Computing, Telecommunication and Control, 2025, Vol. 18, No. 1, Pp. 23–35.
Информатика, телекоммуникации и управление. 2025. Том 18, № 1. С. 23–35.

Research article
DOI: https://doi.org/10.18721/JCSTCS.18102
UDC 004.94

A METHOD FOR MODELING OF INDIVIDUAL AGENT BEHAVIOR IN THE PROCESS-NETWORK PARADIGM OF DISCRETE-EVENT SIMULATION

A.M. Sabutkevich ✉ , *I.V. Nikiforov, A.V. Samochadin*

Peter the Great St. Petersburg Polytechnic University,
St. Petersburg, Russian Federation

✉ artem.sabut@gmail.com

Abstract. Due to the active growth of demand for cloud resources, the task of increasing the efficiency of their use becomes relevant. One of the approaches to solving this problem is the application of discrete-event simulation in the process-network paradigm, which allows describing the modeled process in the form of processing nodes united in a single network. However, this paradigm does not consider the individual behavior of agents, which reduces the adequacy of the resulting models. The paper presents an approach to assessing the adequacy and significance of simulation models, and proposes a method that allows specifying and considering of the individual behavior of agents in the simulation process, implemented in accordance with the process-network paradigm. The application of this method allows increasing the applied significance of the models of the processes under study. The paper describes the integration of the proposed method into systems implementing the process-network paradigm and presents its software implementation. The latter allows to investigate the influence of considering individual agent behavior on the adequacy and significance of the model of virtual machines placement relative to physical servers. Due to the application of the method, it was possible to achieve an increase in the adequacy of the model under study by an average of 12.5% and, as a consequence, to increase the number of significant models by 65% for the selected adequacy threshold value.

Keywords: discrete-event simulation, agent-based simulation, process-network paradigm, model adequacy, simulation of cloud infrastructures

Citation: Sabutkevich A.M., Nikiforov I.V., Samochadin A.V. A method for modeling of individual agent behavior in the process-network paradigm of discrete-event simulation. Computing, Telecommunications and Control, 2025, Vol. 18, No. 1, Pp. 23–35. DOI: 10.18721/JCSTCS.18102

Научная статья
DOI: https://doi.org/10.18721/JCSTCS.18102
УДК 004.94

МЕТОД СИМУЛЯЦИИ ИНДИВИДУАЛЬНОГО ПОВЕДЕНИЯ АГЕНТОВ В ПРОЦЕССНО-СЕТЕВОЙ ПАРАДИГМЕ ДИСКРЕТНО-СОБЫТИЙНОГО МОДЕЛИРОВАНИЯ

А.М. Сабуткевич ✉ , *И.В. Никифоров*, *А.В. Самочадин*

Санкт-Петербургский политехнический университет Петра Великого,
Санкт-Петербург, Российская Федерация

✉ artem.sabut@gmail.com

Аннотация. В связи с активным ростом спроса на облачные ресурсы актуальной становится задача повышения эффективности их использования. Одним из подходов для решения данной задачи является применение дискретно-событийного моделирования в процессно-сетевой парадигме, позволяющей описать моделируемый процесс в виде узлов-обработчиков, объединенных в единую сеть. Однако данная парадигма не учитывает индивидуальное поведение агентов, что снижает адекватность получаемых моделей. В статье представлен подход к оценке адекватности и значимости имитационных моделей, а также предложен метод, позволяющий задавать и учитывать индивидуальное поведение агентов в процессе симуляции, реализуемой в соответствии с процессно-сетевой парадигмой. Применение данного метода позволяет повысить прикладную значимость моделей исследуемых процессов. В работе описан процесс интеграции предложенного метода в системы, реализующие процессно-сетевую парадигму, а также представлена программная реализация предложенного метода, с помощью которой проведено исследование влияния учета индивидуального поведения агента на адекватность и значимость модели размещения виртуальных машин относительно физических серверов. Благодаря применению метода удалось достичь повышения адекватности исследуемой модели в среднем на 12,5% и, как следствие, повысить количество значимых моделей на 65% для выбранного порогового значения адекватности.

Ключевые слова: дискретно-событийное моделирование, агентное моделирование, процессно-сетевая парадигма, адекватность модели, моделирование облачных инфраструктур

Для цитирования: Sabutkevich A.M., Nikiforov I.V., Samochadin A.V. A method for modeling of individual agent behavior in the process-network paradigm of discrete-event simulation // Computing, Telecommunications and Control. 2025. Т. 18, № 1. С. 23–35. DOI: 10.18721/JCSTCS.18102

Introduction

The use of cloud infrastructures is one of the most promising areas of technology development [1], but the effective utilization of cloud distributed resources is often associated with the solution of NP problems such as determination of optimal resource allocation and defragmentation [2], organization of distributed training of machine learning models [3] and others. Simulation modeling [4], in particular process-network paradigm of the discrete-event simulation [5], is used to solve problems of this class.

