# 256. DISCRETE-EVENT SIMULATION: FOUNDATIONS, PROCESS-ORIENTED MODELING, AND APPLICATIONS

*Krat S. I., Levchenko K. M.*

*Belarusian State University of Informatics and Radioelectronics*
*Minsk, Republic of Belarus*

*Yeliseyeva V. A. – Senior Lecturer, Master of Arts*

This work reviews discrete-event simulation (DES) and its theoretical foundations, focusing on the process-interaction paradigm. It covers simulation evolution from event-scheduling to entity-oriented models in GPSS and SIMULA. Key strategies and process-interaction concepts are compared. DES applications in industry, healthcare, logistics, and communications are discussed, alongside trends like digital twins, hybrid models, and machine learning. Process-oriented DES remains fundamental for analysing complex systems.

Operations research, systems engineering, and computer science extensively employ simulation methods in cases where analytical solutions are inapplicable or insufficiently accurate. Within this domain, discrete-event simulation (DES) occupies a central position. Its defining feature is that the system state changes only at discrete points in time corresponding to the occurrence of events, while remaining unchanged between them [1]. In contrast, continuous simulation represents system dynamics through continuously evolving variables that are typically described by differential equations. The event-driven paradigm is particularly appropriate for systems in which changes occur at distinct moments, including service systems, manufacturing processes, logistics operations, healthcare workflows, and communication networks. This correspondence between modeling approach and system characteristics largely explains the sustained relevance and widespread use of DES.

The development of discrete-event simulation dates back to the early 1960s, a period characterised by limited computational resources and the need for efficient modeling techniques. Early simulation languages such as GPSS and SIMSCRIPT established a foundational set of modeling concepts, including entities, resources, queues, and processes, which remain fundamental to this day [2]. A significant milestone in the evolution of DES was the creation of SIMULA by Dahl and Nygaard. This language not only advanced simulation methodology but also introduced key principles of object-oriented programming, including classes and inheritance [3]. Originally developed for simulating maritime operations, SIMULA exerted a lasting influence on both simulation practice and software engineering. Contemporary simulation tools, ranging from commercial platforms such as Arena and AnyLogic to open-source frameworks such as SimPy and salabim, continue to build upon the conceptual foundations established during this formative period.

The construction of a discrete-event simulation model relies on several fundamental concepts. State variables define the condition of the system at a given moment in simulated time. Entities, representing dynamic elements such as customers, jobs, or data packets, traverse the system while competing for limited resources. These resources may include personnel, equipment, or communication capacity. When demand exceeds available capacity, queues are formed, which leads to delays and directly affects system performance. The simulation process is governed by two primary components, namely a simulation clock, which tracks the progression of virtual time, and an event calendar, which maintains an ordered list of scheduled events. The simulation engine iteratively processes events by selecting the next event in chronological order, updating the system state, and recording relevant performance measures such as waiting time, utilisation, and throughput [4]. This execution mechanism ensures that the complexity of the model is primarily determined by its logical structure rather than by the underlying simulation algorithm.

An important aspect of discrete-event simulation is the process of model verification and validation, which ensures that the constructed model is both correctly implemented and adequately represents the real system. Verification focuses on confirming that the model is consistent with its conceptual specification, whereas validation assesses the degree to which the model reflects actual system behavior. These procedures typically involve statistical testing, comparison with empirical data, and sensitivity analysis [4]. In addition, proper experimental design, including multiple simulation runs and confidence interval estimation, is essential for obtaining reliable results [5]. Without rigorous verification and validation, the reliability and practical applicability of simulation results remain limited.

Simulation logic can be organised using several fundamental approaches. The event-scheduling method represents the system as a set of event types, each associated with a specific handler responsible for updating the system state and scheduling future events. Although this approach is efficient and conceptually transparent, it may become difficult to manage as model complexity increases. Activity scanning provides an alternative perspective by periodically evaluating which activities can occur under the current system state; however, this method may lead to increased computational overhead. The process-interaction approach, which has become the dominant paradigm, models the behavior of individual entities as sequences of operations executed over time. In this framework, each entity follows a predefined process that includes waiting for

resources, undergoing processing, and leaving the system. This representation facilitates a more intuitive description of system behavior and improves model readability and maintainability [6].

The development of process-oriented modeling is closely associated with the work of Geoffrey Gordon, who introduced GPSS as a transaction-oriented simulation language [2]. Within this framework, entities move through a sequence of predefined blocks representing system operations such as resource acquisition, delays, and queueing. This approach significantly simplified the construction of simulation models. Further advancements were achieved in SIMULA, where the introduction of coroutines enabled the representation of processes as independent execution flows that can be suspended and resumed [3]. As a result, the need for explicit event management was reduced, and simulation models became more modular and flexible.

