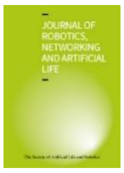
Journal of Robotics, Networking and Artificial Life
Vol. 11(2); December (2025), pp. 175-179
ISSN (Online): 2352-6386; ISSN (Print): 2405-9021
https://alife-robotics.org/jrnal.html



Research Article

# Automated Random Simulation Technique and Its Prototype Tool for Checking Abstract Collaborative Behavior of Multiple Systems Based on EPNAT

Sho Matsumoto[1], Tetsuro Katayama[2], Tomohiko Takagi[3*]
[1]*Graduate School of Science for Creative Emergence, Kagawa University, 2217-20 Hayashi-cho, Takamatsu-shi, Kagawa 761-0396, Japan*
[2]*Department of Computer Science and Systems Engineering, Faculty of Engineering, University of Miyazaki, 1-1 Gakuen-kibanadai nishi, Miyazaki-shi, Miyazaki 889-2192, Japan*
[3]*Department of Engineering and Design, Faculty of Engineering and Design, Kagawa University, 2217-20 Hayashi-cho, Takamatsu-shi, Kagawa 761-0396, Japan*
*Email: takagi@eng.kagawa-u.ac.jp*
[*]Corresponding Author

ARTICLE INFO

*Article History*

Received 25 November 2022
Accepted 31 October 2025

*Keywords*

System design
System quality
Place/transition net
Vienna development method

ABSTRACT

The collaborative behavior of multiple systems provides valuable functions and services to users. However, it is actualized by large and complex implementations, which frequently include serious failures. In this study, we propose an automated random simulation (ARS) technique for checking the abstract collaborative behavior of multiple systems at the design level. The abstract collaborative behavior is expected to be designed using an extended place/transition net with attributed tokens (EPNAT), and the checking is performed dynamically based on the design called "EPNAT model". The ARS technique consists of (1) an algorithm for model execution using random search with the evaluation of constraints including feasibility, and (2) a stopping criterion for model execution focusing on glue transitions. The ARS technique requires tool support; therefore, we developed a prototype tool. We ran the prototype tool with a trial model and three faulty models, and found its effectiveness and future challenges.

## 1. Introduction

Systems are playing an important role as the foundation of modern information society, and a wide range of people uses systems in their daily lives. Many of recent systems are connected by networks and exhibit collaborative behavior to provide valuable functions and services to users. However, the collaborative behavior is actualized by large and complex implementations, which frequently include serious failures that cause considerable damage to society and users. System engineers (SEs) require systematic quality control techniques to effectively detect such failures at the design level, because it is difficult for SEs to detect and remove them at the implementation stage.

To address this issue, this study presents an automated random simulation (ARS) technique for checking the abstract collaborative behavior of multiple systems at the design level. The abstract collaborative behavior is expected to be designed by SEs using an extended place/transition net with attributed tokens (EPNAT), and the checking is performed dynamically based on the design. The ARS technique requires tool support; therefore, we developed a prototype tool. This study is an extension of our previous one [1].

EPNAT [2] is a formal modeling language for concurrent and distributed state-transition systems. It was developed from the place/transition net (PN), and has the following additional elements:

- Attributes (variables given to each token to characterize the states of systems)
- Actions (procedures for data processing that accompany each transition of systems)
- Basic constraints (local conditions that each variable and transition must satisfy, which are classified into invariants, pre- and post-conditions)

EPNAT is intended for use in projects involving Vienna development method (VDM) [3] and/or model-driven development (MDD). This enables SEs to create executable designs for abstract behavior of multiple systems before the implementation stage. In this study, the EPNAT's executable designs will be referred to as "EPNAT models" or "models". The term "abstract" indicates essential parts of system requirements that are explicitly defined without ambiguousness.