The process-network paradigm allows to describe the simulated process in the form of processing nodes connected in a single network, through which agents move during a simulation [6]. Generally, the individual behavior of agents is not considered in discrete-event simulation. However, in order to create adequate and significant models of complex processes in the field of cloud technologies, the ability to consider and further analyze the behavior of agents is necessary.

Consequently, the task of improving the adequacy and significance of models by simulating individual behavior of agents within the process-network paradigm of discrete-event simulation is relevant.

The purpose of this work is to improve the adequacy and significance of complex process models in the cloud technology domain through a method of simulating individual agent behavior in a process-network paradigm implemented in a discrete-event simulation system.

Metrics of model adequacy and significance

The model adequacy metric determines the degree of closeness of the analyzed model to the corresponding representation of the real-world process, which has a direct impact on the informativeness and applied significance of the obtained simulation results [7].

The comparison of simulation results obtained by the model with the actual results of the behavior of the real process at equal values of input parameters is used to assess adequacy [8]. Actual results are the results obtained experimentally or theoretically, for example, by using heuristics. There are two main types of approaches to assessing model adequacy:

1) according to variance of deviations of simulation results from the mean value of actual results;
2) according to the average values of simulation and actual results.

The adequacy metric can be calculated based on variance of deviations of simulation results from the mean value of actual results, since the obtained results and initial values of parameters are statistical data, which allows using methods of statistical theory of estimation and hypothesis testing for this comparison [9–11]. The F-test, the Pearson's chi-squared test or the Kolmogorov–Smirnov test can be used for comparison of variances.

However, this approach imposes limitations on the analyzed results considered as statistical data. For example, to use the F-test, as the universal method, which is independent of the sample size, it is necessary to guarantee that the data have a normal distribution.

In this regard, the use of the approach of assessing the adequacy of the model by the variance of deviations of simulation results from the mean value of actual results is not possible within this work, since the resulting data can be distributed in any way.

Assessment of model adequacy by mean values of the results involves checking the proximity of the mean values of each component of the simulation results to the known mean values of the corresponding component of the actual results [12, 13].

This approach will be used in this work, since it does not impose additional restrictions on the analyzed data.

Proposed metrics of adequacy and significance

As a quantitative indicator of the model adequacy metric, we will use the ratio of actual estimates of the model target metrics determined by the model developer and their values obtained in the simulation process as follows:

$$MA = \frac{\sum_{i=1}^{n} w_i \left| v_{t_i} - v_{s_i} \right|}{\sum_{i=1}^{n} w_i \left| v_{t_i} \right|},$$

where n is the number of target model metrics selected by the model developer; w is the weight of each target metric; v_t is the actual evaluation of the value of the target metrics; v_s is the value of the target metric obtained during the simulation.

The model significance metric is determined based on the adequacy metric value as follows:

$$MS = \begin{cases} 0, & MA < T \\ 1, & MA \geq T \end{cases},$$

where T is the model adequacy threshold defined by the model developer.

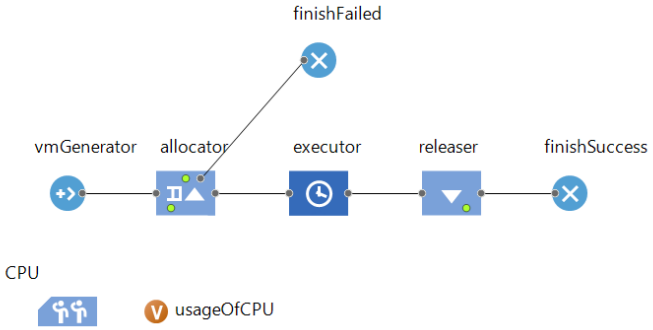

Fig. 1. A simplified model of virtual machines placements in accordance with the process-network paradigm

If the value of the model significance metric takes the value that is equal to one, then the model is considered significant for the selected adequacy threshold.

Components of the process-network paradigm notation

The simulation systems (AnyLogic [14], Arena [15], SIMAN [16]), which implement discrete-event simulation in the process-network paradigm, are based on similar proprietary notations and languages, which in general can be represented through the components described below.

The process under study, for example, the selection of an optimal route for sending a packet within a physical network, is formally specified by means of a model. A model is an entity that is characterized by general information and aggregates components used to describe the properties and behavior of a process [17]. The same process can be described through different models depending on the requirements to the modeling results.

The main types of model components in considered modeling notations, in accordance with the process-network paradigm, are blocks and links between them, defining in aggregate a directed network [5]. Each block has a behavior specific to its type and the number of ports used to specify the links between blocks. An example of a simplified model of virtual machines placement relative to physical servers, implemented using AnyLogic in accordance with the process-network paradigm, is presented in Fig. 1.

During simulation, blocks can generate events, which are applied to the model at a given point in model time [17]. Each of the events contains a pointer to the handler that will be called at the moment of event application. The handler in the model can be either the block that generated the event or a third-party service that implements the handler interface.

