Process Science

RESEARCH **Open Access**

Discovering multi-agent systems for resource-centric business process simulation

Lukas Kirchdorfer[1,2*], Robert Blümel[1,3], Timotheus Kampik[1,4], Han van der Aa[3] and Heiner Stuckenschmidt[2]

*Correspondence: lukas.kirchdorfer@sap.com

[1] SAP Signavio, Walldorf, Germany
[2] Data and Web Science Group, University of Mannheim, Mannheim, Germany
[3] Faculty of Computer Science, University of Vienna, Vienna, Austria
[4] Department of Computing Science, Umeå University, Umeå, Sweden

Abstract

Business process simulation (BPS) is a powerful tool for estimating process performance across different scenarios, offering critical support for organizational process redesign and optimization. Traditional BPS approaches predominantly rely on a control-flow-first perspective by enriching a process model with simulation parameters. While these approaches seem suitable for capturing centrally orchestrated processes, such as those managed by workflow systems, they fall short of accurately reflecting real-world processes characterized by decentralized decision-making and distinct resource behaviors. To overcome this limitation, we propose *AgentSimulator*, a resource-first BPS approach that discovers a multi-agent system from an event log. By modeling the distinct behaviors and interaction patterns of individual resources, AgentSimulator effectively simulates the underlying process. Our approach automatically identifies whether resource behavior is rather orchestrated or autonomous, adapting to the specific decision-making structure of the process. Experimental results reveal that AgentSimulator achieves state-of-the-art simulation accuracy while ensuring high adaptability to various process types.

Keywords: Business process simulation, Multi-agent system, Process mining, Artificial intelligence

Introduction

Business process simulation (BPS) is a key tool for improving and redesigning organizational processes. By establishing a *digital process twin* (Dumas 2021), simulation can be used to estimate the impact of changes to a process with respect to key performance indicators, such as cycle time, resource utilization, or waiting time for a given activity—a practice known as counterfactual reasoning or "what-if" analysis (Dumas et al. 2013). By providing such estimates of the impact of process changes prior to their actual implementation, BPS has the potential to drastically improve the efficiency and reduce the risks of redesign efforts for decision-makers. Nevertheless, the effectiveness of BPS relies heavily on the availability of a simulation model that precisely mirrors the characteristics of a process, since only accurate models can provide trustworthy insights into the impact of a change. Since the manual construction of simulation models is time-consuming and error-prone due to several pitfalls (van der Aalst 2015), various automated approaches have been developed that discover simulation models directly from historical execution data contained in event logs (Rozinat et al. 2009; Camargo et al. 2020; Meneghello et al. 2023; Camargo et al. 2022; Khodyrev and Popova 2014). These data-driven simulation approaches typically initially discover a process model representing the control flow of the entire process and then enhance it with simulation parameters, such as arrival rates and resource availabilities.

This paper emphasizes that control-flow-first simulation models may fall short of accurately reflecting the complexity of certain real-world processes, leading to potential inaccuracies in simulation outcomes. This issue is particularly pronounced in processes where resource behavior varies significantly, or decision-making is decentralized. While some processes operate under centralized coordination, such as those managed by workflow systems (Dumas et al. 2013), many others allow participants greater autonomy in how they execute their tasks. In such scenarios, each actor treats their part of the process execution from their own perspective and, to some extent, in their own manner, i.e., they receive a case from a co-worker, conduct one or more tasks they deem necessary, and pass on the case to the next individual or system. These distinct behaviors, preferences, and decision-making patterns can significantly shape the process execution, creating dynamics that are challenging for traditional control-flow-first approaches to model accurately.

Therefore, we use this paper to highlight the potential of resource-centric simulation and propose *AgentSimulator*, a resource-first approach for data-driven process simulation. To achieve this resource-centricity, we employ the concept of agent-based modeling and simulation (North and Macal 2007). By discovering a multi-agent system (MAS) from an event log, where each agent corresponds to a real-world actor or system, AgentSimulator can simulate the execution of a process through interactions of distinct agents. This allows our approach to achieve various benefits in comparison to existing approaches for data-driven process simulation:

- our approach provides full flexibility over the behavior of individual resources involved in a process, allowing to capture differences in terms of control-flow behavior, interaction preferences, capabilities, and availabilities;
- our approach is modular in how resource behavior is learned, allowing the integration of various methods, three of which are presented in this work: *Frequentist probabilities, LSTM networks*, and *Petri nets*;
- agent-based systems are, contrary to black-box deep learning models, interpretable and adaptable, which is crucial to performing what-if analyses;
- our approach provides a runtime-efficient alternative compared to deep learning-based approaches;
- and finally, our approach yields state-of-the-art simulation accuracy, achieving competitive performance when compared to 4 existing approaches across various process dimensions on a number of event logs.

This paper presents an extended and revised version of our earlier work (Kirchdorfer et al. 2024) in which we introduced the first version of our agent-based approach for process simulation. The current paper extends our earlier work in two main directions. First, we broaden the scope of the approach by incorporating two additional methods for learning agent behavior—*LSTM networks* and *Petri nets*—beyond the simple frequentist approach covered in the original version. Second, we conduct extensive evaluation experiments to assess these extensions, examining the applicability of the different methods for learning agent behavior and identifying scenarios where modeling individual agent behavior is particularly critical to gaining realistic simulations.

The remainder starts with a motivating scenario in the Motivation section, followed by the presentation of the AgentSimulator approach itself. Then, the Experiments and results section reports on several evaluation experiments, before the Related work section discusses related work and the Conclusion section concludes the paper.

Motivation

This section illustrates the benefits of shifting simulation models from a control-flow-first to a resource-first perspective.

For this illustration, we consider a simplified credit application process, for which a schematic visualization is shown in Fig. 1. As depicted, the process starts when an application is received by the system, after which the applicant's credit history and income sources need to be checked by a clerk (in any order). Once both checks have been completed, the application is passed on to a credit officer, who assesses the application and notifies the applicant of the outcome. As shown, there are three clerks (Steve, Oliver, and Angela) and two credit officers (Maria and Patrick) involved.

Even for such a simple scenario, we may observe various ways in which the involvement of specific actors in a case can influence its execution:

- *Process performance.* The execution time of an activity may depend on the employee who performs it. For example, Steve (a less experienced, Junior Clerk) might need 45 minutes to check the credit history and the income sources, respectively. For the same activities, the Senior Clerks Oliver and Angela only need 15 to 20 minutes. Considering these performance differences between resources is critical for accurate simulation, as has been recently demonstrated (López-Pintado and Dumas 2022).
- *Resource availability.* Employees involved in a process may have different availabilities, owing to factors such as part-time work and other duties. Such considerations are particularly relevant in decentralized processes, where cases may be

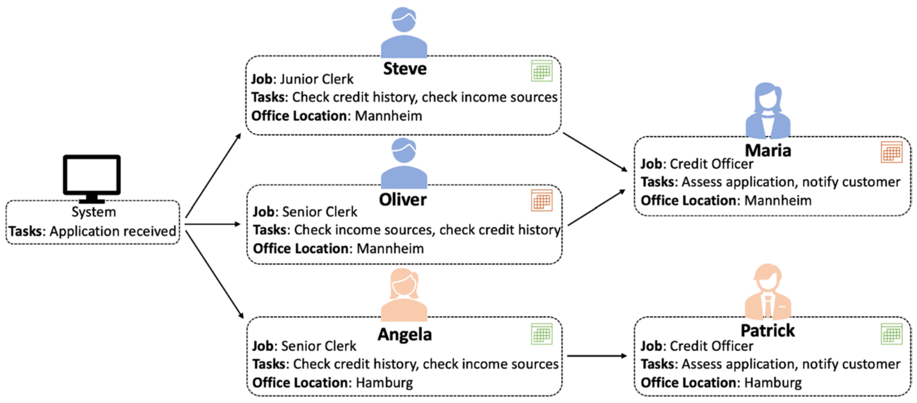

System
Tasks: Application received
Steve
Job: Junior Clerk
Tasks: Check credit history, check income sources
Office Location: Mannheim
Oliver
Job: Senior Clerk
Tasks: Check income sources, check credit history
Office Location: Mannheim
Angela
Job: Senior Clerk
Tasks: Check credit history, check income sources
Office Location: Hamburg
Maria
Job: Credit Officer
Tasks: Assess application, notify customer
Office Location: Mannheim
Patrick
Job: Credit Officer
Tasks: Assess application, notify customer
Office Location: Hamburg

Fig. 1 Sketch of the credit application process

handed over directly from employee to employee, rather than by a central workflow system that can assign a case to the next available person. For example, if Angela hands over an application specifically to Patrick, rather than to the next available credit officer, this can lead to considerable delays in the case's execution if Patrick is not available for the next working days. Such irregularities should be reflected in a simulation model, calling for individual resource calendars, as again was recently demonstrated to positively impact accuracy (López-Pintado and Dumas 2022).

- *Control-flow behavior.* Control-flow-first simulation models impose the assumption that the sequence of activities performed for a given case is independent of the actors involved in it. However, there can be various reasons why this is not the case. In our scenario, for example, we may observe that Angela always checks the credit history first, before checking the income sources, whereas other actors may alternate these orders. Furthermore, there may even be actor-specific rules that influence the possible sequences of a case. For example, it may be necessary that any application handled by a Junior Clerk (e.g., Steve) needs to go through an additional verification step performed by a Senior Clerk.
- *Interaction patterns.* Finally, owing to a lack of central orchestration, there may be specific interaction patterns among actors. For example, we may observe that Steve always hands over a case to Maria, since they both work in Mannheim, while he has no contact with Patrick in Hamburg. Such specific patterns influence the workload of individual resources, e.g., leading to an imbalance between Maria and Patrick, which is missed by general simulation models. Furthermore, as described above, specific interaction patterns may lead to additional delays when the availability of individual resources differs.

