A Multimodel Framework for the Analysis of Processes Described Using UML

Katarina Samoylova
Institute of Physics and Mathematics
Perm State University (PSU)
Perm, Russia
e-mail: katarinasamoilova@yandex.ru
ORCID: 0000-0002-6867-076X

Elena Zamyatina
Department of Information Technologies in Business
HSE University
Perm, Russia
e-mail: e_zamyatina@mail.ru
ORCID: 0000-0001-8123-5984

***Abstract*—UML activity diagrams are widely used for process representation, but their lack of formal semantics limits rigorous analysis. This paper presents a multimodel framework that translates UML activity diagrams into timed Petri net models for validation and simulation. The approach enables the detection of structural and behavioral issues such as deadlocks, unreachable states, and performance bottlenecks. To illustrate the methodology, we apply it to two optimization algorithms — Combinatorial Artificial Bee Colony (CABC) and Discrete Chicken Swarm Optimization (DCSO) — showing how their UML-based descriptions can be systematically transformed and analyzed. The proposed framework improves model precision, supports iterative refinement, and provides a formal basis for evaluating dynamic, concurrent processes.**

***Keywords*—Petri nets, UML diagram, activity diagram, bio-inspired algorithm, simulation modeling.**

I. Introduction

Modern systems integrate parallel workflows and distributed operations, which increases the need for behavioral validation at the design stage. UML activity diagrams are widely used for their clarity in representing sequences, branching, and concurrency, but they lack formal semantics and provide limited support for time-dependent analysis. Although some extensions of UML with execution semantics have been proposed [1], they remain limited in scope and tool support. To address this gap, we propose translating UML activity diagrams into Petri nets, a formalism well-suited for detecting deadlocks, validating properties, and supporting simulation-based studies.

To demonstrate the applicability of our approach, we use two representative bio-inspired algorithms — Cooperative Artificial Bee Colony (CABC) [2] and Discrete Chicken Swarm Optimization (DCSO) [3]. These algorithms are frequently applied in optimization scenarios such as the Traveling Salesman Problem (TSP) [4], where robust and efficient routing solutions are required. Their relevance is especially notable in areas like IoT and smart city systems, where reliable message delivery remains a key challenge. By choosing algorithms that address the same class of combinatorial problems, but differ in their degree of maturity — one being widely established, the other relatively novel — we are able to evaluate the flexibility and generality of the proposed multimodel approach.

II. Related Works

The challenge of executing and validating UML-based models has attracted increasing attention over the past decade. In the systematic review by Ciccozzi et al. [1], the authors analyze 82 studies to classify the dominant approaches to UML model execution. They observe that translation-based methods — which convert UML models into executable code — are more prevalent than interpretative methods that simulate models directly. However, the review also reveals persistent limitations in existing approaches, including insufficient support for model validation, limited integration of temporal semantics, and a lack of formal consistency checks between the UML specifications and their executable counterparts.

Several studies have explored alternatives aimed at overcoming these challenges. In [5], the use of Foundational UML (fUML) and the Action Language for fUML (Alf) is examined as a way to define executable semantics. However, fUML supports only a restricted subset of UML, excluding essential constructs like time triggers and asynchronous events. Moreover, while Alf offers textual behavior descriptions, its integration with graphical diagrams can be cumbersome. To address this, ESysML is proposed as a broader modeling language that incorporates time and event handling. Despite its potential, ESysML remains in the early stages of development and lacks mature tool support.

In another line of work, Cengarle et al. [6] present a system-model-based simulation of UML using Haskell-based modular simulators. Their approach offers greater flexibility and platform independence, especially for simulating time-sensitive or resource-dependent behavior. However, it relies on custom infrastructure, limiting its accessibility.

To address these limitations, many researchers have proposed translating UML diagrams into formal models. Petri nets have proven especially effective, offering concurrency modeling, deadlock detection, and reachability analysis essential for validating complex systems. Prior studies [7], [8] demonstrate systematic transformations of UML Sequence and Activity Diagrams into Queueing and Timed Petri Nets for performance and scheduling analysis. Extensions such as colored, nested, and stochastic Petri nets [11-13] further enhance their expressiveness. Building on this foundation, our framework uses UML activity diagrams as input and translates them into timed Petri nets, enabling structural and behavioral validation while supporting reproducible simulations in platforms like AnyLogic [14] and NetLogo [15].