It is assumed that the event changes the state of the model, where the state of the model means the set of values of its properties at a certain point in time.

Another type of components is an agent which is a digital twin of a real-world object [18], characterized by information of general identification and classification nature. The movement of agents within the network is carried out through events that contain pointers to agents. By sequentially creating and processing events by blocks, agents move from generating blocks to absorbing blocks.

Blocks maintain a common interface for interacting with each other, which includes the operations listed below:

- initialization – implements the logic of initializing the initial state of a block when the simulation starts. During initialization events can be created, if no events are created at this stage – the simulation is terminated;
- check of the possibility of moving to a block – determines the possibility of an agent moving to a block based on the position of the currently processed block and the specified network structure. For example, if the number of agents in a block has reached its capacity, it is impossible to move to it;

• transition to a block – implements specific behavior of a computational block, including the creation of events. In the base case – it checks the possibility of transition to the next block, and if the transition is possible, it processes it;

• agent request from a block – notifies the block that one of the next blocks in the network is ready to accept an agent, in other words – a transition can be realized to one of the next blocks in the network.

The simulation process is implemented by auxiliary components: global model time and an array of scheduled events [6]. The event with the shortest application time is selected from the array of scheduled events. The global model time [19] is raised to the application time of the selected event, and then the handler of this event is called. The above steps are repeated until the array of scheduled events is empty or a critical modeling situation occurs.

The result of the simulation is a list of events applied to the model, which can be used to reconstruct the state of the model at each discrete point in time.

Another important result is a set of target metrics values, which are defined by the model developer. These values can be represented in various visual formats, which allows their further analysis to assess the adequacy and significance of the models.

Approaches to simulate agent behavior within discrete-event simulation

Agent-based simulation

In the field of simulation modeling, the agent-based modeling method is used to simulate the behavior of agents. Agent-based modeling is the most modern approach of simulation modeling [18]. When using this method, the modeled process is represented by means of a set of agents. The Fig. 2 shows an example of a simplified virtual machine life cycle model. This model is built in accordance with the method of agent-based modeling in AnyLogic environment.

Agents within this approach are autonomous real-world objects selected in some system of relations defined by the modeling goals [20]. It is worth noting that the same model can be represented through a different set of agents. Various notations, including our own, can be used to describe models.

Behavior rules are specified for each agent separately, and the behavior of the whole system is determined based on the result of their interaction.

In a more general case, the task considered in this paper can be reduced to the integration of the agent-based modeling method into the discrete-event modeling system.

Approaches of integration of agent-based simulation into discrete-event systems

In [21], an approach based on an initial classification of agents into types is considered. For each type a default behavior is defined, which cannot be overridden by the developer of the model, which is the main disadvantage of this solution.

The implementation of this approach is related to the creation of an additional component called the agent behavior simulator, which provides an interface to interact with the used simulation platform AutoMod [21].

In [22], a mechanism for translating a model built in accordance with the agent-based approach into a discrete-event model for its further simulation is described. The main purpose of applying this mechanism to the original model is to improve performance when simulating high-dimensional models.

The author of the article provides a formal proof that the behavior of agents in a discrete-event model can be implemented by means of a set of events for each of the possible actions of the agent. In addition, the implementation of the action itself, its influence on the state of the model, – by means of external functions. The need to specify an event for each possible action of an agent is a disadvantage of this approach, as it complicates the scalability of the model.

The FAMOS solution described in [23] is an agent-based modeling module for the DESMO-J platform that implements the discrete-event approach. Within this solution, when the state of the model changes, an event is generated and processed by the agent. While processing the event, the agent delegates its state change to another object that encapsulates its behavior.

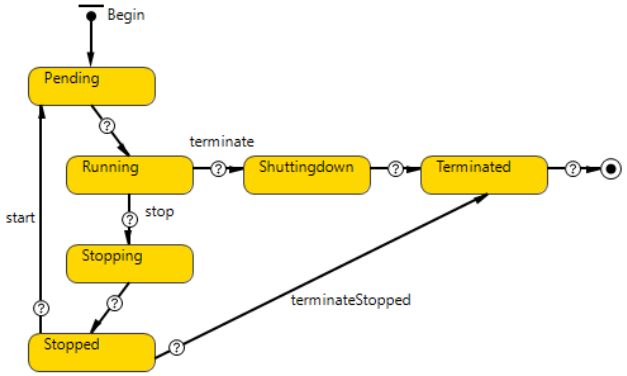

Fig. 2. A simplified model of the virtual machine lifecycle

A common drawback of the considered approaches and solutions is their focus on integration with specific modeling systems. Due to this, as well as the disadvantages mentioned earlier – the impossibility of overriding the behavior of agents and the need to manually set an event for each agent action, the task of developing our own solution is relevant. The solution proposed in this paper allows us to integrate the method of accounting for individual agent behavior into any modeling system supporting the process-network paradigm of the discrete-event approach. The adequacy and significance metrics proposed in this paper, the evaluation of which was not performed in the reviewed works, will allow us to quantitatively evaluate the obtained result of accounting for individual agent behavior in models.