All of the above factors influence the execution of cases in a process. Therefore, process simulation models should reflect these as faithfully as possible in order to appropriately mimic the dynamics of a real-world process. The former two aspects have already been recognized and captured by control-flow-first models (López-Pintado and Dumas 2022). However, the latter two aspects have not been considered so far in previous BPS models. We argue that agent-based simulation models are particularly suitable to explicitly model distinct resource behaviors as well as their respective interaction patterns. Therefore, our proposed AgentSimulator approach, which is described next, models a process as a system of distinct agents.

Our approach: AgentSimulator

This section introduces AgentSimulator, our data-driven agent-based business process simulation approach, with a high-level overview in Fig. 2. To adopt a resource-first perspective, AgentSimulator models agents in an MAS, which is (along with general simulation parameters) discovered from an event log in the discovery phase. Following the MAS discovery, we can simulate the process and generate a new event log. We detail our approach for both discovery and simulation, also discussing alternative configuration options for different process types, highlighting AgentSimulator's adaptability.

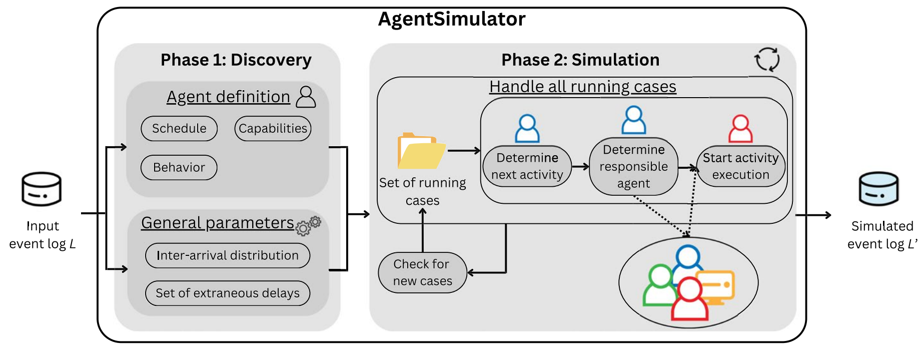


Fig. 2 Overview of the end-to-end AgentSimulator approach

Definitions

In this section, we define event logs, as they constitute the input to our approach, which in turn generates another event log as its output. Afterward, we define agents and the resulting multi-agent system (MAS), the underlying model of our approach.

Event log. We define an event log L as a finite multi-set of traces. A trace $\sigma \in L$ is a finite sequence of events, $\langle e_1, ..., e_n \rangle$, recording the execution of activities performed for a single case in an organizational process. Each event e_i is a tuple $(act, ts_{start}, ts_{end}, res)$, where act is the activity to which the event corresponds, ts_{start} and ts_{end}, respectively, are the start and end timestamps of the activity's execution, and res is the resource that executed the activity. Note that we follow López-Pintado and Dumas (2022) by representing each event with a start and end timestamp, which is required in simulation settings to consider activity durations, and ordering the events in a trace based on their start timestamp.

In the remainder, we commonly use dot notation to refer to components of tuples, e.g., using $e_i.act$ as a shorthand to refer to the activity of an event e_i, whereas we use ACT_L and RES_L to, respectively, refer to the sets of activities and resources contained in the traces of event log L.

Agent. The notion of an agent is a fundamental abstraction in Artificial Intelligence (AI) (Russell and Norvig 2020). Agents perceive their environment (including other agents) to reason about the perceptions and decide on actions, which are then executed against the environment. In our work, we use agents to represent actors and systems involved in a process. We define an agent $a \in A$ as a tuple $a = (s, c, b)$, where:

1. s refers to a **schedule** in the form of a weekly calendar, indicating intervals during which the agent is available to perform activities. The simulation needs to capture the distinct availabilities of agents (e.g., full-time vs. part-time) to faithfully reflect the temporal dimension of the process. Following López-Pintado and Dumas (2022), a weekly calendar can be defined as a binary relation $W \times \Delta$, where $W = \{\text{Monday}, \ldots, \text{Sunday}\}$ represents weekdays, and $\Delta = \{\delta_1, \ldots, \delta_n\}$ is a set of time intervals. Each time interval $\delta_i \in \Delta$ is a pair $\langle \tau_s, \tau_c \rangle$ with start and end times $\tau_s, \tau_c = \langle \text{hour}, \text{minute}, \text{second} \rangle$, satisfying $\tau_s \leq \tau_c$. A calendar entry $\langle \omega, \tau_s, \tau_c \rangle$ specifies an interval for day ω. For example, $\langle \text{Monday}, 08:15:00, 12:00:00 \rangle$ represents Monday from 08:15 to 12:00.

2. c refers to the **capabilities** of an agent, denoted as a tuple $c = (\textsc{Alloc}, PT)$, where:
    - $\textsc{Alloc} \subseteq ACT_L$ is the set of activities that a can execute, i.e., that can be allocated to agent a.
    - PT refers to a set of PDFs, where each $f_{pt}(act) \in PT$ captures a distribution over processing times $pt \in [0, \infty)$ for an activity $act \in \textsc{Alloc}$. Defining PT per agent allows us to differentiate process performance across resources.

3. b refers to the **behavior** of an agent, capturing how agent a hands a case over to continue its execution after finishing an activity. What b exactly entails depends on the configuration of AgentSimulator being used. Since business processes can be either centrally orchestrated or more autonomous, the behavior in AgentSimulator can be defined for both process types accordingly. Therefore, we provide further details and exact definitions in the following sections.

Multi-agent system. A multi-agent system (MAS) is composed of multiple agents that operate within a shared environment, potentially working together to achieve specific goals. We define an MAS for AgentSimulator as a tuple $\mathcal{M} = (A, P)$. Here, A corresponds to the set of agents, whereas P is a tuple of general simulation parameters, reflecting aspects of the process's environment. In our MAS, P consists of a case interarrival distribution, an arrival calendar, and a set of probability density functions (PDFs) over extraneous delays (more details next).

Phase 1: MAS discovery

In this section, we describe the discovery phase of AgentSimulator that yields an MAS $\mathcal{M}$. This phase covers, as visualized in Fig. 2, the instantiation and parameterization of agents in the set A and the general simulation parameters P from an event log L. In the remainder of this section, we describe the details of this discovery phase, starting with the instantiation of agents, followed by the derivation of their three attributes: schedule, capabilities, and behavior. Here, we note that the first two attributes are aspects commonly considered in non-agent-based simulation approaches as well, which is why we largely follow existing works for their discovery. In contrast, the distinctive agent behavior serves as the main ingredient of AgentSimulator, for which we discuss a variety of configuration options, enabling adaptation to diverse collaborative work dynamics.

Agent instantiation

We start by instantiating one agent $a \in A$ for each resource $res \in RES_L$. For our motivating example, we thus instantiate six agents (five human actors and *System*). Since event logs may contain events that do not have any associated resource, we add additional *dummy* agents to A that will be charged with the execution of their corresponding activities. Specifically, we generate a single dummy agent for each activity in ACT_L for which event log L contains one or more events without resource information.

Assigning dummy agents for such events primarily ensures that any activity can be associated with at least one agent, which is required for the simulation. Furthermore, defining dummy agents allows us to differentiate them in terms of their attributes schedule, capabilities, and behavior, enabling a more appropriate simulation of their respective activities. Importantly, dummy agents are treated in the same way as agents representing known resources. For example, if the recorded activity durations for a dummy agent are consistently 0, the simulation models these activities as instantaneous. Conversely, if a dummy agent is associated with non-zero activity durations, these durations are accurately reflected in the simulation, including any potential implications such as resource contention. An analysis of the event logs used in this study's evaluation revealed that approximately half of the logs contain events with missing resource information. Among these events, around 30% correspond to activities with instantaneous durations, whereas the remaining events exhibit a high variability between different activities in terms of the length of their duration. This highlights the necessity to model these dummy agents as well in a differentiated way.

Agent schedule

We discover a schedule $a.s$ for each agent $a \in A$ by using the algorithm proposed in López-Pintado and Dumas (2022), which creates a weekly calendar based on the observed times agent a was active in L. The algorithm divides the timeline into fixed-size intervals (time granules) and maps the start and end timestamps of events to these intervals. Confidence and support metrics are then used to identify recurring availability patterns: confidence measures the proportion of observed activity occurrences within a specific time interval across all relevant days, while support evaluates the proportion of an agent's total activity timestamps covered by the selected intervals, ensuring the calendar reflects consistent and representative working patterns. The schedule could, for example, indicate that *Steve* works Monday to Friday from 08:00 to 16:00, whereas *Oliver* works Monday to Friday only in the mornings. This schedule is a critical attribute for our MAS, as it must closely reflect reality to accurately model resource utilization, waiting times, and cycle times. While the approach enables the discovery of agent-specific availability, which has been shown to produce more realistic simulation outcomes than generalized calendars (López-Pintado and Dumas 2022), it may struggle to capture complex patterns that deviate from weekly periodicity, and limited data availability can impede robust discovery. Note that the more recent approach to discovering probabilistic instead of crisp calendars (López-Pintado and Dumas 2023) could also be integrated.

Agent capabilities

The capabilities $a.c$ of an agent a consist of a set of activities and a set of PDFs over the processing times for these activities.

Set of activities. We determine the set of activities $a.c.$ Alloc that agent a can execute by straightforwardly checking which activities from ACT_L (the resource corresponding to) a has performed in L.

Processing times. The set of PDFs over processing times $a.c.PT$ contains a distribution for each activity in $a.c.$ Alloc, which jointly capture how long agent a requires to execute each of these activities. Thus, we estimate an agent's performance by modeling the duration of each activity as a separate probability distribution, resulting in one distribution per agent-activity pair. Following López-Pintado and Dumas (2022), for each agent-activity pair, we fit a set of different distributions to the observed durations that it took the resource corresponding to agent a to execute a certain activity. This set of distributions includes Exponential, Gamma, Normal, Uniform, Log-Normal, and a fixed value to capture activities with fixed durations. Subsequently, we select one of these distributions that most closely represents the observed durations, measured by the earth mover's distance. More formally, this yields a mapping $ACT_L \times A \rightarrow PTT$, where $PTT = \bigcup_{a \in A} a.c.PT$, ensuring the simulation of realistic activity processing times, as, for example, a junior employee typically requires more time to perform the same activity than a senior employee.

