Simulating Event Logs from Object Lifecycle Processes

Marius Breitmayer^{1,*}, Lisa Arnold¹ and Manfred Reichert¹

¹Institute of Databases and Information Systems, Ulm University, Germany

Abstract

Process Mining leverages event data to discover, analyze, and optimize business processes. Consequently, the availability and quality of event logs is crucial for the applicability as well as the development of process mining algorithms. However, obtaining such event logs from real-world scenarios is limited and costly. In this paper, we present an approach for simulating event logs from object lifecycle processes. Overall, the approach is capable of simulating event logs for object lifecycle processes that may be used to validate process mining algorithms.

Keywords

Event Log Simulation, Object-centric Processes, Object Lifecycle Processes, Flexibility

1. Introduction

Process mining is becoming more and more important for enterprises, as it provides a transparent and data-driven view on operational workflows while also offering insights into inefficiencies, bottlenecks as well as possible compliance issues. In general, process mining leverages event data to discover process models, check the conformance between models and event logs, or enhance the model [1]. In real-world scenarios, such event logs are often unavailable, inappropriate, or costly to obtain, consequently hindering the development and testing of new process mining algorithms [2].

In other domains, simulation provides a possible solution towards this problem by substituting or complementing real-world with simulated data [3, 4, 5].

In object-centric scenarios, the behavior of objects is defined in terms of object lifecycle processes. At runtime, however, objects may deviate from the behavior specified in object lifecycle processes due to flexibility [6]. Such deviations may include removing or adding both states and steps as well as executing them in a different order. Consequently, the simulation of event logs must account for such flexible behavior to ensure that the event logs are comparable to their real-world counterparts.

Overall, the simulation of event logs from object lifecycle processes presented in this paper facilitates the application and testing of process mining algorithms in object-centric scenarios by providing suitable event logs.

S. Böhm and D. Lübke (Eds.): 16th ZEUS Workshop, ZEUS 2024, Ulm, Germany, 29 February–1 March 2024, published at http://ceur-ws.org



¹⁶th Central European Workshop on Services and their Composition, February 29– June 01, 2024, Ulm, Germany *Corresponding author.

[☆] marius.breitmayer@uni-ulm.de (M. Breitmayer); lisa.arnold@uni-ulm.de (L. Arnold); manfred.reichert@uni-ulm.de (M. Reichert)

 ^{0000-0003-1572-4573 (}M. Breitmayer); 0000-0002-2358-2571 (L. Arnold); 0000-0003-2536-4153 (M. Reichert)
0 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Breitmayer et al.: Simulating Event Logs from Object Lifecycle Processes

The paper is structured as follows: Section 2 introduces object lifecycle processes. Section 3 describes the approach for simulating event logs. Section 4 evaluates the presented approach, and Section 5 discusses related work. Finally, Section 6 summarizes the paper and provides an outlook.

2. Fundamentals

In object-centric processes, the behavior of objects is specified in terms of a state-based object lifecycle process. Fig. 1 depicts the object lifecycle process for object *Submission*. An object lifecycle process comprises the states of the object as well as state transitions (see states *Edit*, *Submit*, *Rate*, and *Rated* in Fig. 1). Each state, in turn, comprises several steps (see *Exercise*, *E-Mail*, and *Files* in State *Edit* in Fig. 1) defining which object attributes are required before completing a state and transition to the next one. Note that we also support predicate steps to enable different execution paths through the object lifecycle process.



Figure 1: (simplified) Object Lifecycle Process Object Submission

During process execution, multiple objects (of the same or different type) may exist, and their instances are dynamically generated [7]. For each of these objects, their different object behavior may be observed on both the state level (i.e., how does an object change between its states), and the step level (i.e., how is data provided). Tab. 1 summarizes the guided, tolerated, and deviating behavior on both granularity levels.

On the state level, guided behavior corresponds to the object reaching its end state without returning to a previous state using a backwards transition. Tolerated behavior, in turn, includes backwards transitions. Deviating behavior on the state level is associated with skipping states (i.e., non-reachable) or reaching states not specified in the object lifecycle process model.

On the step level, guided behavior is observed if the steps of a state are provided in the order specified by the object lifecycle process. Tolerated behavior, in turn, only requires that all steps are provided (i.e., the order is not relevant). Deviating behavior on the step level includes skipping steps (i.e., not providing certain attributes) or providing additional steps in a particular state of the object lifecycle process model.

