocols corresponds to UCM constructs (*UC*): $p = u_1 \to u_2 \to \ldots \to u_n$, where $u_i \in UC$, u_1 – StartPoint, u_n – EndPoint, $u_i \to u_j$ indicates that protocol corresponds to u_j construct is applied after protocol corresponds to u_i construct.

Let us show that traversal of UCM maps and generation of valid UCM paths for system testing is possible without generation of additional protocols. Let us divide all UCM constructs (*UC* set) into three disjoint sets *PC* (protocol constructs), *AC* (auxiliary constructs), *IC* (intermediate constructs).

Set *PC* shall contain UCM constructs, which have basic protocol in attachment: Responsibility and WaitingPlace. Also, we consider WaitingPlace as 3 different constructs: WaitingPlace without trigger, WaitingPlace with trigger of transient type, WaitingPlace with trigger of persistent type, because they all have different behavior. These all protocols are user-defined.

Set *AC* shall contain StartPoint, EndPoint, which indicate start and end of path. StartPoint shall be used to set initial model behavior for UCM traversal. EndPoint shall be used to determine the correct end of model behavior.

Set *IC* shall contain OrFork, AndFork, AndJoin, Stub, EmptyPoint connected with WaitingPlace with trigger of transient type, EmptyPoint connected with WaitingPlace with trigger of persistent type. Stub shall be considered as a set of in/out paths. Each in-path is bound with connected StartPoint. Each out-path is bound with connected EndPoint. Thus, set *IC* shall contain connected StartPoint and connected EndPoint instead of Stub.

DirectionArrow, OrJoin, not connected EmptyPoint shall be eliminated, because they do not affect behavior of the system.

Also, let us introduce special intermediate *Fake* constructs (*FC* set) with bound empty basic protocol. Fake construct shall be used to transmit an environment state. It does not have any behavior semantics and just a technical temporary construct, which shall be eliminated together with empty protocols at the end of algorithm.

Let us introduce following types of UCM transitions:

1. $\overset{s}{\to} u_i$ – first transition (initialization), indicates that protocol u_i is applied after StartPoint, $u_i \in PC \cup FC$,
2. $u_i \overset{e}{\to}$ – last transition (leads to path end), indicates that EndPoint is occurred after application of protocol u_i, $u_i \in PC \cup FC$,
3. $u_i \to u_j$ – transition indicates that protocol u_j is applied after protocol u_i, $u_i, u_j \in PC \cup FC$,
4. $u_i \overset{c}{\to} u_j$ – transition indicates that protocol u_j is applied after protocol u_i with the influence of semantics of UCM construct *c*, $u_i, u_j \in PC \cup FC, c \in IC$.

5. $u_i \overset{c^k}{\rightarrow} u_j$ – transition indicates that protocol u_j is applied after protocol u_i with the influence of semantics of sequence of UCM constructs c, $u_i, u_j \in PC \cup FC, c \in IC$, k –number of UCM constructs in the sequence (sequence depth). If k=0, then this is a transition of the type 3. If k=1, then this is a transition of the type 4.

Definition. Two UCM paths p_1, p_2 are called equivalent ($p_1 \equiv p_2$) if they pass through the same non-empty protocol constructs and have the same environment states after application of these protocols.

Lemma. Paths $p_1 = pc_1 \rightarrow pc_2$ and $p_2 = pc_1 \rightarrow fc \rightarrow pc_2$ are equivalent, where $pc_1, pc_2 \in PC$, $fc \in FC$.

Proving.

Paths consist from transitions. Path p_1 consists from one transition and p_2 consists from two transitions. Let us write each transition in the terms of transition rule of transition system. P_1 looks like:

$$\frac{E \xrightarrow{apply(pc_1)} E', a \xrightarrow{apply(pc_1)} a'}{E[a] \xrightarrow{apply(pc_1)} E'[a']},$$

where E is an environment state before application of pc_1, E' – environment state after application of pc_1 and pc_2 is applicable for E', $E[a]$ is an insertion function, which describes interaction between environment and agent a inserted into this environment. In other words, insertion function calls application of basic protocols for given agent a.