Proposed method

The method proposed within this work based on:

- extending the modeling notation through properties, classes of states and trajectories of agent behavior to enable individual agent behavior to be considered;
- adding a mechanism for tracking changes in the agent's state using events;
- implementing an algorithm for simulating the agent's behavior in accordance with its life cycle described by means of a state machine.

The application of the proposed method will make it possible to specify and consider the individual behavior of agents in the simulation process, implemented in accordance with the process-network paradigm of discrete-event simulation, in order to improve the adequacy and significance of models.

The method has the following characteristics and properties:

- possibility of application for simulation modeling systems implementing the discrete-event approach;
- independence of implementation from the behavior of a particular type of agents;
- support of any level of detailing of model characteristics.

Extension of notation

To ensure that the individual behavior of the agent, considered as a digital twin of the real-world object, is considered, the modeling notation has been extended. A description of the changes is presented below.

One of the characteristics of an agent is a finite set of properties. Each agent is characterized by general identification and classification information, as well as a set of related special entities: properties, classes of states, and actions.

Properties describe key attributes of an agent. An agent does not possess all the attributes of a real-world object, as their number may be infinite. The attributes should be selected in such a way that they describe the distinctive features of the real-world object in the most accurate and complete way. The selected features are also determined by the modeling objectives.

Each of the given properties of the agent should take an initial value. In the process of modeling, the value can be changed both under the influence of external information entering the model and aimed at its actualization [24], and through various factors within the model.

A set of properties and their corresponding values at some point in time define a particular state of the agent. Some of the states having insignificant differences in property values can be combined into a single class of states.

A class of states is a set of admissible values specified for the agent's properties, which together specify some expected status of the agent. An agent can be in some class of states only if the values of its properties satisfy all the constraints specified for the related properties of the agent in the considered class of state [4].

Along with the allowed values for the class of states, the identification characteristics are specified, as well as the properties of the class of states that form the local environment for the agents in this class of states.

The change of class of states during simulation takes place by changing the values of agent's properties.

The set of classes of agent states and transitions that determine the possibility of their change can be formally represented in the form of a behavior graph. A behavior graph is an oriented finite graph [25] $GB = (CS, T)$, the nodes of which are classes of agent states (set CS), and the arcs (set T) are uniquely matched to transitions.

An individual behavior trajectory of an agent is a sequence of states classes connected by transitions, formed based on the initial behavior graph and observation of the agent in the process of modeling.

Formally, the behavioral trajectory is a mathematical oriented graph, where each vertex, except for the source and the sink, has a half-degree of output and a half-degree of input [26] equal to one.

Proposed simulation mechanism

Tracking agent state changes

In the course of simulation, the state of the agent changes due to the sequential execution of the logic of behavior of blocks. To track these changes, a new type of events was defined – tracking events. Events of this type are created when the values of the agent property change and contain a pointer to the agent, as well as information about the name of the property, its previous and new values.

Algorithm for simulating agent behavior

At the top level, the algorithm for simulating the agent's behavior is represented as a state machine [27] describing the agent's life cycle (see Fig. 3). Each of the agent's states within the life cycle is defined by its own algorithm of system behavior.

The state of identification of the initial class of states is the initial state for all agents of the model after starting the modeling process. Based on the initial values of the agents' properties set at the model creation stage, the only possible initial class of states is defined. In case an initial class cannot be defined for an agent or an agent can be placed in several initial classes of states at once, an agent life cycle modeling error occurs. If an initial class of states is defined, the agent enters the state of waiting for transition to another class of states.

The agent remains in state of waiting for transition to other classes of states until the next tracking event containing a pointer to it is processed. Based on the given graph of the agent's behavior and its current class of states, a set of possible classes of states to which this agent can transition is determined. This state can also be considered as an absorbing state, since the agent may not change its class of states during the whole simulation.

From the previously defined set of possible target state classes, the one whose constraints are satisfied by the new values of the agent's properties is selected. This is processed within state of checking attribute values are consistent with classes of states.

If the agent's property values do not satisfy the constraints of any of the selected classes of states, an agent life cycle error occurs. If the agent's property values satisfy the constraints of several classes of states at once, a single class is randomly selected.