Some studies are closely related to the ARS technique. Takagi et al. [4] proposed a semiautomated simulation (SS) technique that visually checks the abstract behavior of systems based on EPNAT. The SS technique is intended to check arbitrary aspects of designs from the individual perspectives of SEs; thus, it does not concentrate on collaborative behavior. In addition, SEs must manually select inputs for model execution and evaluate non-basic constraints. In contrast, the ARS technique concentrates on collaborative behavior while automating input selection and non-basic constraint evaluation. Another previous study [5] described a stopping criterion for PN-based software testing to cover transition paths of a specific length, which was evaluated using coverability trees generated from given PN models. However, it cannot deal with the additional elements discussed above and cannot concentrate on collaborative behavior. Therefore, we also discuss a novel stopping criterion in this study.

The remainder of this paper is organized as follows: In Section 2, we propose the ARS technique, which consists of an algorithm and a stopping criterion for model execution. Non-basic constraints are closely related to the algorithm; thus, they are discussed as part of the algorithm. Section 3 shows our prototype tool and experimental results, followed by a discussion on the effectiveness of the ARS technique. In Section 4, we present our conclusions and discuss future challenges.

## 2. ARS Technique

The proposed ARS technique consists of the following algorithm and stopping criterion for model execution:

### *2.1. Algorithm for Model Execution*

The algorithm requires four inputs: (a) a model to be checked, (b) a set of non-basic constraints for the model, (c) a pseudo-initial state of the model, and (d) a set of pseudo-final states of the model. Inputs (a) and (b) are described by SEs on the basis of the system requirements, which specify interactions among multiple systems that actualize collaborative behavior. The non-basic constraints are classified into types A and B on the basis of the timing of failure detection. The constraints of type A must be satisfied by systems at all times; thus, their dissatisfaction (violation) with the fire of a transition results in a definite failure. A typical example of a type A constraint is that when a condition is satisfied in a system, another condition must be always satisfied in another system. However, the constraints of type B must be satisfied by systems sometime in the future. When a type B constraint has never been satisfied during model execution, it indicates a possible failure at the end of the model execution. The term "possible" indicates that the existence of the failure cannot be strictly proven but is strongly inferred. A typical example of a type B constraint is that when a condition $c_1$ is satisfied, another condition $c_2$ must be satisfied sometime in the future. Some kind of behavior of different systems can be done between the satisfaction of $c_1$ and the satisfaction of $c_2$. Inputs (c) and (d) are determined by the SEs on the basis of their objective of checking. In this study, each model state is identified using a marking and variable values. The initial and final states based on the system requirements are explicitly defined by the SEs as part of the model. SEs typically specify the initial state as input (c), but they can also specify another valid state excluding the final states and input (d). In addition, SEs can specify not only the final states but also additional valid states as input (d). Note that if there are any states with no feasible (fireable) transitions, they should be defined as final states and included in input (d).

The algorithm has two outputs: (e) failures of the model and (f) a process of model execution. If the stopping criterion described in Section 2.2 is satisfied without the detection of any failures, output (e) is not provided. Output (f) consists of executed paths and metrics related to the stopping criterion. It is provided not only as text but also as a graphical representation and will assist SEs in understanding and removing failures.

The procedure of the algorithm is described as follows. Note that the description of output (f) has been omitted for simplicity.

[Step 1] Initialize the input model to the input pseudo-initial state.

[Step 2] Create a set of candidates for feasible transitions on the basis of only the current marking. Hereafter, the set is referred to as *T*. Note that the feasibility of each transition is exactly determined not only by the current marking but also by the pre-conditions, which are evaluated in a later step.

[Step 3] If *T* is empty, output it as a failure and terminate the algorithm. Otherwise, select a transition randomly from *T*. Hereafter, the selected transition is referred to as *t*.

[Step 4] Evaluate the pre-condition of *t*. If *t* has any arguments, select their values randomly from their respective sets of valid values that make their respective invariants and the pre-condition true. If there are any arguments for which no valid values can be found or if the evaluation results in false, remove *t* from *T* and return to Step 3.

[Step 5] Fire *t* with the selected values of the arguments, and execute the action of *t*. All related constraints, such as the invariants of variables used in the action, the post-condition of *t*, and the input non-basic constraints, are evaluated to detect failures. If there are any constraints (except the non-basic constraints of type B) whose evaluation results are false, then output them as definite failures and terminate the procedure. Otherwise, the current state of the input model is successfully changed. Note that the evaluation results of the non-basic constraints of type B are not determined, even if they have not been satisfied at this instance.