Level	Behavior	Description	
State	Guided	The end state is reached without following any backwards transitions.	
	Tolerated	The end state is reached, but backwards transitions were chosen.	
	Deviating	The lifecycle transitions to a non-reachable or unspecified state during its execution.	
Step	Guided	All steps of a lifecycle state are provided according to the pre-specified order.	
	Tolerated	All steps have been filled prior to state completion.	
	Deviating	Steps have been skipped or additional steps have been added.	

Table 1Object Behavior on state and step level [8]

3. Event Log Simulation

The event log simulation must be capable of simulating guided, tolerated, and deviating behavior on both the state as well as the step level (see Tab. 1). To be more precise, simulated event logs must be able to represent all 3 behaviors regarding the state as well as the step level. Due to their well-defined syntax and replay capabilities, we decided to represent object lifecycle processes in terms of Petri nets [9].

3.1. Object Lifecycle Process Representation

We enable the simulation of different behavior by representing each object lifecycle process as two Petri nets. One of these Petri nets corresponds to the guided behavior. In other words, this Petri net only allows for the object lifecycle process to be executed exactly as modeled. The other Petri net represents the tolerated behavior by allowing for additional (tolerated) behavior. This behavior includes, for example, providing object attributes in an order different from the one specified by the object lifecycle process model, while also ensuring that all required attributes are provided. The representation of object lifecycle processes in terms of Petri nets enables the application of token-based simulation on both nets.

Figs. 2 and 3 depict two Petri nets generated for state *Edit* of object *Submission* (see Fig. 1). In a nutshell, we represent guided behavior by accounting for the sequence in which steps (and states) are modeled in the object lifecycle process, whereas we allow for tolerated behavior using an AND-split syntax within the state. Note that, we only depict the guided and tolerated nets for state *Edit* of object *Submission*, however, we concatenate the guided and tolerated Petri nets for each state to represent the state level and include backwards transitions if necessary.



Figure 2: Guided Petri Net State Edit

Figure 3: Tolerated Petri Net State Edit

3.2. Trace Generation

After representing object lifecycle processes, we generate traces for individual objects. Alg. 1 describes the generation of traces in pseudocode. First, we check which transitions are enabled by the Petri net and check if the current marking equals the final marking. If not, we randomly choose an enabled transition, add it to the trace, and execute it. This updates the current marking. The trace generation terminates if no more enabled transitions exist, or the final marking is reached. In the latter case, the trace is added to the list of traces. Otherwise, the trace is discarded. This is repeated while the counter is smaller than the number of requested traces. In other words, we apply a token-based simulation in which tokens are propagated through the Petri net and the transitions passed are recorded as events in the event log. In a post-processing step, deviating behavior may be introduced to the generated event log by adding or removing corresponding events (i.e., writing (additional) attributes or transitioning to (additional) states).

Algorithm 1 Trace Generation						
Require: PetriNet	⊳ guided or tolerated net					
1: visitedTransitions = []						
2: marking ← copy(initialMarking)						
3: while True do						
4: if not semantics.enabled_transitions(Petr	riNet, marking) then					
5: Break						
6: end if						
7: enabledTransitions \leftarrow semantics.enabled	_transitions(PetriNet, marking)					
8: if marking == finalMarking then						
9: Break						
10: else						
11: transition ← randomElement(enabled	Transitions)					
12: end if						
13: if transition.label is not None then						
14: visitedTransitions.append(transition)						
15: end if						
16: marking ← semantics.execute(transition,	PetriNet,marking)					
17: end while						
18: if marking = finalMarking then						
19: allVisitedTransitions.append(visitedTrans	sitions)					
20: counter \leftarrow counter +1						
21: end if						

4. Evaluation

The evaluation of the presented approach for simulating event logs from object lifecycle processes is two-fold. First, we implemented a proof-of-concept prototype capable of simulating event logs from object lifecycle processes and applied it to two different scenarios (E-learning and Recruitment). Second, we checked the conformance of the simulated event logs with the models used for their generation and calculated resulting fitness values using alignments [8, 10]. On a scale from 1 (perfect fitness) to 0 (bad fitness), fitness measures to which degree a model allows for the behavior recorded in an event log. The proof-of-concept prototype as well as the event logs simulated during the evaluation are publicly available¹.