Agent behavior

The behavior $a.b$ of an agent a is used to determine how the execution of a case continues after agent a has just performed an activity for it, which involves determining the next activity and which agent should perform that activity. To capture such behavior, we provide various configuration options, which differ along two dimensions:

1. **Overall setting.** AgentSimulator offers the possibility to discover and simulate process behavior for orchestrated and autonomous settings, following two high-level ways in which process participants can behave in processes.
   - *Orchestrated behavior* refers to agent behavior that is determined based on *global* control-flow patterns only, independent of specific agents. Thus, the selection of the next activity solely depends on the previous activities, whereas the agent to perform that activity is simply assigned based on capabilities and availability. This configuration option is useful when simulating processes whose execution is centrally guided, such as those supported by a workflow or business process management system (Dumas et al. 2013). In these processes, the system generally determines which activity should be performed next and who it is assigned to (or users pull activities).
   - *Autonomous behavior* refers to agent behavior that is determined based on *local* control-flow and handover patterns, which depend on specific agents. Here, the selection of the next activity depends on both the previous activities and the previous agent. Also, the selection of the next agent is determined by the previous agent, acknowledging the autonomy of human resources in choosing who they work with. This configuration option is particularly useful for capturing processes that provide high flexibility and decision power to the human actors involved, e.g., knowledge-intensive processes.
2. **Learning method.** We propose three different methods for representing and learning agent behavior, where each method can be adapted to orchestrated and autonomous behavior. These methods jointly cover three main families through which process behavior can be captured:
   - *Frequentist probabilities* provide a purely probabilistic view on agent behavior, where the choice for the next activity (or agent) is determined based on historical probabilities.
   - *LSTM networks* are representative of black-box deep learning techniques commonly used in predictive process monitoring (Tax et al. 2017) and simulation (Camargo et al. 2019) to determine transitions and handovers.

- *Petri nets* provide symbolic means to explicitly capture control-flow logic in processes.

Table 1 provides an overview of how these options are combined into six configurations for AgentSimulator, which we next describe in detail. Note that only the Petri net-based approach explicitly models concurrency, which is a limitation of the other approaches. Furthermore, note that in orchestrated settings, the method that learns agent behavior only needs to provide an explicit representation to determine the next activity, as the next agent selection is determined during simulation based on capabilities and availabilities. On the contrary, in autonomous settings, the method needs to provide an explicit representation for both next activity and agent.

Frequentist probabilities. The simplest method for learning agent behavior in AgentSimulator is purely probabilistic. It determines the next activity and agent based on the observed frequencies of activity transitions and agent handovers in the event log L. Here, *activity transition* refers to the occurrence of one activity directly followed by another activity within a trace, whereas *agent handover* describes when one agent is directly followed by another agent within a trace. The details of the method are introduced first for orchestrated behavior, and afterwards for autonomous behavior.

a) Orchestrated. To determine the next activity in orchestrated settings using frequentist probabilities, we compute activity transition probabilities at the global log level. Specifically, we compute the frequentist probability of transitioning to an activity given the activity prefix of an ongoing case, where a prefix $\sigma_{\text{prefix}} = \langle e_1, e_2, ..., e_k \rangle$ is the sequence of events from the beginning of σ up to event e_k, with $\sigma^{act}_{\text{prefix}}$ being the corresponding activity sequence. This transition probability $P(act|\sigma^{act}_{\text{prefix}})$ is computed by determining how often each possible activity prefix $\sigma^{act}_{\text{prefix}}$ in log L is followed by each activity $act \in ACT_L$. This is achieved by dividing the number of times the specific transition from $\sigma^{act}_{\text{prefix}}$ to act happens by the number of times the activity prefix

Table 1 Overview of methods for learning agent behavior in AgentSimulator

		Frequentist prob.	LSTM network	Petri net
Orchestrated	**Activity**	Observed relative frequency of transitioning from an activity prefix to any other next activity	LSTM trained to predict next activity based on activity prefix	Inductive Miner-based Petri net with branching probabilities
	Agent	The earliest available agent capable of executing the selected activity	— ″—	— ″—
Autonomous	**Activity**	Observed relative frequency of transitioning from an activity prefix (with the last activity performed by a specific agent) to any other next activity	LSTM trained to predict next activity based on activity and agent prefixes	Agent Miner-based agent net with branching probabilities
	Agent	Observed relative frequency of one agent performing a specific activity and handing over to any other next agent	LSTM trained to predict next agent based on activity and agent prefixes	Agent Miner-based interaction net with branching probabilities

occurs in the log. Note that, in case an activity prefix $\sigma^{act}_{\text{prefix}}$ has not been observed in the log, we iteratively remove the first activity of the prefix until we reach a subsequence that has been observed in L. For example, if we have not seen $\langle a, b, c, d\rangle$, we next check for occurrences of $\langle b, c, d\rangle$. To transition to the end of a case, we introduce a placeholder end event following the last activity in each case.

b) Autonomous. In autonomous settings, both next activity and agent need to be explicitly modeled from the local agent perspective. Regarding the activity, we adapt the computation of the probabilities described before for orchestrated behavior and make it agent-specific. Thus, $P(act|\sigma^{act}_{\text{prefix}}, a)$ is computed by counting for each agent $a \in A$ the number of transitions from $\sigma^{act}_{\text{prefix}}$ to act with the last activity of the prefix being performed by a, divided by the total number of occurrences of $\sigma^{act}_{\text{prefix}}$ with its last activity being executed by a. Consider Oliver in Fig. 1 who, given the prefix *application received*, always first checks the income sources before the credit history, whereas Steve and Angela proceed the other way round.

To determine the next agent, we compute the frequentist probability of handing over a case from one agent to another agent, specifically for each activity. The conditional probability $P(a_i|a_j, act)$ of handing over from agent a_j to a_i, $i, j \in \{1, ..., |A|\}$ with a_j having performed activity act, is computed by counting all occurrences where the specific activity act is performed by a_j and the following activity is executed by a_i, divided by the total number of times act is performed by a_j. Following our example, this results in $P(\text{Maria}|\text{Angela}, \text{Check income sources}) = 0.0$ and $P(\text{Patrick}|\text{Angela}, \text{Check income sources}) = 1.0$.

The combination of $P(act|\sigma^{act}_{\text{prefix}}, a)$ and $P(a_i|a_j, act)$ defines agent behavior for autonomous settings in a fully probabilistic way and captures agent-specifics that influence the progress of a case.

LSTM network. LSTM models belong to the most popular methods in the field of predictive process monitoring, aiming to solve tasks such as next activity prediction, next time prediction, or remaining time prediction. Also, LSTM models are used in some state-of-the-art BPS approaches (Camargo et al. 2019, 2022; Meneghello et al. 2023). In particular, they demonstrate strong performance in learning the control-flow of a process in the context of BPS, as recently shown by Meneghello et al. (2025). Due to their performance and popularity, we also integrate LSTM networks and explain how they can handle next activity prediction for both orchestrated and autonomous behaviors, as well as next agent prediction for autonomous behavior.

a) Orchestrated. Orchestrated behavior in the context of an LSTM model can be captured by training the model on the global control-flow of the process, thus predicting the next activity based on the prefix of activities. Similar to the model proposed by Camargo et al. (2019), our LSTM uses n-grams to represent the prefix of activities in a case. Given an activity prefix, the model takes this sequence alone and encodes it through an embedding layer to represent categorical activities in a continuous space. An LSTM layer then processes the sequence, capturing dependencies and patterns within the activities over time, and a dense output layer with softmax activation computes a distribution over the next activities.

The idea is very similar to the fully probabilistic method, as both determine the next activity based on the prefix of activities. However, the LSTM goes one step further by trying to learn underlying patterns in the control-flow.

b) Autonomous. To adapt the LSTM for autonomous behavior learning, which includes predicting both the next activity and the next agent, we extend the orchestrated method by incorporating prefixes of both activities and their associated agents as input. This design captures agent-specific patterns in process execution. The model uses two input embeddings—one for activities and one for agents—allowing it to encode activity-agent pairs in parallel. These embeddings are concatenated and processed together by the LSTM layer, enabling the model to learn patterns influenced by agent assignments.

The model is trained in a multi-task learning framework, predicting both the next activity and the next agent within a single model. The activity output layer predicts a probability distribution over possible next activities, while the agent output layer predicts a distribution over potential next agents.

Petri net. Petri nets are one of the most common representations of business processes. Various algorithms have been proposed for automatically discovering Petri net models from an event log (Augusto et al. 2019; Leemans et al. 2013). Most commonly, Petri nets only describe the global control-flow perspective, making them a suitable approach for representing orchestrated behavior. However, a more recent discovery algorithm (Tour et al. 2023) also considers the agent perspective, resulting in Petri nets that represent the local control-flow as well as agent handovers. Thus, the latter can be used to obtain a Petri net-based representation of autonomous behavior in AgentSimulator. Again, the details of the method are introduced first for orchestrated behavior, and afterwards for autonomous behavior.

a) Orchestrated. For modeling orchestrated behavior with Petri nets, we use the Inductive Miner algorithm (Leemans et al. 2013) to discover a Petri net from the input event log L, extended with a stochastic map to represent branching probabilities at decision points. To compute a stochastic map for a given Petri net, we employ token-based replay of the log on the Petri net model. During replay, each event in the trace is matched with a corresponding Petri net transition $\tau \in \mathcal{T}$, simulating the progression of tokens through the net from an initial to a final marking. The replay enables us to assess how often each transition τ is enabled, denoted as $C_{\text{enabled}}(\tau)$ versus how often it is actually fired, denoted as $C_{\text{fired}}(\tau)$. Thus, for each decision point in the Petri net, the branching probability of choosing that path is computed as $\frac{C_{\text{fired}}(\tau)}{C_{\text{enabled}}(\tau)}$. The resulting model captures the global control-flow of the process, determining at each step which activities are allowed to execute next and the likelihood of each.

b) Autonomous. For modeling autonomous behavior with Petri nets, we employ the Agent Miner algorithm (Tour et al. 2023)—an approach for discovering Petri net models of agents and their interactions from event data. The Agent Miner algorithm comprises three main steps, i.e., the construction of i) agent nets, ii) an interaction net, and iii) an MAS net, to model business processes with an agent-centric approach.

i)Agent Nets: The algorithm first constructs an individual model, or agent net, for each identified group of agents, where agents that perform the similar activities belong to the same group. This net captures the internal workflow of each group, detailing its specific local control-flow.

ii) Interaction Net: The interaction net is then developed to map the interactions between agent groups. This step highlights points where agent groups collaborate, allowing the algorithm to model the handover paths that define the multi-agent interactions within the process.

iii)MAS Net: Finally, the individual agent nets and the interaction net are integrated to form a comprehensive MAS net, representing the entire process.