[Step 6] Evaluate the stopping criterion. If satisfied, proceed to Step 8.

[Step 7] If the current state is one of the input pseudo-final states, return to Step 1. Otherwise, return to Step 2.

[Step 8] If any non-basic constraints of type B have never been satisfied, output them as possible failures. Terminate the procedure in all cases.

### 2.2. *Stopping Criterion for Model Execution*

Step 6 of the proposed algorithm requires a stopping criterion that influences the failure detection power of the ARS technique. The essential parts of abstract collaborative behavior, that is, the interactions among multiple systems, are defined as glue transitions in the EPNAT models. Therefore, we propose a stopping criterion that focuses on the glue transitions.

One model is composed of multiple sub-models that correspond to multiple systems. The sub-models are connected to each other by glue transitions. Defining glue transitions is a characteristic of modeling using EPNAT. When a glue transition is fired, attributes can be written and read among the sub-models connected by the glue transition. The reading of an attribute depends on the preceding writing of the attribute. Writing and reading will be related to markings on which they are executed.

According to the aforementioned discussion, the model execution should cover combinations of markings, writing, and reading related to glue transitions. Part of this concept was derived from data flow software testing [6]. The combinations are called "check objects" in this study, and are defined as follows:

$$C = \bigcup_{a\in A} \Big( \bigcup_{w\in W_a} (M_w \times \{w\}) \times \bigcup_{r\in R_a} (M_r \times \{r\}) \Big) \qquad (1)$$

where $C$ represents a set of check objects. $A$ represents a set of attributes. $W_a$ and $R_a$ represent sets of the writing and the reading of $a$ ($a\in A$), respectively. In each element of $C$, at least one of $w$ ($w\in W_a$) and $r$ ($r\in R_a$) is executed in actions of glue transitions. $M_w$ and $M_r$ represent sets of markings on which transitions accompanied by actions including $w$ and $r$ are fired, respectively. Note that the distributions of tokens on sub-models that are not connected by a glue transition to be fired are masked in each element of $C$. For example, if sub-models $x$, $y$, and $z$ have places $\{p_{x1}, p_{x2}\}$, $\{p_{y1}\}$, and $\{p_{z1}, p_{z2}\}$, respectively, and if only $x$ and $y$ are connected by a glue transition to be fired in an element of $C$, a marking in the element is expressed as [$\#p_{x1}$, $\#p_{x2}$, $\#p_{y1}$, -, -], where $\#p$ indicates the number of tokens in $p$. The distribution of tokens on $z$ was masked by "-", and thus markings were distinguished only by $\{p_{x1}, p_{x2}\}$ and $\{p_{y1}\}$. This masking technique is useful for reducing the number of elements of $C$.

Note that in most cases, equation (1) contains infeasible check objects. As in the case of programs, state-transition systems with pre-conditions (guards) and actions usually contain infeasible paths, and it is difficult to evaluate the feasibility of the given paths strictly [6] [7]. This is known as the feasibility problem. For this reason, we used a random search approach in the algorithm for model execution. When the number of check objects that have actually been reached during the model execution converges, the stopping criterion is satisfied in Step 6.

## 3. Prototype Tool and Discussion

The ARS technique requires tool support; therefore, we developed a prototype tool that provides core functions to

- edit an EPNAT model and its non-basic constraints,
- run the algorithm for model execution, and
- evaluate the stopping criterion for model execution.

The prototype tool runs on VDMJ (a command line tool that includes an interpreter for VDM++) [3] and Visual Studio Code (a sophisticated code editor). Figure 1 shows a screenshot of the prototype tool that is running the algorithm. It includes two windows: the left one shows a simplified EPNAT model under execution, and the right one shows an overview of the model execution process based on the number of check objects.

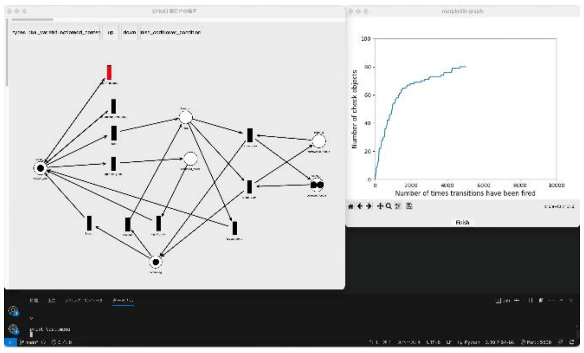


Figure 1. Screenshot of the prototype tool.

