DORY: Dynamic Object-centric Relations for sYnthetic simulation

Denise Cosolo[1], Francesca Meneghello[2] and Massimiliano Ronzani[3]

[1]Free University of Bolzano, Italy
[2]Technical University of Denmark, Kgs. Lyngby, Denmark
[3]Fondazione Bruno Kessler, Trento, Italy

Abstract

Object-Centric Process Mining (OCPM) models real-world processes where multiple interacting objects evolve together during execution. OCPM is widely used to support decision-making via what-if analysis and for evaluating process mining techniques. However, existing OCPM simulation tools fail to capture the dynamic evolution of objects and their evolving relationships over time, including creation, modification and removal. DORY is an object-centric process simulator that address these limitations. The system simulates multi-object processes in which objects and their relationships are dynamically generated, and removed during execution. By combining Petri net–based process definitions with configurable simulation parameters, DORY generates realistic object-centric event logs.

Keywords

Business Process Simulation, Object-centric Process Mining, Digital Twin

1. Introduction

Object-centric Process Mining (OCPM) has gained signification traction due to the limitations of the traditional case-centric paradigm to describe interactions among multiple objects in real-world processes. In this context, Business process simulation (BPS) provides a means to generate synthetic object-centric event logs, enabling the analysis of system behavior, the exploration of what-if scenarios, and the evaluation of process mining techniques when real-world data is unavailable. In the object-centric setting, simulation should capture the dynamics of multiple object types that coexist and interact throughout the process, establishing, modifying, and dissolving relationships over time.

In this field, only a limited number of works have focused on developing BPS techniques for OCPM. Starting from an Object-Centric Event Log (OCEL), [1] employs a simulation model based on a graph representing objects and their relationships, which is used to manage objects during the simulation. The model allows the discovery of object relationships from OCEL, without allowing for their dynamic evolution. In fact, relationships between objects are predefined at the start of the simulation and cannot be created or removed dynamically during its execution. Another relevant work is Renew [2], which supports the development and execution of object-oriented Petri nets enriched with synchronous channels. Nevertheless, as discussed in [1], Renew exhibits significant limitations with respect to dynamic creation or deletion of relationships at runtime and requires a detailed specification of object types, relationships, and constraints.

In [3] we proposed a novel discrete-event simulation (DES) framework able to reproduce dynamic relationships among objects. The simulation model is defined as a collection of Petri nets, one for each object type, while channels are used to manage synchronization between objects, enabling the creation, manipulation, and inspection of relationships. In particular, the process model in [3] builds on a trade-off between two state-of-the-art object-centric process model, namely Proclets [4] and DOPIDs [5], with the goal of providing an intuitive and easy-to-use simulation model. It leverages Proclets capability to clearly encapsulate the behavior of each object type and where it needs to synchronize with that of

Proceedings of the BPM 2026 Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Forum (BPM 2026), Toronto, Canada, September 27th to October 2nd, 2026.

denise.cosolo@student.unibz.it (D. Cosolo); frmen@dtu.dk (F. Meneghello); mronzani@fbk.eu (M. Ronzani)

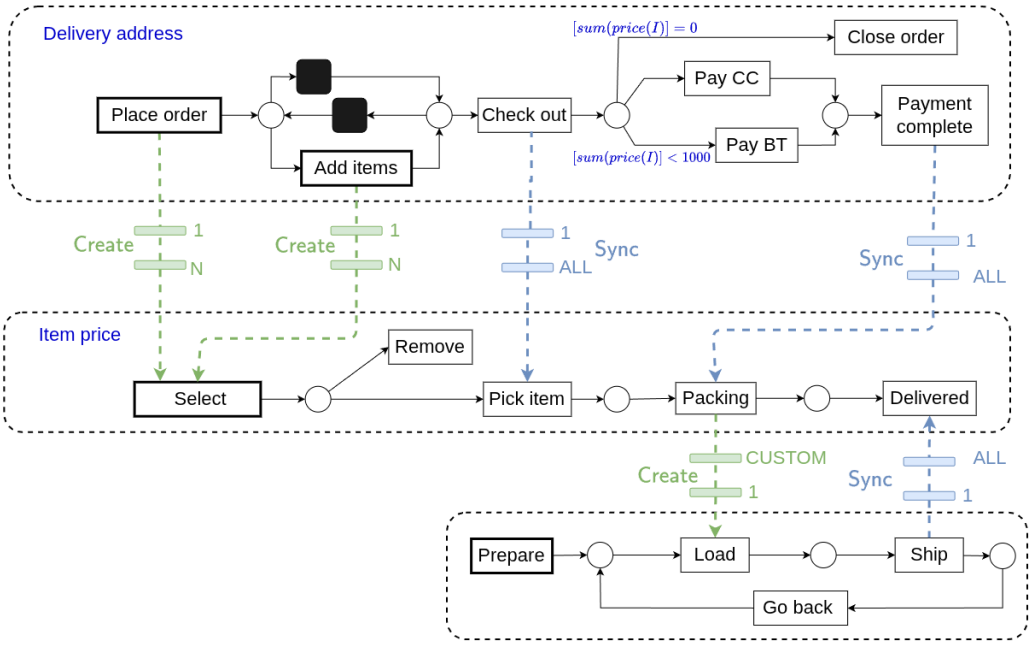


Figure 1: Example of order-to-ship object-centric process

other types, making them well-suited for simulator implementation, while using DOPIDs to capture time-varying relationships and to express synchronization conditions and data-dependent constraints.