III. Multimodel Approach Overview

To ensure model correctness and support decision-making in complex systems, we adopt a multimodel approach [16] and validation workflow that integrates conceptual, graphical, and executable representations.

The proposed framework follows a multimodel workflow that integrates conceptual, graphical, and executable representations. At the first stage, a conceptual model (M) is defined. It is then represented as a graphical model (M_x) using a standard notation such as UML activity diagrams. Finally, the graphical model is transformed into an executable simulation model (M_{Yz}), where Y denotes the chose simulation platform (e.g., AnyLogic, NetLogo) and Z the mathematical formalism applied (e.g., Petri nets, queuing systems, Markov chains).

If risks are identified during the simulation, knowledge is extracted from a risk ontology regarding how to mitigate the detected threats [6]. This knowledge is then used to modify the original model M_x, and the process starts again. The cycle continues until the simulation experiment no longer reveals any critical threats.

Figure 1 presents the algorithm for applying the proposed software framework.

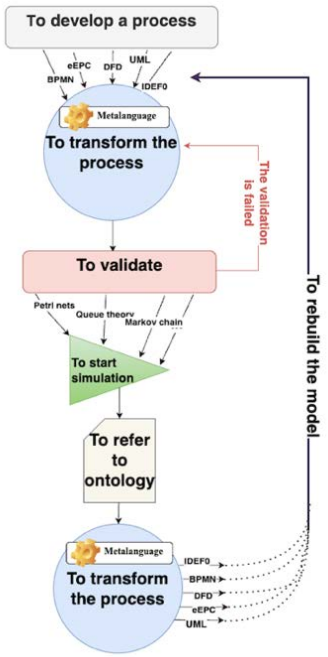

Fig. 1. Multimodel Approach

IV. Modeling Bio-Inspired Algorithms with Timed Petri Nets

To illustrate the framework, we selected two optimization algorithms as case studies: Combinatorial Artificial Bee Colony (CABC) [2] and Discrete Chicken Swarm Optimization (DCSO) [3]. Both address the Traveling Salesman Problem, but differ in maturity, with CABC being well-established and DCSO relatively new. Their UML activity diagrams were chosen as input models for translation into timed Petri nets.

Figure 2 shows the UML activity diagram of the CABC algorithm, which serves as a representative example. It demonstrates the sequential, branching, and iterative constructs that are formally captured during the transformation into a Petri net.

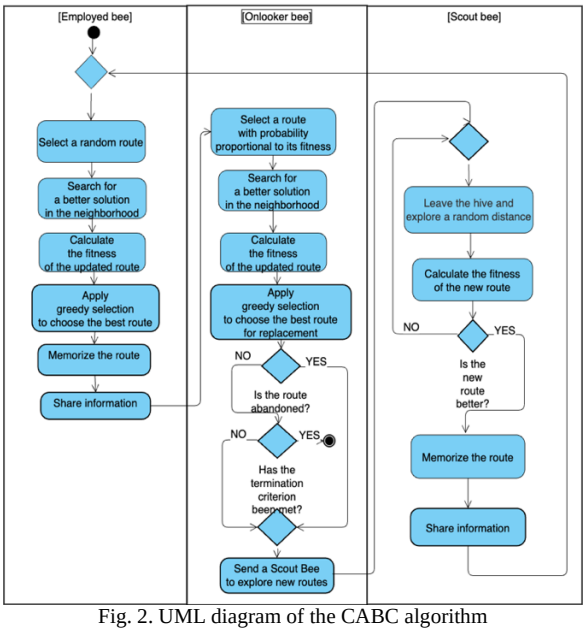

Fig. 2. UML diagram of the CABC algorithm

To demonstrate the application of this approach, we transformed the UML activity diagram of the Combinatorial Artificial Bee Colony (CABC) algorithm into a timed Petri net model. The transformation was carried out in accordance with the principles outlined in Section III, ensuring structural and behavioral correspondence between the original diagram and the resulting Petri net. The simulation model was implemented in NetLogo, with time delays assigned to transitions to reflect the algorithm's iterative nature and phase durations. This representation enables a detailed examination of the algorithm's control flow, including its cycles, stopping conditions, and branching logic.