P_2 transitions look like:

$$\frac{E \xrightarrow{apply(pc_1)} E', a \xrightarrow{apply(pc_1)} a'}{E[a] \xrightarrow{apply(pc_1)} E'[a']}, \quad \frac{E' \longrightarrow E'', a' \longrightarrow a''}{E[a'] \longrightarrow E'[a'']}.$$

Since *fake* construct has an empty bound protocol, it does not affect an environment state and $E'' \Leftrightarrow E'$. So, by definition, $p_1 \equiv p_2$. Lemma is proved.

Theorem. Let $p_1 = u_1 \rightarrow u_2 \rightarrow \ldots \rightarrow u_n$, where $u_i \in PC \cup IC, i = \overline{2..n-1}$, u_1 – StartPoint, u_n – EndPoint. Let us denote *PC* constructs contained in p_1, as PC_{p_1} and *IC* constructs contained in p_1, as IC_{p_1}.

Let $p_2 = \overset{s}{\rightarrow} pc_1 \overset{c_1^k}{\rightarrow} \ldots \overset{c_p^k}{\rightarrow} pc_n \overset{e}{\rightarrow}$. Let us denote $pc_1, \ldots, pc_n \in PC$ constructs contained in p_2, as PC_{p_2} and $c_1, \ldots c_p \in IC$ contained in p_2 transitions, as IC_{p_2}.

If $PC_{p_1} = PC_{p_2}$ and $IC_{p_1} = IC_{p_2}$, then $p_1 \equiv p_2$.

Proving.

At first, let us split all transitions of the type 5 with depth $k \geq 2$ into k transitions of the type 4. $pc_i \overset{c^k}{\rightarrow} pc_j \Leftrightarrow pc_i \overset{c}{\rightarrow} fc \overset{c}{\rightarrow} \ldots \overset{c}{\rightarrow} fc \overset{c}{\rightarrow} pc_j$, where $fc \in FC$. As a fake construct does not have any behavior semantics this transformation is clear and does not require an additional proof. Also, shifts of initialization from generated protocol to the first one and finalization to the last one are obvious.

So, a path is considered as a sequence of transitions in the form of deuce $pc_i \rightarrow pc_j$ or triple $pc_i \overset{c}{\rightarrow} pc_j$: $p_2 = \overset{s}{\rightarrow} pc_1 \rightarrow pc_2 \rightarrow \ldots \rightarrow pc_i \overset{c}{\rightarrow} \ldots \overset{c}{\rightarrow} pc_n \overset{e}{\rightarrow}$. We shall prove that each possible transition in the form of triple of a path without generated protocols p_2 is an equivalent to transition of path with generated protocols p_1. Total number of all triples is 112. It is impossible to show proof for all of them in this paper. Let us show the equivalence for one of them.

Let us have a map that is linked via Stubs two times. So we have two different exits from the map. In [17] a copy of $Responsibility_1$ is generated and two transitions are obtained:

$$Responsibility_1 \xrightarrow{\text{ConnectedEndPoint}(exit_1)} Responsibility_2,$$
$$Responsibility_1 \xrightarrow{\text{ConnectedEndPoint}(exit_2)} Responsibility_3$$

Without generation of protocols it could be presented in the form of rewriting rule with deterministic choice operator "+" [15] and action *check* that verifies a stub binding:

$$Responsibility_1 = check(exit_1).Responsibility_2 + check(exit_2).Responsibility_3.$$

This is an equivalent transformation by rewriting rule definition. The correctness of other transformations can be proved similarly. Further, using the lemma described above, the theorem is proved.