To integrate the Agent Miner algorithm into AgentSimulator, we use the discovered interaction net to identify the subsequent agent group and the corresponding agent net to determine the next activity. Based on this selected agent group and activity, we specify the next agent as we do in the frequentist method. In alignment with our approach of augmenting the Inductive Miner-derived Petri net with a stochastic map, we enhance the interaction and agent nets by incorporating branching probabilities.

General simulation parameters

After discovering the set of agents A, we discover some general simulation parameters $\mathcal{M}.P = (f_{iat}, AC, D)$, where f_{iat} is a PDF of case inter-arrival times, AC denotes the arrival calendar, and D a set of PDFs over extraneous delays. These parameters are not specific to an agent-based simulation model but are also used in other BPS approaches (Camargo et al. 2020).

Inter-arrival times and arrival calendar. Inter-arrival times denote the duration between the start of two consecutive cases. The PDF over inter-arrival times is used during simulation to sample new cases. For discovering the PDF over case inter-arrival times f_{iat}, we follow Camargo et al. (2020) who fit different distributions to the inter-arrival times and take the one that yields the smallest earth mover's distance. To ensure that cases can only start during the organization's working hours, we additionally discover an arrival calendar AC, containing a set of intervals that describe the working hours. Thus, during the simulation, samples from f_{iat} will only create new case arrival timestamps within intervals of AC.

Extraneous delays. Since different instances of a process typically compete for limited resources and because resources do not always start an activity as soon as it can possibly be performed, processes are affected by waiting times. Extraneous delays are waiting times that are not caused by resource contention or unavailability (e.g., a resource waits for a response from the customer). They need to be modeled explicitly. Following the algorithm in Chapela-Campa and Dumas (2024), we discover a PDF over extraneous delays per activity, resulting in a set of extraneous delay distributions D.

Output of phase 1

Having discovered the MAS, consisting of both the set of agents A and the general simulation parameters P, we have created a fine-granular and interpretable model of the underlying process. This model explicitly represents each process participant (agent) along with their schedules, capabilities, and behaviors, providing a white-box view of the system. The interpretability of the model stems from its explicit representation, which allows practitioners to modify key aspects of the simulation. For example:

- Agents can be added or removed from the process to explore changes in resource staffing levels.
- The availability of individual agents (*a.s*) can be adjusted to simulate different working time configurations, such as shift changes or increased availability.
- Agent capabilities can be modified by adding or removing activities from *a.c.* Alloc or altering execution times in *a.c.PT*, allowing for the simulation of upskilling or restructuring.
- Handovers between agents can be adjusted by modifying handover probabilities in *a.b*, enabling the exploration of alternative collaboration structures.
- Changes to the activity transition probabilities in *a.b* allow for the simulation of alternative process control-flow scenarios.

While this paper does not include specific what-if scenarios, the fine-granular nature of the model positions AgentSimulator as a versatile tool for such analyses in future studies. The ability to modify these elements in a targeted manner makes the model suitable for scenario testing, optimization, and decision support. However, note that a key limitation for now compared to process model-based approaches like Simod (Camargo et al. 2020) is the lack of a straightforward method for visualizing the discovered MAS.

Algorithm 1 Simulation

```
    Input: $A$, $f_{iat}$, $AC$, $D$, $startAt$, Num. of cases $n$
    Output: Simulated event log $L'$
1:  $\Sigma \leftarrow generate\_arrival\_events(f_{iat}, AC, n, startAt)$, $L' \leftarrow \{\}$
2:  for $a \in A$ do
3:      $a.next\_avail \leftarrow minFrom(a.s, startAt)$
4:  while $|L'| < n$ do
5:      $\sigma_{\text{prefix}} \leftarrow get\_case(\Sigma)$    ▷ get the case with the smallest current timestamp
6:      $e \leftarrow ()$    ▷ initialize new event
7:      $e.act \leftarrow get\_next\_activity(\sigma_{\text{prefix}})$    ▷ retrieve the next activity for the case
8:      $e.a \leftarrow get\_next\_agent(\sigma_{\text{prefix}}, e.act)$    ▷ assign an agent
9:      $e.ts_{start} \leftarrow max(end\_time\_preceding\_event(\sigma_{\text{prefix}}), e.a.next\_avail)$
10:     $e.ts_{start} \leftarrow add\_extr\_delays(e.ts_{start}, D)$
11:     $e.ts_{end} \leftarrow e.ts_{start} + execution\_time(e.act, e.a) + off\_time(e.a)$ ▷ calculate end time
12:     $e.a.next\_avail \leftarrow minFrom(e.a.s, e.ts_{end})$    ▷ update agent availability
13:     $\sigma_{\text{prefix}} \leftarrow add\_to\_prefix(e)$    ▷ extend the prefix with the new event
14:     if $e.act$ = end then    ▷ check if case is completed
15:         $L', \Sigma \leftarrow exit\_and\_add\_to\_log(\sigma_{\text{prefix}})$    ▷ finalize case
```

Phase 2: simulation

This section details how the AgentSimulator approach uses the discovered MAS $\mathcal{M}$ to simulate an event log L', illustrated by the pseudo-code in Algorithm 1. The following procedure, which is visualized on a high level in Fig. 2, applies universally to all three methods for learning agent behavior discussed in previous sections. The simulation begins with an initialization, where *n* arrival events are generated based on the inter-arrival time distribution and the arrival calendar (Line 1). Additionally, each agent's next availability is set based on their predefined schedule *a.s* and the simulation's starting timestamp (Line 3). The simulation proceeds in discrete steps until the specified number of cases n has been completed. Each simulation step is structured as follows:

1) Case selection. At each iteration, the case σ_{prefix} with the earliest current timestamp is selected from the set of active cases Σ (Line 5). This ensures that events are processed in chronological order.

2) Activity determination. Given the prefix of the selected case σ_{prefix}, the next activity to be executed is identified (Line 7). The activity selection is determined using one of the three proposed methods: *Frequentist probabilities*, *LSTM networks*, or *Petri nets*. Depending on the simulation mode, this selection can be performed either globally, modeling orchestrated behavior, or locally, modeling autonomous decision-making. For newly initiated cases, the first activity is always selected globally, as there is no prior agent to influence the decision.

3) Agent assignment. Once the activity has been determined, it must be assigned to an agent for execution. The agent is selected using the function *get_agent* (Line 8), differentiating between the two settings:

- *Orchestrated.* In an MAS with orchestrated handovers, no agent interaction patterns are modeled. Therefore, the agent selection is based on agent capabilities and availabilities. First, based on the set of activities that an agent can execute *a.c.* ALLOC, we identify possibly responsible agents. Second, we order this list based on the agent availabilities and assign the activity to the agent that offers the earliest availability indicated by *a.next_avail*.
- *Autonomous.* In a system with autonomous handovers, agent transitions are explicitly modeled through predefined interaction patterns, defining a probability distribution over possible next agents. Unlike the orchestrated setting, agent selection does not prioritize availability. Instead, an agent is sampled from the distribution, favoring those with a higher probability of receiving the handover.

Note that a case's first agent is always determined as described for orchestrated handovers, as there is no previous agent to impact the agent selection.

4) Event timestamp calculation. The event's start timestamp is determined by the maximum timestamp between the end timestamp of the preceding activity *end_time_preceding_event*(σ_{prefix}) and the timestamp of the agent's next availability *a.next_avail* (Line 9). In case extraneous delays should be considered in the simulation, they can be added to determine the actual start timestamp (Line 10). The event's end timestamp is then determined by sampling its execution time from the agent-specific activity duration distribution $f_{pt}(act) \in a.c.PT$ and potentially adding the time periods in which the agent is not working according to its schedule *a.s*. Thus, we allow simulating interruptions in activity executions.

5) Event finalization. Having determined all required event attributes, the agent availabilities are updated in Line 12 and the event is added to the prefix of the current case (Line 13). Finally, we check if the selected activity represents the process's end activity and if so the case is removed from Σ and added to the output event log L' (Line 15).

Table 2 Description of log properties

Log	Type	#Traces	#Events	#Activities	#Resources	#Agents
Loan Application	syn	1000	7492	12	19	19
P2P	syn	608	9119	21	27	27
CVS	syn	10000	103906	15	6	8
Confidential 1000	syn	1000	38160	42	14	26
Confidential 2000	syn	2000	77418	42	14	26
ACR	real	954	6870	18	432	432
Production	real	225	4503	24	41	41
BPI12W	real	8616	59302	6	52	56
BPI17W	real	30276	240854	8	136	136

The entire simulation concludes when the specified number of cases n has been processed and finished. The output of the simulation is an event log L', where each event is a tuple $e = (act, ts_{start}, ts_{end}, a)$.