The Petri net implemented in NetLogo, resulting from the transformation of the UML activity diagram of the CABC algorithm, is shown in the figure below (Fig. 3).

Petri net analysis techniques are generally classified into two main categories: behavioral and structural methods [17]. Behavioral analysis focuses on the dynamic execution properties of the net and includes the following:

- Reachability analysis: assesses whether a specific system state can be reached from the initial configuration — for instance, verifying whether a process can be completed successfully.
- Liveness analysis: ensures that the net does not contain deadlocks, and that all transitions remain potentially fireable, allowing the process to proceed indefinitely.
- Dead transition detection: identifies transitions that are never triggered during execution. Recognizing these inactive elements is useful for uncovering unused logic or unreachable process paths.

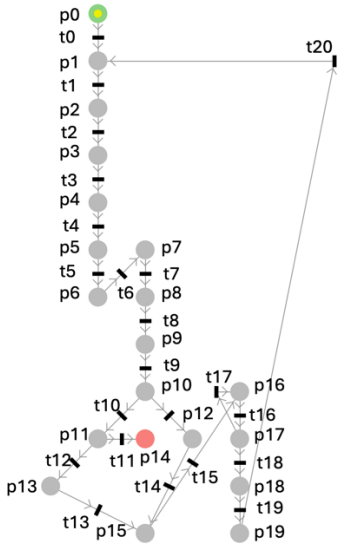

Fig. 3. Petri net of the CABC algorithm

The second category encompasses structural analysis methods, which focus on identifying specific configurations of places and transitions that influence the overall system behavior. As noted in [18], key structural components include traps, siphons, and cycles, each of which plays a critical role in determining the stability and correctness of the modeled process (see Fig. 4).

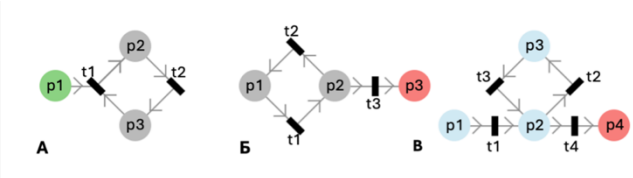

Fig. 4 Trap (A), Siphon (B), Cycle (C)

Let us now consider the resulting timed Petri net model of the bio-inspired Artificial Bee Colony algorithm. To perform reachability analysis, it is necessary to define the target state that the system is expected to reach. In our case, the desired condition is the activation of the termination criterion, which indicates that the predefined number of iterations has been completed. In terms of the timed Petri net, this corresponds to verifying that a token eventually reaches position p14, which represents the fulfillment of the stopping condition.

When analyzing the liveness of the Petri net, it is important to ensure that the system can execute indefinitely, meaning that all transitions remain fireable during the process. To verify this, an experiment was conducted with 10 tokens, and the statistics on the number of visits to each position were collected (Fig. 5).

```
observer: "Place 1 was visited 22 times."
observer: "Place 2 was visited 22 times."
observer: "Place 3 was visited 22 times."
observer: "Place 4 was visited 21 times."
observer: "Place 5 was visited 20 times."
observer: "Place 6 was visited 20 times."
observer: "Place 7 was visited 20 times."
observer: "Place 8 was visited 20 times."
observer: "Place 9 was visited 20 times."
observer: "Place 10 was visited 20 times."
observer: "Place 11 was visited 6 times."
observer: "Place 12 was visited 14 times."
observer: "Place 13 was visited 3 times."
observer: "Place 14 was visited 3 times."
observer: "Place 15 was visited 17 times."
observer: "Place 16 was visited 29 times."
observer: "Place 17 was visited 29 times."
observer: "Place 18 was visited 15 times."
observer: "Place 19 was visited 15 times."
```

Fig. 5. Token visit statistics for each position in the CABC Petri net

Since all positions in the net are activated during execution, we can conclude that there are no dead transitions. Additionally, the experimental results show that all initial tokens reach the final position, which indicates the absence of traps and siphons, as these are typically characterized by tokens getting stuck in specific states. However, in the analyzed net, there is a cyclic structure formed by the segment t15 → p16 → t16 → p17 → t17 → p16, as well as a secondary loop starting from p19 → t20 → p1, both of which constitute cycles within the model (Fig. 6).