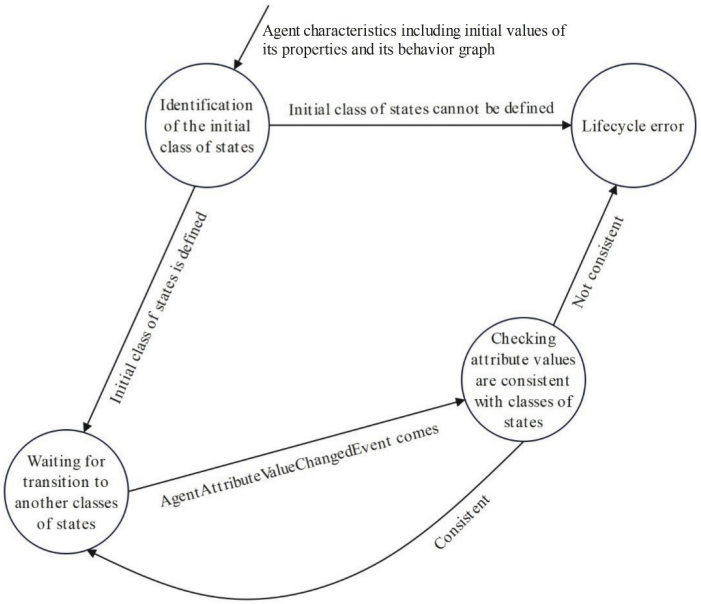

Fig. 3. Agent lifecycle state machine

If the agent is successfully placed in a class of states, it enters the state of waiting for transition to other classes.

Implementation of the proposed method

In order to integrate the proposed method, a prototype of the discrete-event simulation system was developed. It satisfies the generalized characteristics and requirements for the implemented notation described previously. This decision is explained by the need to test the method in a universal software environment, independent of the peculiarities of the implementation of private solutions and their additional functions.

The prototype of the system is implemented as a console application using the Java 17 program language. The diagram of classes implementing the proposed method as one of the system components is presented in Fig. 4.

Extension of notation within the data model

The necessary extension of the notation described in previous section involves changing the program data model, in particular, the agent entity.

Agent properties are implemented as generated fields of the corresponding "Agent" class and accessed by the "AgentAttributeValueChangeSupport" wrapper class. Additionally, to access agent properties without using reflection, a collection of pointers to them, stored as a separate attributes field, is implemented. It is not possible to get a value or change an agent property directly.

The class of states is implemented by means of the "ClassOfStates" class, which contains as collections pointers to agent properties and their corresponding ranges of possible values defined by means of the "AbstractRange" class. This class provides an abstract value checking method that must be overridden by the model developer.

Agent behavior is specified at the "Agent" class level as a collection of transitions between classes of states, implemented using the "Transition" class. It contains pointers to the source and target classes of states.

Implementation of the method at the level of the simulation execution

The event that tracks the change in the value of agent properties is represented by the "AgentAttributeValueChangedEvent" class, which implements a common event interface "Event". The attributes unique to other events for this event are:

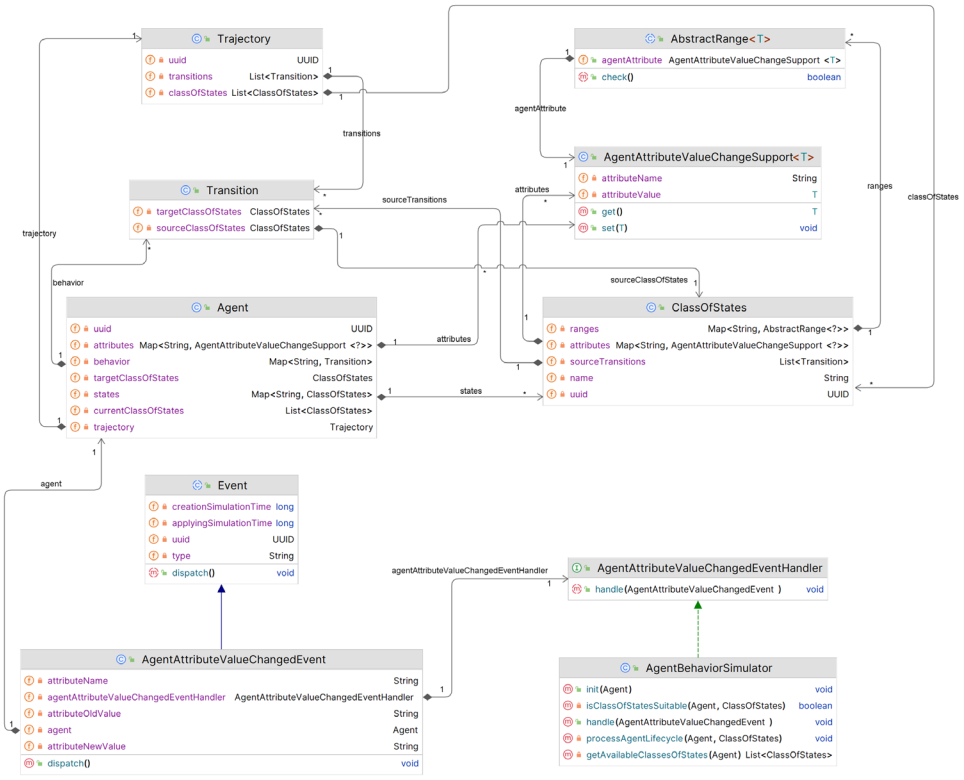

Fig. 4. Diagram of classes implementing the proposed method

• pointer to the handler of this event, whose interface is implemented by the “AgentBehaviorSimulator” class. It is a service that encapsulates the logic of simulating agent behavior according to the method described earlier;

• pointer to the agent whose state has been changed, and hence whose behavior change should be processed;

• name of the agent property whose value has been changed, as well as the previous and new values.