Experiments and results

This section details the experiments conducted to comprehensively evaluate the performance of our AgentSimulator approach in realistically simulating business processes. The evaluation comprises three parts: in the first evaluation, we examine the ability of AgentSimulator in its basic configuration—using frequentist probabilities—to faithfully simulate a range of synthetic and real-life processes and thereby automatically determine per process if the orchestrated or autonomous setting is more suitable. We compare our results to various state-of-the-art BPS approaches. In the second evaluation section, we investigate the impact of the various agent behavior learning methods outlined in Table 1, also identifying to what extent the choice between the orchestrated or autonomous setting is critical for different process types. Finally, in the third evaluation, we evaluate the impact of orchestrated and autonomous settings for the simulation of a synthetic event log that is subject to specific resource characteristics and interaction patterns. Table 2 summarizes the characteristics of the 9 publicly available event logs used in the first two experiments, which are commonly used in BPS evaluations (Camargo et al. 2022; Meneghello et al. 2023; Chapela-Campa et al. 2023) as they contain both start and end timestamps. Note that 6 of the 9 event logs contain concurrent activities. Our implementation, the event logs (with train-test splits), and additional results are available through our repository[1].

Evaluation 1: simulation performance

In this section, we evaluate AgentSimulator against state-of-the-art BPS approaches in accurately simulating different processes, both synthetic and real-world. This evaluation considers *Frequentist Probabilities* for learning agent behavior, whereas AgentSimulator needs to automatically determine and adapt to the underlying process type, whether

[1] https://github.com/lukaskirchdorfer/AgentSimulator

orchestrated or autonomous. In the following, we describe the experimental setup and discuss the obtained results.

Experimental setup

Implementation. We implemented our approach in Python using the agent-based modeling framework *mesa* (Kazil et al. 2020).

Benchmark approaches. We empirically compare our AgentSimulator (AgentSim) against four common data-driven BPS approaches. We adopt the state-of-the-art Data-driven Process Simulation (DDPS) approach from López-Pintado and Dumas (2022), which we refer to as Simod. It improves the original Simod from Camargo et al. (2020) by considering differentiated resource availability and performance and was shown to outperform the original Simod. DeepGenerator (DGEN) (Camargo et al. 2019) is a pure Deep Learning (DL) approach, whereas DeepSimulator (DSIM) (Camargo et al. 2022) and RIMS (Meneghello et al. 2023) are both a hybrid of DDPS and DL. RIMS extends upon DSIM by integrating the predictions of the DL model at runtime during the simulation (more details in the Related work section). Note that we use the vanilla RIMS approach, not any adaptation of it as presented in Meneghello et al. (2025).

Data split. We follow evaluations of existing BPS approaches (Camargo et al. 2019, 2022; Meneghello et al. 2023) and perform a temporal hold-out split, excluding all cases that span the separation time between the train set (first 80% of cases) and the test set (last 20% of cases).

Hyperparameters. In this first experiment, we let AgentSim have two automatically determined hyperparameters: the setting for learning agent behavior (orchestrated or autonomous) and whether to consider extraneous delays, resulting in 4 possible configurations. We treat the latter as a hyperparameter because we noticed considerable differences regarding the benefit of considering extraneous delays between different event logs. We determine both hyperparameters by initially simulating the last 20% of the training set for each of the 4 possible configurations and checking which simulation most closely resembles the training subset in terms of cycle time. To ensure a fair comparison, we use the same option regarding extraneous delays for Simod. Generally, for the benchmark approaches, we tune all respective hyperparameters, trying to provide a fair comparison.

Metrics. To evaluate and compare the different simulation approaches, we use multiple metrics that are designed to evaluate simulation models across three dimensions: control-flow, time, and congestion (Chapela-Campa et al., 2023)). Each metric calculates a distance between the simulated and test logs, where lower values indicate better performance. To measure control-flow, we use the N-Gram Distance (NGD), which compares the frequencies of activity n-grams observed in the event logs. In this work, we set $n = 3$. The remaining 4 metrics are all based on the 1st Wasserstein Distance (1WD), which is a computationally efficient variation of the Earth Mover's Distance (EMD), a measure that quantifies the dissimilarity between two probability distributions. 1WD intuitively represents the minimum *effort* required to transform one distribution into another by redistributing their *mass*. The time dimension is evaluated by 3 metrics: The Absolute Event Distribution Distance (AEDD) measures differences in the absolute timestamps of events, capturing how events are distributed over time. The Relative

Event Distribution Distance (REDD) evaluates the distribution of events relative to the arrival timestamp of their respective case, capturing how cases progress over time. The Circadian Event Distribution Distance (CEDD) focuses on weekly activity patterns, identifying differences in how events are distributed across days of the week. Finally, we evaluate congestion using the Cycle Time Distribution Distance (CTDD), which measures how accurately the simulated cycle times of cases match those in the test log. We do not include the Case Arrival Distribution distance as a metric in our analysis, as the same arrival method as Simod is applied, allowing us to concentrate on metrics where AgentSim exhibits distinct performance.

Results

Overall results. Tables 3 and 4 summarize the results for the 9 event logs, showing the average metrics from 10 simulation runs as well as the standard deviation. The best value per log and metric is marked in bold. For this, we apply a paired t-test with a threshold for the *p-value* of 0.05 to determine statistical significance. Generally, no single approach consistently outperforms the others across all datasets and metrics as also observed in previous BPS works (Meneghello et al. 2023; Camargo et al. 2022). However, AgentSim and DSIM stand out by most often achieving the best performance. AgentSim simulates

Table 3 Comparison of simulation approaches across various performance metrics. The results for AgentSim are based on the *Frequentist probabilities* method for learning agent behavior

Metric	Method	Event Log				
		Loan	P2P	CVS	C.1000	C.2000
NGD	Simod	0.15 (0.02)	0.42 (0.01)	0.44 (0.00)	0.25 (0.01)	0.25 (0.00)
	DGEN	0.21 (0.01)	**0.20 (0.02)**	0.22 (0.00)	0.58 (0.00)	**0.16 (0.01)**
	DSIM	n/a	0.22 (0.01)	0.20 (0.00)	**0.20 (0.01)**	0.18 (0.01)
	RIMS	n/a	0.22 (0.01)	0.44 (0.00)	0.25 (0.01)	0.28 (0.01)
	AgentSim	**0.07 (0.01)**	0.24 (0.02)	**0.10 (0.01)**	0.25 (0.01)	0.25 (0.01)
AEDD	Simod	13.55 (1.84)	**1044.25 (57.19)**	52.95 (1.7)	344.48 (21.05)	820.45 (48.14)
	DGEN	212.27 (0.01)	1481.46 (2.12)	310.39 (0.01)	462.84 (0.11)	857.68 (0.28)
	DSIM	n/a	1310.03 (14.22)	**36.23 (2.94)**	246.41 (8.50)	591.13 (5.50)
	RIMS	n/a	1266.30 (11.24)	58.73 (1.72)	242.80 (9.34)	620.61 (5.44)
	AgentSim	**2.97 (0.77)**	1214.54 (12.51)	93.43 (1.44)	**106.51 (11.43)**	**221.48 (9.56)**
CEDD	Simod	0.40 (0.04)	2.21 (0.18)	**0.44 (0.06)**	3.01 (0.14)	**2.96 (0.06)**
	DGEN	13.40 (0.00)	2.55 (0.08)	11.69 (0.01)	18.93 (0.05)	18.09 (0.49)
	DSIM	n/a	1.16 (0.10)	8.98 (0.05)	**2.28 (0.30)**	**2.84 (0.14)**
	RIMS	n/a	**0.83 (0.12)**	8.88 (0.04)	**2.10 (0.3)**	**2.82 (0.14)**
	AgentSim	**0.27 (0.04)**	1.22 (0.13)	7.63 (0.04)	3.13 (0.95)	**2.82 (0.92)**
REDD	Simod	9.22 (1.67)	840.19 (1.23)	39.43 (3.53)	468.81 (33.93)	952.37 (85.53)
	DGEN	5.26 (0.01)	828.09 (2.07)	176.65 (0.01)	8.11 (0.10)	4.58 (0.23)
	DSIM	n/a	**722.33 (6.33)**	**19.74 (0.30)**	**5.34 (0.33)**	**1.7 (0.31)**
	RIMS	n/a	**727.77 (6.4)**	40.05 (0.75)	**5.06 (0.86)**	**2.29 (0.82)**
	AgentSim	**1.66 (0.72)**	**725.37 (11.77)**	87.24 (1.39)	16.70 (7.70)	31.74 (9.26)
CTDD	Simod	20.42 (2.78)	677.05 (1.78)	54.59 (4.12)	804.07 (56.60)	1614.91 (135.64)
	DGEN	9.38 (0.01)	670.05 (4.14)	294.21 (0.07)	13.92 (0.11)	8.12 (0.25)
	DSIM	n/a	**566.63 (8.13)**	52.43 (0.74)	**7.29 (0.31)**	**2.26 (0.43)**
	RIMS	n/a	581.41 (13.85)	**28.26 (1.24)**	**7.39 (0.95)**	**2.33 (0.68)**
	AgentSim	**2.71 (0.99)**	**558.36 (8.54)**	107.49 (2.25)	28.13 (13.08)	52.42 (13.10)

Table 4 Comparison of simulation approaches across various performance metrics. The results for AgentSim are based on the *Frequentist probabilities* method for learning agent behavior