Conclusions

Method of traversal of high-level multi-threaded models formalized in UCM language was proposed. The proposed method uses symbolic solving and proving and generates symbolic traces used for software system testing. It was implemented in VRS/TAT technological chain that makes it more convenient and efficient on projects of medium and high complexity.

1. *International Telecommunications Union. Recommendation.* Z.151 – User Requirements Notation (URN), 208 p. – 2008.
2. *Kealey J. and Amyot D.* Enhanced Use Case Map Traversal Semantics. 13th SDL Forum (SDL'07), Paris, France, Springer. – September 2007. – P. 133–149; 288 p.
3. *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. – 334 p.
4. *International* Telecommunications. Union. Recommendation Z.120 – Message Sequence Charts (MSC), 1998. – 136 p.
5. *Guan R.* From Requirements to Scenarios through Specifications: A Translation Procedure from Use Case Maps to LOTOS. M.Sc. thesis, OCICS, University of Ottawa, Canada, 2002. – 146 p.
6. *Hassine J., Rilling J., and Dssouli R.* An ASM Operational Semantics for Use Case Maps // 13th IEEE International Requirement Engineering Conference (RE'05), IEEE Computer Society Press, September 2005. – P. 467–468; 494 p.
7. *Jameleddine Hassine.* Formal semantics and verification of use case maps, A thesis in The Department of Computer Science and Software Engineering, Concordia University, Montreal, Quebec, Canada, 2008. – 299 p.
8. *Weiss M. and Amyot D.* Business Process Modeling with URN. International Journal of E-Business Research, July-September 2005. – Vol. 1(3). – P. 63–90; 112 p.
9. *Peter Huber, Kurt Jensen, and Robert M. Shapiro*. Hierarchies in coloured petri nets // In Proceedings of the 10th International Conference on Applications and Theory of Petri Nets, Springer-Verlag, London, UK – 1991. – P. 313–341; 724 p.
10. *Губа А.А., Шушпанов К.И.* Инсерционная семантика плоских многопотоковых моделей языка UCM // Управляющие системы и машины. – 2012. – № 6. – С. 15–21.
11. *Годлевский А.Б.* Инсерционная семантика параллельных процедурных конструктов языка UCM // Там же, 2012. – № 6. – С. 22–34.
12. *Vladimir Peschanenko, Anton Guba, Constantin Shushpanov*: (en) Specializations and Symbolic Modeling. In: Ermolayev, V. et al. (eds.) Proc. 9-th Int. Conf. ICTERI, Kherson, Ukraine, Jun 19–22, 2013. – P. 490–505.
13. *Letichevsky A., Kapitonova J., Volkov V., Letichevsky Jr.A., Baranov S., Kotlyarov V., Weigert T.* System Specification with Basic Protocols. Cybernetics and System Analysis. – 2005. – (4), P. 3–21.
14. *Letichevsky A.A., Godlevsky A.B., Letichevsky Jr. A.A., Potienko S.V. Peschanenko V.S.* Properties of Predicate Transformer of VRS System. Cybernetics and System Analyses. – 2010. – (4), P. 3–16.
15. *Letichevsky A., Letychevskyi O., Peschanenko V.* Insertion Modeling System. In: Clarke, E.M., Virbitskaite, I., Voronkov, A. (eds.) PSI 2011. LNCS 7162, Springer Verlag, Berlin Heidelberg, 2011. – P. 262–274.
16. *Baranov S.N., Drobintsev P.D., Kotlyarov V.P., Letichevsky A.A.* Implementation of an integrated verification and testing technology in telecommunication project. Proceedings // IEEE Russia Northwest Section. 110 Anniversary of Radio Invention conference. S.Petyersburg, 2005. – 11 p.
17. *Никифоров И.В., Петров А.В., Юсупов Ю.В.* Генерация формальной модели системы по требованиям, заданным в нотации USE CASE MAPS. Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. СПб.: Изд-во Политехнического ун-та, 2010. – № 4 (103). – С. 191–195.