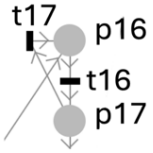

Fig. 6. Cyclic structure in the Petri net model

However, the presence of cycles is expected, as cyclic behavior is a fundamental characteristic of the algorithm: it iteratively repeats the search and update of solutions. It is important to note that the model includes termination criteria that limit the number of iterations, thereby preventing infinite execution.

To assess the temporal characteristics of a single iteration of the CABC algorithm, a series of simulation runs was performed using the corresponding Petri net model. In each run, a single token completed a full algorithmic cycle — from the beginning of the exploration phase to the end of the scout phase and returned to the initial state. The termination condition was temporarily disabled to allow uninterrupted execution. Based on the results of independent runs, the average duration of one iteration cycle was 18 ticks, which corresponds to 18 minutes under the assumed time scale (1 tick = 1 minute) (Fig. 7).

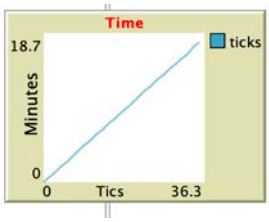

Fig. 7. Duration of one iteration cycle in the CABC

During the analysis of position activation frequency in the Petri net model of the CABC algorithm, it was observed that positions 16 and 17 were among the most frequently visited. These positions correspond to the phase in which the algorithm returns to the scout stage if the newly found solution is worse than the previous one. The increased activation frequency of this segment indicates that the mechanism of solution evaluation and reassessment is actively utilized during execution.

After completing the analysis of the CABC algorithm model, including both the structural characteristics of the Petri net and the results of simulation experiments, we now proceed to the second algorithm under consideration — Discrete Chicken Swarm Optimization (DCSO). The corresponding Petri net model of this algorithm is shown in the figure below (see Fig. 8).

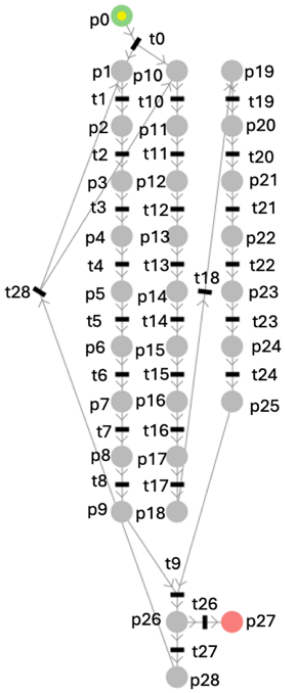

Fig. 8. Petri net of the DCSO algorithm

As with the previous Petri net, in the DSCO algorithm model, reachability analysis is performed by identifying the positions that represent the fulfillment of the termination criterion. In this case, this is the p27 position.

Continuing the analysis of the DCSO algorithm model, the Petri net was tested for liveness. A simulation involving 10 tokens was conducted, during which the number of visits to each position was monitored. The results showed that all positions were activated at least once, indicating the absence of dead transitions. Furthermore, none of the tokens became “stuck” in intermediate positions — each one reached a final state. This confirms the absence of traps and siphons, and verifies that the net is live, meaning it is capable of continuous operation in the presence of input tokens.

The DCSO Petri net also contains cyclic structures, which reflect its iterative logic but do not compromise correctness, as termination criteria are explicitly defined. Simulation showed that one iteration takes about 19 ticks, and activity is uniformly distributed across all positions. This balanced cycle confirms both behavioral correctness and predictable dynamics. Compared to the branched refinement structure of CABC, the DCSO model provides stable execution and broad exploration, making it suitable for discrete combinatorial problems.

V. Conclusion

This paper presented a multimodel framework that translates UML activity diagrams into timed Petri nets for formal validation and simulation. The approach combines the intuitive representation of UML with the analytical power of Petri nets enabling the detection of deadlocks, unreachable states, and performance bottlenecks. Case studies of CABC and DCSO demonstrated that both models are structurally correct and executable, yet differ in architecture: CABC emphasizes local refinement, while DCSO provides uniform cyclic exploration. These differences highlight the framework’s ability to capture algorithm-specific behaviors and evaluate their suitability for different problem domains. Overall, the proposed approach enhances model precision, supports iterative refinement, and can be extended to other processes and algorithmic paradigms in future work.