Metric	Method	Event Log			
		ACR	**Prod**	**BPI12W**	**BPI17W**
NGD	Simod	**0.24 (0.02)**	0.93 (0.01)	0.72 (0.00)	0.59 (0.00)
	DGEN	0.31 (0.03)	**0.52 (0.03)**	0.43 (0.01)	0.67 (0.00)
	DSIM	**0.26 (0.02)**	0.86 (0.01)	0.65 (0.01)	0.54 (0.01)
	RIMS	**0.26 (0.02)**	0.87 (0.01)	0.56 (0.00)	0.69 (0.00)
	AgentSim	0.36 (0.01)	0.59 (0.03)	**0.15 (0.01)**	**0.19 (0.00)**
AEDD	Simod	287.27 (29.41)	146.38 (86.72)	**71.97 (9.78)**	300.28 (9.39)
	DGEN	559.67 (0.18)	224.45 (10.54)	306.28 (0.93)	4557.19 (123.24)
	DSIM	273.46 (8.71)	154.31 (9.47)	**78.62 (6.76)**	**54.61 (4.04)**
	RIMS	**241.95 (9.01)**	132.82 (12.98)	**73.82 (7.05)**	122.32 (11.53)
	AgentSim	328.71 (1.93)	**61.13 (8.60)**	115.45 (9.30)	218.43 (2.58)
CEDD	Simod	**2.60 (0.10)**	**2.82 (0.18)**	**1.71 (0.09)**	3.34 (0.02)
	DGEN	17.84 (0.76)	9.30 (3.42)	4.53 (0.22)	3.39 (0.01)
	DSIM	4.64 (0.24)	**2.66 (0.17)**	2.88 (0.05)	3.35 (0.08)
	RIMS	3.07 (0.08)	**2.73 (0.23)**	3.01 (0.14)	3.72 (0.15)
	AgentSim	7.61 (0.22)	5.70 (0.32)	1.85 (0.04)	**2.39 (0.02)**
REDD	Simod	32.46 (0.82)	83.88 (2.53)	95.72 (1.69)	136.63 (0.89)
	DGEN	30.87 (0.15)	70.11 (10.48)	116.18 (0.80)	118.84 (0.24)
	DSIM	**15.62 (2.79)**	33.30 (8.04)	119.12 (1.02)	**33.10 (2.57)**
	RIMS	**18.55 (8.67)**	**18.69 (5.41)**	99.12 (1.09)	54.43 (1.54)
	AgentSim	**22.91 (0.62)**	31.72 (6.58)	**54.85 (6.13)**	40.69 (1.27)
CTDD	Simod	93.51 (93.51)	89.15 (5.52)	155.46 (1.74)	148.40 (1.77)
	DGEN	95.11 (0.24)	90.82	176.79 (0.73)	172.94 (1.46)
	DSIM	**48.24 (4.14)**	43.26 (8.67)	173.49 (1.03)	**30.26 (2.44)**
	RIMS	**41.58 (10.30)**	**24.71 (7.94)**	150.93 (1.67)	108.66 (1.58)
	AgentSim	62.48 (2.00)	**29.95 (7.96)**	**89.07 (3.72)**	41.06 (2.18)

four of these logs (P2P, C.1000, C.2000, ACR) using the autonomous setting, while the remaining five are simulated using the orchestrated setting. When looking more closely at the 3 dimensions captured by the metrics, we can obtain the following main insights:

Control-flow. Achieving the best NGD (N-Gram Distance) in 4 out of 9 logs and considerably outperforming Simod, DSIM, and RIMS, the results indicate that our resource-first AgentSim approach, which is—with the frequentist method—independent of an underlying process model, enhances the accuracy of the control-flow dimension compared to control-flow-first approaches such as Simod. This is particularly evident when analyzing the two real-life BPI logs. Thus, putting the resource at the core of the simulation often delivers more realistic control-flow patterns.

Time. In terms of absolute (AEDD), circadian (CEDD), and relative (REDD) event distributions over time, we observe mixed results with AgentSim mostly leading in AEDD, Simod mostly leading in CEDD, and excelling in REDD. These results indicate that AgentSim captures the temporal patterns in the log comparatively well.

Congestion. Accurately capturing the cycle time of a process instance serves as a critical indicator of the accuracy of a simulation approach. The CTDD metric is influenced by the processing times of individual activities and the corresponding waiting times, which in turn are dependent on resource availability. Consequently, the cycle time

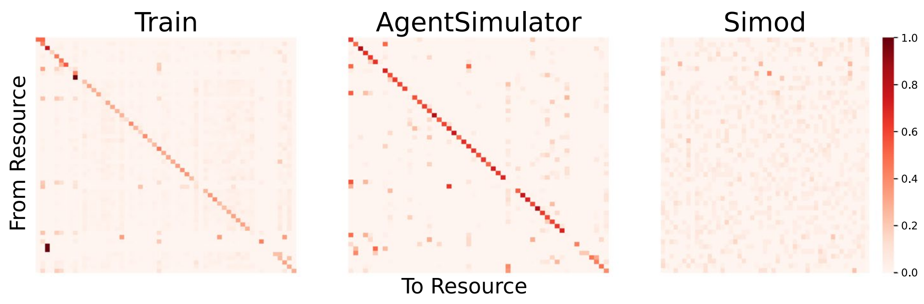


Fig. 3 Resource interactions for the BPI12W log

Table 5 Average runtimes (with standard deviation) across all event logs. For AgentSim, we measured the runtimes using the frequentist method for learning agent behavior

	Simod	DGEN	DSIM	AgentSim
Discovery (minutes)	9.7 (4.6)	10.5 (12.0)	61.1 (89.1)	4.3 (5.9)
Simulation (seconds)	3.5 (1.2)	153.0 (294.3)	718.1 (597.3)	10.8 (24.0)

metric captures a comprehensive range of factors. Here, AgentSim, DSIM, and RIMS clearly outperform Simod and DGEN. The advantage of AgentSim over Simod is particularly pronounced with the Confidential logs. Here, Simod's simulated logs exhibit significantly longer cycle times compared to reality, primarily due to resource contention and the consequent waiting times. AgentSim mitigates this issue by recognizing that some agents do not require waiting time, such as agents performing instantaneous activities.

Post-hoc analysis on agent interactions. To illustrate AgentSim's capability to model agent interactions, we compare its interaction patterns to those of Simod using the real-life BPI12W log. As shown in Fig. 3, AgentSim effectively replicates the interaction dynamics of the training log, accurately capturing the given activity chaining of resources (visible along the diagonal). In contrast, Simod demonstrates random interactions, failing to represent these patterns. This highlights the importance of considering individual interaction patterns in certain processes. It is worth noting that other benchmark approaches are excluded from this comparison, as they simulate only higher-level roles rather than individual resources.

Runtime. Given the inherent stochasticity of simulation, simulating numerous logs is crucial for reliable predictions, making runtime a key factor in the practical use of BPS. Despite this, detailed runtime comparisons of existing data-driven BPS approaches have not yet been made in the literature. To address this gap, we evaluated the runtimes of Simod, DGEN, DSIM, and AgentSim on a machine with 18GB RAM and an Apple M3 Pro 12-core CPU. The results, summarized in Table 5, represent the time required to simulate a single event log. It is important to note that all the evaluated approaches are research prototypes and were not specifically optimized for runtime performance. Nevertheless, the observed runtime magnitudes provide a general indication of their computational efficiency. Unsurprisingly, the Deep Learning-based methods, DGEN and DSIM, demonstrated significantly longer runtimes compared to Simod and AgentSimulator.

Evaluation 2: assessment of AgentSimulator configurations

In this section, we assess the impact of the different methods for learning agent behavior presented in Table 1, consisting of *Frequentist probabilities*, *LSTM networks*, and *Petri nets*, each in its orchestrated and autonomous setting. In the remainder, we first describe the experimental setup and then discuss the obtained results. We use the same event logs and data splits as used in the first evaluation.

Experimental setup

Implementation. We integrated both the Petri net-based and LSTM-based methods into our Python framework. For discovering Petri nets with Inductive Miner, we use the pm4py library (Berti et al. 2023). The Agent Miner Petri nets are discovered based on the implementation provided by Tour et al. (2023). The LSTM networks are implemented using the TensorFlow framework.

Benchmark approaches. We benchmark six different configurations of AgentSimulator. Each of the three agent behavior methods is evaluated in its orchestrated and autonomous setting.

Hyperparameters. For all configurations, we fix the extraneous delays hyperparameter for each log, as determined in the first evaluation, to ensure a controlled comparison between the orchestrated and autonomous settings. The two Petri net-based configurations rely solely on the default settings of their respective discovery algorithms without additional hyperparameter tuning. We train both LSTM networks for 20 epochs with the Adam optimizer, a learning rate of 0.001, and a batch size of 32. Note that we do not tune the LSTM hyperparameters, as initial tuning experiments showed no performance improvement. We attribute this to the simplicity of the data, as the model is trained solely on sequences of activities and resources, without incorporating additional case or event attributes.

Metrics. In this experiment, we want to focus our evaluation on only two of the five metrics we used for the first evaluation. Specifically, we want to investigate the impact of the different configurations on the control-flow and the cycle time—the two process characteristics that are potentially most influenced by different agent behaviors—represented by the NGD and CTDD metrics. For completeness, the results for the other metrics can be found in our public repository.

Results

Tables 6 and 7 present the experimental results for the 9 event logs, summarizing the average outcomes across 10 simulation runs. It highlights the differences in control-flow and cycle time accuracy across the six configurations of learning agent behavior in AgentSimulator. We begin by comparing the general performance of the three methods—*Frequentist probabilities*, *LSTM networks*, and *Petri nets*—before analyzing the impact of orchestrated versus autonomous settings.