Modern simulation tools preserve the principles of process interaction while providing enhanced usability and flexibility. Commercial environments such as Arena and AnyLogic offer graphical interfaces for constructing simulation models [6]. In contrast, open-source frameworks such as SimPy implement process interaction using programming language constructs, including generators and coroutines [7]. Regardless of the implementation, the underlying modeling principles remain consistent, since entities interact with resources, experience delays, and progress through the system according to defined processes. This abstraction is sufficiently expressive to represent systems of varying scale and complexity, ranging from simple service models to large-scale distributed systems.

Another important aspect of discrete-event simulation is its ability to represent stochastic behaviour. Real-world systems are often subject to uncertainty, including random arrival times, service durations, and resource failures. DES models incorporate randomness through probability distributions, which allows for a more realistic representation of system dynamics [8]. Consequently, simulation outputs are analysed using statistical methods, including estimation of mean values, variances, and confidence intervals. This probabilistic nature distinguishes simulation from purely deterministic analytical approaches and significantly enhances its practical applicability.

In addition, experimentation within simulation models provides valuable insights into system behavior under varying conditions. By modifying input parameters such as resource capacity or arrival rates, it is possible to evaluate different scenarios without interfering with the real system. Such an approach is particularly useful in decision support, where simulation enables the comparison of alternative strategies and the identification of optimal configurations [5]. Therefore, DES serves not only as an analysis tool but also as an effective instrument for system design and improvement.

The utility of DES lies in its ability to model complex dependencies where resources are finite. In production and logistics, it moves beyond static averages to reveal how small delays in one sector cascade through the supply chain. In manufacturing, it is used to identify bottlenecks on assembly lines, minimise machine downtime, and optimise maintenance schedules [4]. The practical utility of discrete-event simulation (DES) lies in its ability to support strategic and operational decisions across different fields.

Recent developments indicate a growing integration of DES with other technological paradigms. The concept of digital twins involves the creation of dynamic virtual representations of physical systems, where DES serves as a core modeling approach. Additionally, machine learning techniques are increasingly incorporated into simulation workflows. For example, reinforcement learning methods have been successfully applied within simulated environments to derive effective control strategies, as demonstrated in production system optimisation studies [9].

In conclusion, discrete-event simulation has evolved from an early computational technique into a fundamental tool for the analysis of complex systems. The process-interaction paradigm provides an intuitive and effective framework for modeling system behaviour. The continued development of simulation technologies, combined with their integration with data-driven methods, ensures that DES will remain an essential component of modern scientific and engineering practice.

The transition from isolated models to integrated digital twins positions discrete-event simulation as a critical analytical engine for modern complex systems. By providing high-fidelity forecasts under stochastic uncertainty, DES remains an indispensable bridge between theoretical system architecture and operational optimisation.

***References:***

*1. Law, A. M., & Kelton, W. D. (2000). Simulation Modeling and Analysis (3rd ed.). McGraw-Hill*

*2. Gordon, G. (1961). A general-purpose systems simulation program. In Proceedings of the Eastern Joint Computer Conference (pp. 87–104). Macmillan. – URL: https://dl.acm.org/doi/10.1145/1460764.1460768. – (date of access: 10.03.2026).*

*3. Dahl, O.-J., & Nygaard, K. (1966). SIMULA: An ALGOL-based simulation language. Communications of the ACM, 9(9), 671–678. – URL: https://dl.acm.org/doi/10.1145/365813.365819. – (date of access: 14.03.2026).*

*4. Banks, J., Carson, J. S., Nelson, B. L., & Nicol, D. M. (2010). Discrete-Event System Simulation (5th ed.). Prentice Hall.*

*5. Robinson, S. (2014). Simulation: The Practice of Model Development and Use (2nd ed.). Palgrave Macmillan.*

*6. Kelton, W. D., Sadowski, R. P., & Zupick, N. B. (2015). Simulation with Arena (6th ed.). McGraw-Hill.*

*7. Zinoviev, D. (2024). Discrete event simulation: It's easy with SimPy! arXiv preprint, arXiv:2405.01562. – URL: https://arxiv.org/abs/2405.01562. – (date of access: 17.03.2026).*

*8. Fishman, G. S. (2001). Discrete-Event Simulation: Modeling, Programming, and Analysis. Springer.*

*9. Kuhnle, A., Kaiser, J.-P., Theiss, F., Stricker, N., & Lanza, G. (2020). Designing an adaptive production control system using reinforcement learning. Journal of Intelligent Manufacturing, 32, 855–876. – URL: https://link.springer.com/article/10.1007/s10845-020-01612-y. – (date of access: 17.03.2026).*