4.1. Conformance Checking

For every object in both the e-learning and recruitment scenario, we simulated 18 event logs (i.e., one log per possible configuration), each comprising 100 traces. In total, this results in 198 event logs from 11 different objects.

Tab. 2 describes the conformance categories into which the simulated traces are categorized for object *Submission* [8].

The event logs simulated for guided behavior configuration show the respective behavior in 100% of cases. Moreover, the event logs generated for the tolerated behavior show guided behavior in 17% of the cases and tolerated behavior in 83 % of cases. The inclusion of guided behavior within tolerated behavior is intended, as it generalizes guided behavior. Regarding deviating behavior (i.e., skipping or adding states and steps) all traces simulated with the respective configuration are correctly simulated with deviations.

Table 2

Categorization of Simulated Event Logs for Object Submission

	Guided Log	Tolerated Log	Deviating Log
Guided Behavior	100 %	17 %	0%
Tolerated Behavior	0%	83 %	0%
Deviating Behavior	0%	0%	100%

We further checked conformance of the simulated event logs regarding the state (i.e., how does the object instance change states?) and the step level (i.e., how are attributes written?). On the state level, this allows further verifying the deviations (e.g., additional states, unspecified backwards transitions, or state changes not allowed by the object lifecycle process model). We checked the conformance between the object lifecycle process model and the event logs simulated with additional states as well as steps in every trace on both the state and step level individually (see Tab. 3). In other words, the event log contained deviations on both the state and step level. Note that we only show the results for state *Edit* of object *Submission* for brevity.

Table 3

Object Submission on State and Step level

	State Level		Step Level (State Edit)	
	Average Fitness Value	% of traces	Average Fitness Value	% of traces
Guided Behavior	0.31	0 %	0.46	0 %
Tolerated Behavior	0.58	0 %	0.67	0 %
Deviating Behavior	NA	100 %	NA	100 %

Overall, the approach for generating event logs from object lifecycle processes is capable of reproducing all possible behavior based on the specification selected in the proof-of-concept implementation.

¹https://cloudstore.uni-ulm.de/s/M4AmamPLWZHniZ6

5. Related Work

The approach presented in this paper is related to the generation of event logs.

[11] presents the *PURpose-guided Log gEnerator (PURPLE)* capable of generating event logs conforming to a process model (i.e., a BPMN model or Petri net), including noise (i.e., skipping, adding, or reordering events). In contrast, the presented approach considers granularity and uses data-driven object behavior specified in object lifecycle process models as input.

[12] introduce the *Data Aware event Log Generator (DALG)* that can generate event logs from a Petri net. DALG offers several capabilities for temporal constraints and writing variables, however, no capabilities regarding deviations in the event logs are available.

The *Straightforward Petri Net-based Event Log Generation* plugin available in ProM [13] generates event logs based on Petri nets using token-based simulation. In contrast to the presented approach, it does not differentiate between guided and tolerated behavior and does not generate deviating event logs.

Event log generation in the context of Internet of Things devices is presented in [2]. A Petri net is used to generate guided event logs as well as event logs containing noise. In contrast to the presented approach, different granularity levels are not supported.

[14] can generate event logs from BPMN models and is available in the process mining framework *ProM*. The presented approach leverages data-driven object lifecycle processes rather than activity-centric BPMN models.

Other approaches leverage relational databases to generate event logs based on the data available in the database [15]. In this approach, a case table with a unique case ID has to be chosen. Users then recommend other tables that could provide additional events. These tables, however, have to contain the case ID as foreign keys. The approach presented in this paper can produce guided, tolerated and deviating behavior on both state and step level and it may generate event logs of different objects as well as multiple instances of each object.

6. Summary & Outlook

This paper presented an approach for simulating event logs from object lifecycle processes, while considering the behavior of objects on both the state and the step level. The approach is capable of simulating guided, tolerated, and deviating object behavior and document the behavior in event logs using multiple Petri nets derived from object lifecycle processes. It has been evaluated using a publicly available proof-of-concept implementation which has been successfully applied to simulate event logs from different scenarios. Event logs simulated using the presented approach reliably conform with the specification set out, facilitating the development and testing of new process mining algorithms by providing suitable event logs for a variety of scenarios. In future work, we plan to extend the approach in multiple directions. First, we want to incorporate coordination constraints between objects of different types (i.e., object A may only reach state S_A if object B is in state S_B). Second, the actual object attribute values documented in the event log are currently generic placeholders. We plan to replace them with more realistic, process-specific values. Finally, we plan to provide the event logs in additional formats (e.g., OCEL [16] or XES [17]) instead of the currently used CSV-format.