Comparison of methods for learning agent behavior. The choice of the learning method significantly impacts both control-flow and cycle time accuracy. Remarkably, the simple frequentist method outperforms others in control-flow accuracy, achieving the best or second-best NGD in all 9 logs. While the LSTM methods consistently

Table 6 Comparison of *Frequentist probabilities* (FP), *LSTM network* (LS), and *Petri net* (PN) for learning agent behavior in AgentSimulator for orchestrated (Orch) and autonomous (Auto) settings for synthetic event logs. The best value per metric and event log according to a paired t-test is marked in bold

Metric	Method	Event Log				
		Loan	P2P	CVS	C.1000	C.2000
NGD	FP Orch	**0.07 (0.01)**	**0.25 (0.04)**	0.10 (0.01)	**0.24 (0.01)**	**0.24 (0.01)**
	FP Auto	**0.07 (0.01)**	**0.24 (0.02)**	**0.09 (0.00)**	0.25 (0.01)	**0.24 (0.01)**
	LS Orch	**0.08 (0.01)**	0.31 (0.02)	0.10 (0.00)	**0.24 (0.01)**	0.29 (0.01)
	LS Auto	0.14 (0.02)	0.30 (0.01)	0.11 (0.00)	0.29 (0.01)	0.28 (0.01)
	PN Orch	**0.09 (0.01)**	0.54 (0.02)	0.28 (0.00)	0.56 (0.01)	0.55 (0.01)
	PN Auto	0.65 (0.01)	0.89 (0.01)	0.85 (0.00)	0.80 (0.00)	0.82 (0.01)
CTDD	FP Orch	**2.71 (0.99)**	566.29 (11.56)	**107.49 (2.25)**	46.37 (21.13)	52.76 (8.49)
	FP Auto	3.75 (0.80)	558.36 (8.54)	120.11 (2.24)	28.13 (13.08)	52.42 (13.10)
	LS Orch	2.89 (0.87)	574.82 (6.28)	119.24 (1.99)	41.34 (14.38)	26.19 (8.92)
	LS Auto	3.46 (0.98)	559.33 (6.26)	113.82 (2.23)	42.48 (17.14)	78.79 (26.71)
	PN Orch	3.10 (1.47)	**490.12 (14.37)**	110.82 (1.73)	21.56 (2.21)	20.23 (1.91)
	PN Auto	3.03 (0.51)	664.96 (7.96)	196.82 (2.62)	**7.56 (0.64)**	**13.74 (1.21)**

Table 7 Comparison of *Frequentist probabilities* (FP), *LSTM network* (LS), and *Petri net* (PN) for learning agent behavior in AgentSimulator for orchestrated (Orch) and autonomous (Auto) settings for real-life event logs. The best value per metric and event log according to a paired t-test is marked in bold

Metric	Method	Event Log			
		ACR	Prod	BPI12W	BPI17W
NGD	FP Orch	0.39 (0.02)	**0.60 (0.03)**	**0.15 (0.01)**	0.19 (0.00)
	FP Auto	**0.36 (0.01)**	0.68 (0.07)	**0.16 (0.01)**	0.22 (0.00)
	LS Orch	0.40 (0.02)	**0.57 (0.02)**	0.17 (0.01)	0.19 (0.00)
	LS Auto	0.47 (0.03)	0.83 (0.02)	0.27 (0.02)	**0.18 (0.00)**
	PN Orch	0.75 (0.02)	0.85 (0.02)	0.26 (0.01)	0.35 (0.00)
	PN Auto	0.76 (0.01)	0.73 (0.02)	0.30 (0.01)	0.23 (0.00)
CTDD	FP Orch	69.67 (1.00)	**29.95 (7.96)**	89.07 (3.72)	41.06 (2.18)
	FP Auto	**62.48 (2.00)**	**38.19 (17.16)**	96.79 (3.27)	44.65 (1.97)
	LS Orch	71.68 (2.10)	**24.80 (9.47)**	**72.98 (4.12)**	34.83 (1.43)
	LS Auto	67.89 (2.21)	**20.03 (7.29)**	114.26 (11.52)	48.34 (2.34)
	PN Orch	76.80 (1.81)	2121.32 (238.43)	92.72 (4.72)	56.76 (2.67)
	PN Auto	74.64 (1.20)	64.49 (4.08)	115.24 (2.89)	**28.77 (1.91)**

perform only slightly worse, Petri nets fall substantially behind, often exhibiting 2 to 3 times higher NGD values. In terms of the cycle time, the results are more mixed among the three methods, with Petri nets and the frequentist approaches being slightly better than LSTMs. Overall, the *Frequentist probabilities* method seems to provide an accurate and scalable alternative for learning agent behavior compared to the other two methods, which currently are state-of-the-art in process simulation. While *Petri nets* nets seem significantly worse at capturing the control-flow, *LSTM networks* achieve comparable results but require training a separate black-box model for each log, raising efficiency and interpretability concerns compared to the simpler, statistics-based frequentist method.

Impact of orchestrated vs. autonomous setting. While the choice of learning method has a more significant overall impact than the orchestration setting, the observed differences between orchestrated and autonomous settings highlight the importance of considering this process perspective. The influence of the setting on performance varies depending on both the selected method and the characteristics of the simulated process.

For instance, the frequentist method and LSTM networks generally yield comparable results across orchestrated and autonomous settings, with notable exceptions such as the Production and C.1000 logs. In the Production log, the orchestrated setting aligns well with the structured nature of production processes, resulting in superior performance. Conversely, in the C.1000 log, the autonomous setting offers distinct advantages: the frequentist method achieves a CTDD of 28.13, nearly halving the error observed in the orchestrated setting. For Petri nets, the setting exerts a more pronounced impact. Orchestrated settings often yield better results across both metrics. However, the autonomous setting occasionally excels for some logs such as BPI17W, where it achieves the best-in-class CTDD.

Overall, and particularly for the frequentist and LSTM-based approaches, the impact of the agent behavior setting remains relatively minor. This limitation primarily stems from the restricted availability of event logs suitable for simulation. Only nine publicly available event logs contain both start and end timestamps for each activity, and most of these logs either originate from synthetically generated orchestrated process models or represent real-world orchestrated workflows, where resource-level decision-making plays only a minor role in process execution. To address this gap, our final evaluation examines agent behavior settings in a more decentralized process environment.

Evaluation 3: comparison of orchestrated and autonomous behavior in synthetic process

To provide a more controlled scenario for assessing the impact of modeling autonomous behavior, we generated a synthetic event log representing our motivating credit application process (see Fig. 1). This process involves distinct agents with varying capabilities, availabilities, and specific interaction patterns. The event log contains 1000 cases executed by six agents, with each case consisting of five activities. The event log is publicly available through our repository.

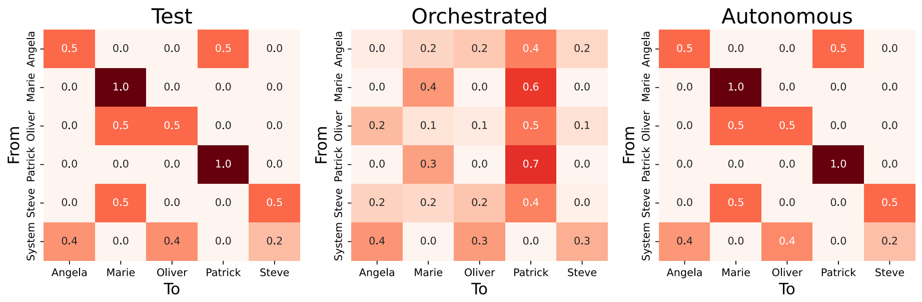
The resources in this process exhibit the following key differences:

- All human resources work from Monday to Friday, 9 a.m. to 5 p.m., except for Maria, who is only available from Monday to Wednesday.
- Oliver follows a different execution order for checking income sources and credit history compared to Steve and Angela.
- Angela exclusively collaborates with Patrick, while Steve and Oliver only collaborate with Maria.

Table 8 presents the results of simulating this process using the frequentist approach of AgentSimulator under both orchestrated and autonomous settings. Across all metrics, the autonomous setting significantly outperforms the orchestrated one, except for the CEDD metric, where both settings perform similarly.

Table 8 Comparison of orchestrated (Orch) and autonomous (Auto) behavior using the frequentist approach of AgentSimulator for the synthetic credit application process. The best value per metric according to a paired t-test is marked in bold

Method	Metrics				
	NGD	AEDD	CEDD	REDD	CTDD
FP Orch	0.07 (0.01)	40.89 (0.52)	**0.32 (0.02)**	14.43 (0.06)	39.11 (0.13)
FP Auto	**0.01 (0.01)**	**14.63 (0.41)**	**0.32 (0.02)**	**6.89 (0.52)**	**20.04 (1.30)**



Test	Angela	Marie	Oliver	Patrick	Steve
Angela	0.5	0.0	0.0	0.5	0.0
Marie	0.0	1.0	0.0	0.0	0.0
Oliver	0.0	0.5	0.5	0.0	0.0
Patrick	0.0	0.0	0.0	1.0	0.0
Steve	0.0	0.5	0.0	0.0	0.5
System	0.4	0.0	0.4	0.0	0.2

Orchestrated	Angela	Marie	Oliver	Patrick	Steve
Angela	0.0	0.2	0.2	0.4	0.2
Marie	0.0	0.4	0.0	0.6	0.0
Oliver	0.2	0.1	0.1	0.5	0.1
Patrick	0.0	0.3	0.0	0.7	0.0
Steve	0.2	0.2	0.2	0.4	0.0
System	0.4	0.0	0.3	0.0	0.3

Autonomous	Angela	Marie	Oliver	Patrick	Steve
Angela	0.5	0.0	0.0	0.5	0.0
Marie	0.0	1.0	0.0	0.0	0.0
Oliver	0.0	0.5	0.5	0.0	0.0
Patrick	0.0	0.0	0.0	1.0	0.0
Steve	0.0	0.5	0.0	0.0	0.5
System	0.4	0.0	0.4	0.0	0.2

Fig. 4 Resource interactions for the synthetic credit application process

The superior performance of the autonomous setting can be attributed to two key factors. First, it better represents time- and congestion-related dynamics by preserving real interaction constraints, such as the lack of collaboration between Angela and Maria or between Steve, Patrick, and Oliver. Second, it better captures the control-flow by explicitly modeling the different execution orders used by clerks, particularly accounting for Oliver's distinct behavior.

A crucial limitation of the orchestrated setting is its inability to model realistic workload distributions. Since Maria works only part-time, the orchestrated approach, which assigns cases to the next available resource without explicitly modeling handovers, fails to capture realistic delays. For example, it allows for handovers from Steve to Patrick when Maria is unavailable even though this has never been observed in the event log, artificially reducing waiting times. Consequently, the orchestrated model underestimates cycle times by failing to simulate the workload bottlenecks caused by Maria's limited availability.

Finally, Fig. 4 illustrates the accuracy of resource interaction modeling in both settings. While the autonomous approach precisely reproduces the interaction patterns observed in the test log, the orchestrated approach completely disregards these patterns, leading to unrealistic resource allocations.

Related work

This section briefly discusses related work on automated BPS and agent-based modeling and simulation.