In addition, to discuss the effectiveness of the ARS technique, we constructed a trial model that represents the abstract collaborative behavior of a student management system (SMS) and book management system (BMS). SMS and BMS are imaginary systems outlined as follows:

- SMS and BMS store data on students and books at a university, respectively. SMS and BMS provide each student with functions to login, logout, request a leave of absence, request a resumption of studies, and borrow/return a book. Each student uses SMS and BMS through an integrated website, and thus SMS and BMS look like a single system.
- A student who is not on a leave of absence can login to borrow/return books. The student must return all his/her books before he/she requests a leave of absence.
- A student who is on a leave of absence can login only as a guest user and cannot borrow any books. The student is allowed to login to borrow/return books after he/she has resumed his/her studies.

Simultaneously, we defined the following three non-basic constraints for the trial model:

[C1] Each book that is in the state of having been borrowed in BMS is always in the state of having been borrowed by a student in SMS.

[C2] A student borrows multiple books sometime.

[C3] After each book has been borrowed, it is returned sometime.

C1 is classified as type A. C2 and C3 are classified as type B. C3 includes an order relation between two events, and thus C3 is a more complex constraint than C2. The trial model was constructed so as to satisfy them. Generally, constructing a model and its non-basic constraints may be difficult for SEs who are unfamiliar with formal methods.

Subsequently, we applied the ARS technique to SMS and BMS; that is, we ran the prototype tool with the trial model, three non-basic constraints, and a pseudo-initial state. As a result of this preliminary experiment, the prototype tool reported that no failures were detected, meeting our expectations. In addition, the prototype tool outputted an overview of the model execution process, as shown in Figure 2. A total of 85 check objects were reached in the trial model. The number of check objects increases rapidly during the early stages of model execution. It converges to 85 after approximately 8,000 transitions are fired. In general, reaching new check objects becomes more difficult as the model execution progresses. The prototype tool may have failed to reach certain check objects because of (1) the feasibility problem described in Section 2.2 and (2) the random search approach introduced into the algorithm. In addition, the total number of check objects will increase as a given EPNAT model becomes large.

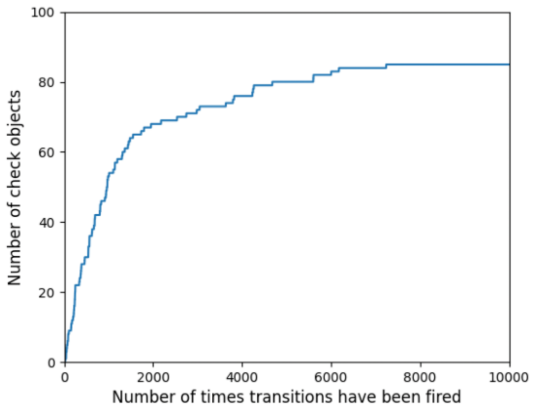


Figure 2. Overview of a model execution process.

To evaluate the failure detection power, we constructed three faulty models, F1, F2, and F3 that did not satisfy C1, C2, and C3, respectively, and we similarly ran the prototype tool five times with F1, F2, and F3, respectively. The results of these experiments are as follows:

- The prototype tool terminated the model execution for F1 after firing 13 transitions on average. It always outputted a definite failure that corresponds to the violation of C1.
- The prototype tool terminated the model execution for F2 and F3 after the number of check objects converged. It always outputted possible failures that correspond to violations of C2 and C3, respectively.

The faulty models will be relatively small. However, the prototype tool successfully detected all inserted failures in our experiments and showed the expected failure detection power.

## 4. Conclusion

In this study, we propose an ARS technique for checking the abstract collaborative behavior of multiple systems at the design level. The abstract collaborative behavior is expected to be designed by SEs using EPNAT, and the checking is performed dynamically based on the design called "EPNAT model". The ARS technique consists of (1) an algorithm for model execution using random search with the evaluation of constraints including feasibility, and (2) a stopping criterion for model execution focusing on glue transitions.