Breitmayer et al.: Simulating Event Logs from Object Lifecycle Processes

Acknowledgments This work is part of the ProcMape project, funded by the KMU Innovativ Program of the Federal Ministry of Education and Research, Germany (F.No. 01IS23045B)

References

- W. M. P. van der Aalst, Process Mining: Data Science in Action, 2 ed., Springer, Heidelberg, 2016.
- [2] Y. Zisgen, D. Janssen, A. Koschmider, Generating synthetic sensor event logs for process mining, in: International Conference on Advanced Information Systems Engineering, Springer, 2022, pp. 130–137.
- [3] J. Chen, D. Chun, M. Patel, E. Chiang, J. James, The validity of synthetic clinical data: a validation study of a leading synthetic data generator (synthea) using clinical quality measures, BMC medical informatics and decision making 19 (2019) 1–9.
- [4] N. Patki, R. Wedge, K. Veeramachaneni, The synthetic data vault, in: 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA), 2016, pp. 399–410.
- [5] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, S. Birchfield, Training deep networks with synthetic data: Bridging the reality gap by domain randomization, in: Proceedings of the IEEE conference on computer vision and pattern recognition workshops, 2018, pp. 969–977.
- [6] K. Andrews, S. Steinau, M. Reichert, Enabling runtime flexibility in data-centric and data-driven process execution engines, Information Systems 101 (2021) 101447. URL: https://www.sciencedirect.com/science/article/pii/S0306437919304995. doi:https://doi. org/10.1016/j.is.2019.101447.
- [7] S. Steinau, K. Andrews, M. Reichert, Executing lifecycle processes in object-aware process management, in: Data-Driven Process Discovery and Analysis, Lecture Notes in Business Information Processing, Springer, 2017, pp. 25–44.
- [8] M. Breitmayer, L. Arnold, M. Reichert, Enabling conformance checking for object lifecycle processes, in: Research Challenges in Information Science, Springer International Publishing, Cham, 2022, pp. 124–141.
- [9] J. L. Peterson, Petri nets, ACM Computing Surveys (CSUR) 9 (1977) 223-252.
- [10] W. M. P. van der Aalst, A. Adriansyah, B. van Dongen, Replaying history on process models for conformance checking and performance analysis, WIREs Data Mining and Knowledge Discovery 2 (2012) 182–192.
- [11] A. Burattin, B. Re, L. Rossi, F. Tiezzi, Purple: a purpose-guided log generator, in: Proceedings of the ICPM Doctoral Consortium and Demo Track 2022, CEUR Workshop Proceedings, CEUR-WS, 2022.
- [12] D. Jilg, Generating synthetic procedural multi-perspective electronic healthcare treatment cases, 2022.
- [13] S. Vanden Broucke, J. Vanthienen, B. Baesens, Straightforward petri net-based event log generation in prom, SSRN 2489051 (2014).
- [14] A. A. Mitsyuk, I. S. Shugurov, A. A. Kalenkova, W. M. P. van der Aalst, Generating event logs for high-level process models, Simulation Modelling Practice and Theory 74 (2017) 1–16.

Breitmayer et al.: Simulating Event Logs from Object Lifecycle Processes

- [15] R. Andrews, C. G. J. van Dun, M. T. Wynn, W. Kratsch, M. Röglinger, A. H. ter Hofstede, Quality-informed semi-automated event log generation for process mining, Decision Support Systems 132 (2020) 113265.
- [16] A. F. Ghahfarokhi, G. Park, A. Berti, W. M. P. van der Aalst, Ocel: A standard for objectcentric event logs, in: L. Bellatreche, M. Dumas, P. Karras, R. Matulevičius, A. Awad, M. Weidlich, M. Ivanović, O. Hartig (Eds.), New Trends in Database and Information Systems, Springer International Publishing, Cham, 2021, pp. 169–175.
- [17] Ieee standard for extensible event stream (xes) for achieving interoperability in event logs and event streams, IEEE Std 1849-2016 (2016) 1–50. doi:10.1109/IEEESTD.2016.7740858.