Event creation is implemented through the previously mentioned “AgentAttributeValueChangeSupport”, through which interaction with agent properties is performed.

When the value of an agent property changes, a tracking event is created with an application time that coincides with the model time of its creation. This event is placed in an array of scheduled events. At the same model time, the event is retrieved from the array and processed in the same way as other events in the system by supporting a common interface, namely by calling the “dispatch” method.

Implementation of the “dispatch” method for the tracking event is reduced to calling the handle method of the “AgentBehaviorSimulator” handler, which takes this event as an argument, further extracting the agent from it for subsequent operations according to the method described earlier.

Application of the proposed method to create an experimental model

In the process-network paradigm of discrete-event simulation, agents do not have individual behavior, and thus are indistinguishable in their state.

One example of a model in which, without considering the individual behavior of agents, the results obtained have little adequacy and significant for the analysis of a real process is the model of virtual machine placement relative to physical servers [28]. This is due to the fact that the server has individual behavior – they may fail, which will affect the configuration of the whole cluster.

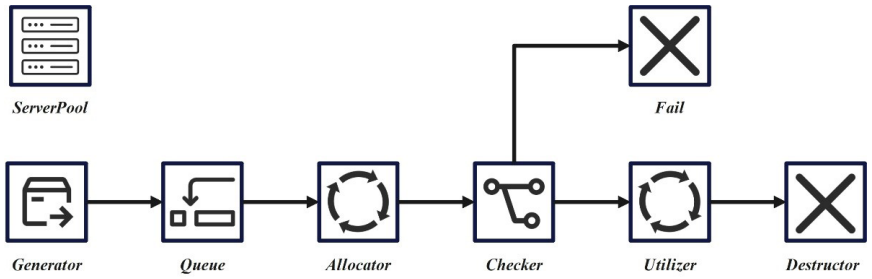

Fig. 5. The scheme of the process-network model of virtual machines placement

The use of simulation to analyze the process of virtual machine placement is significant due to the need to improve resource utilization efficiency, to predict the workload of the physical cluster and to scale it in a timely manner.

The target metric for resource utilization efficiency in this model is the resource allocation ratio, which ranges from 0 to 100%. In case the allocation ratio reaches its maximum value, it indicates a lack of resources and the need to increase the number of servers in the cluster. A low value of the allocation ratio corresponds to the case of inefficient resource utilization and the need to release servers from the cluster.

The model of this process is described by means of a process-network (see Fig. 5), which includes the following components:

- Generator – the initial unit that provides generation of requests from virtual machines to capture resources;
- Queue – realizes the accumulation of requests;
- Allocator – realizes the allocation of computational resources of servers for requests;
- Checker – checks resource allocation. If the resources are allocated - the resources can be used, otherwise an error of request processing occurs;
- Utilizer – delay block that simulates the time of using the captured resources;
- Destructor – releases the captured resources;
- Fail – terminates processing of the request with a resource allocation error;
- Servers pool – collection of available servers.

The presented model does not consider the behavior of agents; therefore, the simulation does not consider the possibility of breakdown or temporary decommissioning of a server defined as an agent within the model.

Considering this feature of server behavior in the process of modeling and further analysis of the results is important for the adequacy of the model, as it has a direct impact on the efficiency of computing resources.

Consequently, a behavior graph can be defined for the server, which describes a set of its possible classes of states. For simplification, the presented graph (see Fig. 6) contains three classes of states: exploited, failed and decommissioned.

Additionally, when attempting to place virtual machines, a check of the current server class of states has been added – it must be exploited. Separately handled cases of changing the server class of states at the moment of its use by a virtual machine in order to reallocate it on another server in case of failure of the current one.

As one of the simulation results, individual server behavior trajectories were generated that describe the sequence of state class changes. Based on these trajectories, an analytical model of cluster operation can be built.

An experiment was conducted for the described model. It consists of single simulation runs in order to calculate the target metric of the model – the resource allocation ratio for different cluster configurations

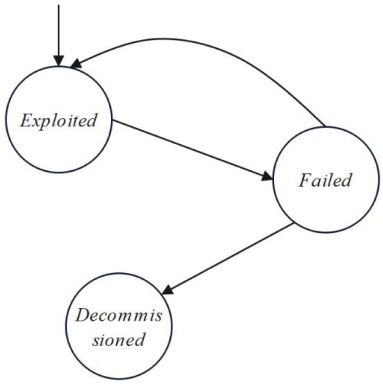

Fig. 6. Server behavior graph

and behaviors, as well as a set of virtual machine allocation requests. In the experiment, we used the adequacy threshold value $T = 0.9$, stochastic values are absent, which is explained by the need to fix the initial values of all parameters for accurate analysis of the obtained results. The obtained results are presented in Table 1.