Automated BPS. We can divide existing literature on automated BPS approaches into three categories: Data-Driven Process Simulation (DDPS), Deep Learning (DL), and hybrid approaches. DDPS approaches automate simulation model discovery from event logs by initially identifying a process model and then enhancing it with simulation parameters. A semi-automated approach using colored Petri nets is proposed in Rozinat et al. (2009), while Khodyrev and Popova (2014) introduces a data-driven approach without considering resources. A more recent approach is Simod (Camargo et al. 2020), which improves upon Rozinat et al. (2009) and incorporates hyperparameter tuning. Later, López-Pintado and Dumas (2022) extended Simod by modeling differentiated resources. DL approaches for BPS typically rely on recurrent neural networks. LSTM models are employed in Tax et al. (2017) to predict events and timestamps, later improved upon by DGEN (Camargo et al. 2019), incorporating n-grams and embeddings. Due to their black-box nature, DL models are not applicable for what-if analysis. Hybrid models combine DDPS and DL approaches. DSIM (Camargo et al. 2022) combines a stochastic process model with DL for event timestamping, extended by RIMS that integrates predictions at runtime (Meneghello et al. 2023).

Agent-based modeling and simulation. Over the past decades, the application of MAS to various domains has been studied extensively (cf. Dorri et al. (2018) for a review). Applying agents to Business Process Management (BPM) was initially proposed in the 1990's (Jennings et al. 1996), where a business process is modeled as a system of negotiating agents. More recently, the concept of agent system mining has been introduced, recognizing that processes often emerge from interactions of autonomous agents (Tour et al. 2021), as demonstrated by an agent-based discovery algorithm (Tour et al. 2023), and also shown for simulation (Halaška and Šperka 2018). Furthermore, the recent proposal of agent system event data provides a promising data model for discovering meaningful agent types from event logs (Shen et al. 2025). A general introduction to agent-based BPS can be found in Sulis and Taveter (2022). However, such agent-based simulation approaches in BPM rely on manual configurations to simulate a specific process, e.g., in a factory production domain (Dornhöfer et al. 2020). To the best of our knowledge, our approach is the first to use event logs to automatically infer MAS models for process simulation.

Conclusion

This paper introduced *AgentSimulator*—an agent-based approach for data-driven business process simulation. Our approach discovers a multi-agent system from an event log, representing real-world actors and systems, each characterized by unique behaviors and interaction patterns. The discovered multi-agent system is then used to simulate the execution of the process. Our resource-first approach provides more means to capture distinct resource behaviors and interactions than traditional control-flow-first approaches and achieves state-of-the-art results.

The evaluation highlights that centrally orchestrated and decentralized processes often require different modeling approaches, with AgentSimulator automatically adapting to both scenarios. However, the dominance of orchestrated event logs in simulation datasets limits the extent to which autonomous behavior can demonstrate its advantages. Moreover, our findings indicate that simple statistical methods for learning agent behavior—responsible for determining the next activity and resource in an ongoing case—often rival or outperform more complex techniques such as LSTM networks and Petri nets, raising questions about the efficiency of these advanced methods.

Modeling human behavior remains a challenging task. While AgentSimulator effectively captures certain agent-specific behaviors, other factors, such as multitasking, may require additional dedicated modeling efforts. Additionally, AgentSimulator currently does not account for dynamic resource behaviors, such as improvements in activity processing durations over time due to learning effects. Our analysis comparing orchestrated and autonomous agent behavior settings is also limited by the availability of suitable event logs for simulation, particularly those derived from knowledge-intensive processes. To advance research in resource-centric simulation and simulation in general, it is crucial to develop and prepare more real-life event logs from a diverse range of application domains that are readily usable for simulation. This would allow for a more comprehensive comparison of different configurations of AgentSimulator across varying degrees of freedom in knowledge-intensive processes. Finally, we plan to enhance the usability of AgentSimulator by integrating the capability for conducting what-if analyses and providing a visual process representation, to increase its user adoption and provide an accessible framework for other researchers to contribute to the field of resource-centric simulation.

Authors' contributions
All authors contributed to the work's conception; L.K. suggested the main idea, implemented the approach, and conducted the experiments; L.K., R.B., T.K., and H.A. wrote and revised the manuscript throughout all its versions.

Funding
Open Access funding enabled and organized by Projekt DEAL.

Data availability
The implementation, datasets, experimental details, and obtained raw results are available through the repository linked in the Experiments and results section.

Declarations

Ethics approval and consent to participate
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 12 December 2024 Accepted: 25 March 2025
Published online: 07 April 2025

References
Augusto A, Conforti R, Dumas M et al (2019) Split miner: automated discovery of accurate and simple business process models from event logs. Knowl Inf Syst 59(2):251–284. https://doi.org/10.1007/s10115-018-1214-x
Berti A, van Zelst SJ, Schuster D (2023) Pm4py: A process mining library for python. Softw Impacts 17:100556. https://doi.org/10.1016/j.simpa.2023.100556
Camargo M, Dumas M, González-Rojas O (2020) Automated discovery of business process simulation models from event logs. Decis Support Syst 134:113284
Camargo M, Dumas M, Rojas OG (2019) Learning accurate LSTM models of business processes. In: BPM. Springer, Cham pp 286–302
Camargo M, Dumas M, Rojas OG (2022) Learning accurate business process simulation models from event logs via automated process discovery and deep learning. In: CAiSE. Springer, Cham, pp 55–71
Chapela-Campa D, Benchekroun I, Baron O et al (2023) Can I trust my simulation model? measuring the quality of business process simulation models. In: BPM. Springer, Cham, pp 20–37
Chapela-Campa D, Dumas M (2024) Enhancing business process simulation models with extraneous activity delays. Inf Syst 122:102346
Dornhöfer M, Sack S, Zenkert J et al (2020) Simulation of smart factory processes applying multi-agent-systems-a knowledge management perspective. JMMP 4(3):n. pag.
Dorri A, Kanhere SS, Jurdak R (2018) Multi-agent systems: A survey. IEEE Access 6:28573–28593

Dumas M (2021) Constructing digital twins for accurate and reliable what-if business process analysis. In: Beerepoot I, Ciccio CD, Marrella A, et al (eds) Proceedings of the International Workshop on BPM Problems to Solve Before We Die (PROBLEMS 2021) co-located with the 19th International Conference on Business Process Management (BPM 2021), Rome, Italy, September 6-10, 2021, CEUR Workshop Proceedings, vol 2938. CEUR-WS.org, pp 23–27.
Dumas M, Rosa ML, Mendling J et al (2013) Fundamentals of business process management. Springer, Cham
Halaška M, Šperka R (2018) Is there a need for agent-based modelling and simulation in business process management? Organizacija 51(4):255–269
Jennings NR, Faratin P, Johnson MJ et al (1996) Agent-based business process management. Int J Coop Inf Syst 5(2 &3):105–130
Kazil J, Masad D, Crooks AT (2020) Utilizing python for agent-based modeling: the mesa framework. In: social, cultural, and behavioral modeling: 13th international conference, SBP-BRiMS 2020, Washington, DC, USA, October 18–21, 2020, Proceedings 13, vol 12268. Springer, Cham, pp 308–317
Khodyrev I, Popova S (2014) Discrete modeling and simulation of business processes using event logs. In: ICCS. Elsevier, Amsterdam
Kirchdorfer L, Blümel R, Kampik T et al (2024) Agentsimulator: An agent-based approach for data-driven business process simulation. In: 6th International Conference on Process Mining, ICPM 2024, Kgs. Lyngby, Denmark, October 14-18, 2024. IEEE, pp 97–104. https://doi.org/10.1109/ICPM63005.2024.10680660
Leemans SJJ, Fahland D, van der Aalst WMP (2013) Discovering block-structured process models from event logs containing infrequent behaviour. In: Lohmann N, Song M, Wohed P (eds) Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers, Lecture Notes in Business Information Processing, vol 171. Springer, pp 66–78. https://doi.org/10.1007/978-3-319-06257-0_6
López-Pintado O, Dumas M (2022) Business process simulation with differentiated resources: Does it make a difference? In: BPM. Springer, Cham
López-Pintado O, Dumas M (2023) Discovery and simulation of business processes with probabilistic resource availability calendars. In: ICPM. IEEE, New York
Meneghello F, Francescomarino CD, Ghidini C (2023) Runtime integration of machine learning and simulation for business processes. In: ICPM. IEEE, New York
Meneghello F, Francescomarino CD, Ghidini C et al (2025) Runtime integration of machine learning and simulation for business processes: Time and decision mining predictions. Inf Syst 128:102472. https://doi.org/10.1016/j.is.2024.102472
North MJ, Macal CM (2007) Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation. Oxford University Press Inc, USA
Rozinat A, Mans RS, Song M et al (2009) Discovering simulation models. Inf Syst 34(3):305–327
Russell S, Norvig P (2020) Artificial Intelligence: A Modern Approach, 4th Edition. Pearson, London
Shen Q, Polyvyanyy A, Lipovetzky N et al (2025) Agent system event data: Concepts, dimensions, applications. In: Maass W, Han H, Yasar H et al (eds) Conceptual Modeling. Springer Nature Switzerland, Cham, pp 56–72
Sulis E, Taveter K (2022) Agent-Based Business Process Simulation - A Primer with Applications and Examples. Springer, Cham
Tax N, Verenich I, Rosa ML et al (2017) Predictive business process monitoring with LSTM neural networks. In: CAiSE. Springer, Cham
Tour A, Polyvyanyy A, Kalenkova AA (2021) Agent system mining: Vision, benefits, and challenges. IEEE Access 9:99480–99494
Tour A, Polyvyanyy A, Kalenkova AA et al (2023) Agent miner: An algorithm for discovering agent systems from event data. In: BPM. Springer, Cham
van der Aalst WMP (2015) Business process simulation survival guide. In: vom Brocke J, Rosemann M (eds) Handbook on Business Process Management 1, Introduction, Methods, and Information Systems, 2nd Ed. International Handbooks on Information Systems. Springer, Cham, pp 337–370

Publisher’s Note