The ARS technique requires tool support; therefore, we developed a prototype tool. We ran the prototype tool with a trial model and three faulty models, and found the following:

- There may be check objects that the prototype tool failed to reach because of (1) the feasibility problem of the EPNAT models and (2) our random search approach. For example, metaheuristics that have been successfully applied to state-transition-based software testing [7] may be effective in reaching new check objects; this discussion will be addressed in future studies.
- The total number of check objects will increase as a given EPNAT model becomes large. The scalability of the ARS technique will need to be discussed in future studies. Addressing the scalability problem may require not only our masking technique but also additional techniques.
- The prototype tool successfully detected all inserted failures in our experiments and showed the expected failure detection power. The faulty models will be relatively small; thus, additional experiments in real-life development projects will need to be performed in the future for strict evaluation.
- Constructing a model and its non-basic constraints may be difficult for SEs who are unfamiliar with formal methods, which is one of the greatest obstacles to applying the ARS technique to real-life development projects.

**Acknowledgements**

A part of this work was developed from the graduation thesis written by Mr. Hikaru Morimoto at Kagawa

University. This work was supported by JSPS KAKENHI Grant Number JP22K11976, and Young Scientists Fund of Kagawa University Research Promotion Program 2021 (KURPP). We would like to thank Editage (www.editage.jp) for English language editing.

## References

1. S. Matsumoto, T. Katayama and T. Takagi, Automated Random Simulation for Checking a Behavioral Model of Systems Based on Extended Place/Transition Net with Attributed Tokens, *Proceedings of International Conference on Artificial Life and Robotics*, online, Japan, pp.337-340, Feb. 2023.
2. T. Takagi and R. Kurozumi, Software Modeling Technique and its Prototype Tool for Behavior of Multiple Objects Using Extended Place/Transition Nets with Attributed Tokens, *Journal of Robotics, Networking and Artificial Life*, Vol.7, No.3, pp.194-198, Dec. 2020.
3. J. Fitzgerald, P.G. Larsen, P. Mukherjee, N. Plat and M. Verhoef, *Validated Designs for Object-Oriented Systems*, Springer-Verlag London, 2005.
4. T. Takagi and R. Kurozumi, Simulation and Regression Testing Technique for Software Formal Specifications Based on Extended Place/Transition Net with Attributed Tokens, *Journal of Robotics, Networking and Artificial Life*, Vol.8, No.2, pp.112-116, Sep. 2021.
5. T. Takagi and Z. Furukawa, Test Case Generation Technique Based on Extended Coverability Trees, *Proceedings of 13th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, IEEE, Kyoto, Japan, pp. 301-306, Aug. 2012.
6. B. Beizer, *Software Testing Techniques*, Van Nostrand Reinhold, 2nd edition, 1990.
7. A. Kalaji, R.M. Hierons and S. Swift, Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM), *Proceedings of International Conference on Software Testing Verification and Validation*, IEEE, CO, USA, pp.230-239, April 2009.

## Authors Introduction

Mr. Sho Matsumoto

He received B.S. and M.S. degrees from Kagawa University, Japan, in 2023 and 2025, respectively. His research interests include software design and its quality control. This work was developed from his graduation thesis.

Dr. Tetsuro Katayama

He received a Ph.D. degree in engineering from Kyushu University, Fukuoka, Japan, in 1996. From 1996 to 2000, he has been a Research Associate at the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. Since 2000 he has been an Associate Professor at the Faculty of Engineering, Miyazaki University, Japan. He is currently a Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include software testing and quality. He is a member of the IPSJ, IEICE and JSSST.

Dr. Tomohiko Takagi

He received a Ph.D. degree in engineering from Kagawa University, Japan, in 2007. He became an Assistant Professor in 2008, and a Lecturer in 2013 at the Faculty of Engineering, Kagawa University. He became an Associate Professor in 2018, and a Professor in 2025 at the Faculty of Engineering and Design, Kagawa University. His research interests include software design and testing. He is a member of the IPSJ, IEICE and JSSST.