Table 1

Results of assessment of adequacy and significance for the model, which describes the virtual machine placement relative to physical servers

No.	Cluster dimension	Number of requests	Number of server failures	Adequacy		Significance	
				Without method	With method	Without method	With method
1	12	2500	0	97%	97%	1	1
2	12	2500	3	89%	95%	0	1
3	40	10000	14	76%	95%	0	1

Based on the table we can conclude that for configuration No. 1, where there were no server failures, the adequacy and significance estimates coincide regardless of the application of the method. However, for configurations No. 2 and 3, where there is a need to consider the individual behavior of servers, it was possible to achieve an increase in the adequacy of the model on average by 12.5% and consequently increase the number of significant models by 65% for the selected threshold value of adequacy.

Conclusion

This paper proposes a method for simulating individual agent behavior in the process-network paradigm of discrete-event simulation. This method consists in extending the modeling notation by introducing entities that collectively describe the individual behavior of agents, as well as modifying the simulation algorithm.

The paper presents model adequacy and significance metrics to assess the degree of closeness of the model under study to the corresponding representation of the real-world process.

The application of the proposed method to the experimental model of virtual machines placement relative to physical servers demonstrated an increase in the model adequacy by 12.5%, and increased the number of significant models by 65%.

REFERENCES

1. **Hedge R.R., Narayan D.R., Moolya S., Chethan, Pushparani M.K.** A review on the future of technology: How cloud computing is changing the game. *International Research Journal on Advanced Engineering Hub* (*IRJAEH*), 2024, Vol. 2, No. 6, Pp. 1784–1793. DOI: 10.47392/IRJAEH.2024.0245

2. **Bohez S., Verbelen T., Simoens P., Dhoedt B.** Discrete-event simulation for efficient and stable resource allocation in collaborative mobile cloudlets. *Simulation Modelling Practice and Theory*, 2015, Vol. 50, Pp. 109–129. DOI: 10.1016/j.simpat.2014.05.006

3. **Hasan S.S., Zeebaree S.R.M.** Distributed systems for machine learning in cloud computing: A review of scalable and efficient training and inference. *Indonesian Journal of Computer Science*, 2024, Vol. 13, No. 2, Pp. 1685–1707. DOI: 10.33022/ijcs.v13i2.3814

4. **Sabutkevich A.M., Vikhlyayev D.A., Nikiforov I.V., Samochadin A.V.** Imitatsionnoye modelirovaniye povedeniya slozhnykh mnogoagentnykh sistem s ispolzovaniyem veroyatnostnoy modeli. [Simulation of behavior of complex multi-agent systems using probabilistic model]. *Sovremennye tekhnologii v teorii i praktike programmirovaniya* [*Modern technologies in theory and practice of programming*], 2022, Pp. 98–100.

5. **Wagner G.** Introduction to information and process modeling for simulation. *2017 Winter Simulation Conference* (*WSC*), 2017, Pp. 520–534. DOI: 10.1109/WSC.2017.8247812

6. **Wagner G.** An abstract state machine semantics for discrete event simulation. *2017 Winter Simulation Conference* (*WSC*), 2017, Pp. 762–773. DOI: 10.1109/WSC.2017.8247830

7. **Robinson S.** Exploring the relationship between simulation model accuracy and complexity. *Journal of the Operational Research Society*, 2023, Vol. 74, No. 9, Pp. 1992–2011. DOI: 10.1080/01605682.2022.2122740

8. **Ilin V.A., Kiryushow N.P.** Method of testing the training model for the adequacy. *Software & Systems*, 2021, Vol. 34, No. 1, Pp. 61–66. DOI: 10.15827/0236-235X.133.061-066

9. **Potapov A.N., Ovcharov B.B.** Assessment of the adequacy of simulation in simulators, èrgotehničeskih operators with hierarchical structure of building. *Bulletin of Voronezh State Technical University*, 2013, Vol. 9, No. 3–1, Pp. 45–48.

10. **Belyakov V.V., Tumasov A.V., Butin D.A., Vashurin A.S.** Adequacy simulation model of a light commercial car. *Trudy NGTU im. R.E. Alekseeva* [*Proceedings of NSTU n. a. R.E. Alekseev*]. 2021, Vol. 132, No. 1, Pp. 62–69. DOI: 10.46960/1816-210X_2021_1_62

11. **Spear R.C.** Large simulation models: calibration, uniqueness and goodness of fit. *Environmental Modelling & Software*, 1997, Vol. 12, No. 2–3, Pp. 219–228. DOI: 10.1016/S1364-8152(97)00014-5

12. **Minaev V.A., Stepanov R.O., Faddeev A.O.** Modeling of energy transition in a stress-strain geological environment for seismic risk assessment (Part 1). *Modeling, Optimization and Information Technology*, 2022, Vol. 10, No. 1, Pp. 1–19. DOI: 10.26102/2310-6018/2022.36.1.007

13. **Godunov A.I., Brostilov A.N.** Statisticheskiye kriterii otsenki adekvatnosti imitatsionnogo modelirovaniya v trenazherostroyenii. [Statistical criteria for assessing the adequacy of simulation modeling in simulator building]. *International Symposium "Reliability and Quality"*, 2005, Pp. 161–163.