References

[1] F. Ciccozzi, I. Malavolta and B. Selic, "Execution of UML models: a systematic review of research and practice," *Software & Systems Modeling*, vol. 18, no. 3, pp. 2313–2360, Apr. 2018.

[2] D. Karaboga and B. Gorkemli, "A combinatorial artificial bee colony algorithm for traveling salesman problem," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, New Orleans, LA, USA, 2011, pp. 1–7.

[3] Y. Liu, Q. Liu and Z. Tang, "A discrete chicken swarm optimization for traveling salesman problem," *J. Phys.: Conf. Ser.*, vol. 1978, pp. 1–7, 2021.

[4] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *Eur. J. Oper. Res.*, vol. 59, no. 2, pp. 231–247, 1992.

[5] M. Amissah, "A Framework for Executable Systems Modeling," Ph.D. dissertation, Engineering Management & Systems Engineering, Old Dominion University, 2018.

[6] M. V. Cengarle, J. Dingel, H. Grönninger and B. Rumpe, "System-Model-Based Simulation of UML Models," arXiv:1409.6622, 2014. [Online]. Available: https://arxiv.org/abs/1409.6622.

[7] V.-D. Vu, T.-B. Nguyen and Q.-T. Huynh, "Formal transformation from UML sequence diagrams to queueing Petri nets," in *Proc. Int. Conf. Adv. Comput. Appl. (ACOMP)*, 2019, pp. 64–70.

[8] S. Distefano, M. Scarpa and A. Puliafito, "From UML to Petri nets: The PCM-based methodology," *J. Logic Algebraic Program.*, vol. 80, no. 1, pp. 25–38, 2011.

[9] A. V. Markov and D. O. Romannikov, "Algoritm avtomaticheskoy translyatsii diagrammy aktivnosti v set' Petri," *Vestn. Yuzh.-Ural. Gos. Univ. Ser. Vychisl. Tekhnol. Sistemy Upravl.*, no. 4(31), pp. 107–113, 2014. (In Russian)

[10] K. I. Bushmeleva and L. R. Zaripova, "Modelirovanie zhiznennogo tsikla programmnogo obespecheniya ot sbora trebovaniy do vnedreniya na osnove primeneniya UML-diagrammy," *Vestn. Kibernetiki*, no. 1(28), pp. 23–29, 2019. (In Russian)

[11] R. A. Dyachenko, A. V. Fisher and V. V. Bogdanov, "Modeling of data collection and transmission systems using colored Petri nets," *Fundamental Research*, no. 11-6, pp. 1122–1126, 2013. (In Russian)

[12] I. A. Lomazova, "Nested Petri Nets for Adaptive Process Modeling," in *Lecture Notes in Computer Science*, vol. 493, pp. 243–254, 2008.

[13] I. Sbeity, L. Brenner and M. Dbouk, "Generating a Performance Stochastic Model from UML Specifications," arXiv:1202.0414, 2012. [Online]. Available: https://arxiv.org/abs/1202.0414.

[14] Yu. Karpov, *Simulation modeling systems. Introduction to modeling with AnyLogic 5*. Saint Petersburg, Russia: BHV-Petersburg, 2005. (In Russian)

[15] U. Wilensky, "NetLogo: A simple environment for modeling complexity," *Proc. Int. Conf. Complex Systems*, pp. 16–21, 2001.

[16] K. V. Samoilova and E. Zamyatina, "Application of a multi-model approach for designing a parallel business process," *Informatization and Communication*, vol. 15, no. 5, pp. 112–118, 2024. doi: 10.34219/2078-8320-2024-15-5-112-118. (In Russian)

[17] V. B. Marakhovsky, L. Ya. Rozenblyum and A. V. Yakovlev, *Modelirovanie parallelnykh protsessov. Seti Petri*. St. Petersburg, Russia: Professionalnaya literatura, 2014. (In Russian)

[18] I. G. Fedorov, "Metod otobrazheniya ispolnyaemoy modeli biznes-protsessa v seti Petri," *Statistika i Ekonomika*, no. 4, 2013. (In Russian)