14. **Borshchev A.V., Karpov Y.G., Kharitonov V.V.** Distributed simulation of hybrid systems with AnyLogic and HLA. *Future Generation Computer Systems*, 2002, Vol. 18, No. 6, Pp. 829–839. DOI: 10.1016/S0167-739X(02)00055-9

15. **Dias A.S.M.E, Antunes R.M.G, Abreu A., Anes V., Navas H.V.G., Morgado T., Calado J.M.F.** Utilization of the Arena simulation software and Lean improvements in the management of metal surface treatment processes. *Procedia Computer Science*, 2022, Vol. 204, Pp. 140–147. DOI: 10.1016/j.procs.2022.08.017

16. **Wagner G., Seck M., McKenzie F.** Process modeling for simulation: Observations and open issues. 2016 *Winter Simulation Conference* (*WSC*), 2016, pp. 1072–1083. DOI: 10.1109/WSC.2016.7822166

17. **Borshchev A.V.** Prakticheskoye agentnoye modelirovaniye i yego mesto v arsenale analitika. [Practical agent modeling and its place in the analyst's arsenal]. *Exponenta Pro*, 2004, Vol. 3–4, Pp. 38–47.

18. **Jabri A., Zayed T.** Agent-based modeling and simulation of earthmoving operations. *Automation in Construction*, 2017, Vol. 81, Pp. 210–223. DOI: 10.1016/j.autcon.2017.06.017

19. **Okol'nishnikov V.V.** Time representation in imitational modelling. *Computational Technologies*, 2005, Vol. 10, No. 5, Pp. 57–80.

20. **Gorodetskiy V.I., Karsayev O.V., Samoylov V.V., Konyushiy V.G.** Yazyk opisaniya mnogoagentnykh sistem. [Multi-agent system description language]. *Journal of Instrument Engineering*, 2008, Vol. 51, No. 11, Pp. 7–12.

21. **Dubiel B., Tsimhoni O.** Integrating agent based modeling into a discrete event simulation. *2005 Winter Simulation Conference* (*WSC*), 2005, pp. 1029–1037. DOI: 10.1109/WSC.2005.1574355

22. **Onggo B.S.** Running agent-based models on a discrete-event simulator. *Proceedings of the 24th European Simulation and Modelling Conference*, 2010, pp. 51–55.

23. **Page B., Knaak N., Kruse A.** A discrete event simulation framework for agent-based modelling of logistic systems. *INFORMATIK 2007*, 2007, pp. 397–404.

24. **Gorodetskiy V.I.** Printsipy avtonomnogo gruppovogo upravleniya [Principles of Autonomous Group Management]. *Integrirovannye modeli i myagkie vychisleniya v iskusstvennom intellekte* [*Integrated Models and Soft Computing in Artificial Intelligence*] (IMMV-2021), 2021, Vol. 1, Pp. 16–48.

25. **Riensche A., Severson J., Yavari R., Piercy N.L., Cole K.D., Rao P.** Thermal modeling of directed energy deposition additive manufacturing using graph theory. *Rapid Prototyping Journal*, 2023, Vol. 29, No. 2, Pp. 324–343. DOI: 10.1108/RPJ-07-2021-0184

26. **Nikiforov I.V., Petrov A.V., Yusupov Yu.V., Kotlyarov V.P.** Usage of formalization approaches for creation of system models from UCM-specification. *St. Petersburg State Polytechnical University Journal. Computer Science. Telecommunications and Control Systems*, 2011, Vol. 126, No. 3, Pp. 180–184.

27. **Sakellariou I.** Agent based modelling and simulation using state machines. *Proceedings of the 2nd International Conference on Simulation and Modeling Methodologies, Technologies and Applications* (*SIMULTECH 2012*), 2012, Vol. 1., Pp. 270–279. DOI: 10.5220/0004164802700279

28. **Dong D.** Agent-based cloud simulation model for resource management. *Journal of Cloud Computing*, 2023, Vol. 12, Art. no. 156. DOI: 10.1186/s13677-023-00540-5

INFORMATION ABOUT AUTHORS / СВЕДЕНИЯ ОБ АВТОРАХ

Sabutkevich Artem M.
Сабуткевич Артем Михайлович
E-mail: artem.sabut@gmail.com

Nikiforov Igor V.
Никифоров Игорь Валерьевич
E-mail: igor.nikiforovv@gmail.com

Samochadin Alexander V.
Самочадин Александр Викторович
E-mail: samochadin@gmail.com

Submitted: 08.11.2024; Approved: 10.02.2025; Accepted: 13.03.2025.
Поступила: 08.11.2024; Одобрена: 10.02.2025; Принята: 13.03.2025.