In this demo paper we present DORY (Dynamic Object-centric Relations for sYnthetic simulation) based on the simulation framework in [3]. In comparison with the work in [3], we improve the usability in the setting of the simulation model without compromising its flexibility and high degree of configurability.

Specifically, DORY provides a high degree of freedom by defining each object type through its own Petri net and simulation parameters related to time, resources, data, and control-flow perspectives. Similarly, for relationships between objects, it is possible to specify how many objects can participate in a transition and how they are selected, namely by defining cardinality constraints and selection rules.

For the simulation framework presented in [3], slight adjustments have been introduced for simulation purposes, in particular in the representation of channels, in order to ensure a closer alignment with the underlying simulation implementation while preserving the original semantics of the model. These modifications also improve usability by making it easier for users to set and configure simulation parameters. Additionally, we include a preparation step that generates a template input for DORY, simplifying the specification of the simulation model and its parameters by relieving users from defining them from scratch. Lastly, we provide users with the possibility to conduct general statistics analysis including number of created objects and simulation duration.

2. Motivating Example

To illustrate the capabilities of DORY, we defined a concrete scenario based on a typical order-to-ship process. A customer creates an order with an associated city of delivery and a set of items, each characterized by its own price. The order can be modified by adding or removing items until all items are confirmed for the payment. After that, order fulfilment proceeds: the items are packed and assigned to trucks based on their destination, capacity, and availability. Indeed, trucks may perform multiple delivery rounds across different cities. The process terminates when all items belonging to an order have been delivered. A possible representation of the process could be given by the three Petri nets in Figure 1, where the objects synchronization are represented by dashed arrows between transitions of different object types.

In this example, the relationships among individual object types are not known a priori but emerge during process execution as a result of object interactions and their life-cycle. For example, neither the

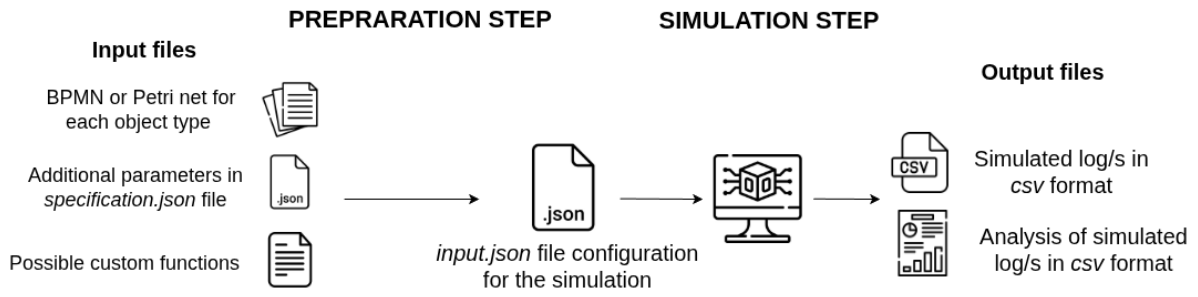


Figure 2: Structure of DORY defined as input and output files across different steps

number of items associated with an order nor the truck responsible for delivering a given item can be determined in advance. The assignment of a truck depends on several factors, including the completion of item packing, payment confirmation by the customer, and the truck's current availability, which is itself influenced by previous delivery and the truck's capacity.

Therefore, to accurately simulate this object-centric process and answer questions such as how many delivery rounds each truck will perform or which destinations it will serve, it is essential to dynamically reproduce the evolving relationships among objects throughout process execution.

3. DORY: Approach

As shown in Figure 2, the execution of DORY consists of two main steps. The first step concerns the preparation of the input files: *i)* a BPMN or Petri net model for each object type involved in the process; *ii)* a `specification.json` file, in which users have to specify the relationships between object types and, optionally, the simulation parameters related to each object; *iii)* optionally, a `custom_function.py` configuration file, in which users can define custom functions to manage different process perspectives.

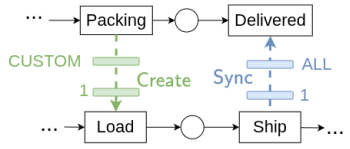
The script `preparation.py` automatically transforms the BPMN models into Petri nets (if needed) and produces, based on `specification.json`, the `input.json` file that drives the DES simulation. This file contains all simulation parameters, namely those referring to individual objects as well as those defining the relationships between them. If users do not specify the simulation parameters related to a single object type, the script automatically inserts default values as a template that can be further customized by the user. This preparation step can be skipped if the user has already manually created `input.json` file and defined the Petri nets.

The second step consists of running the simulation, obtaining as output a simulated object-centric log in *csv* format[1]. Additionally user can perform brief statistical analyses over the simulated results.

Figure 1 provides a graphical representation of the `input.json` file and the corresponding Petri nets for the order-to-ship process described in Section 2 and originally introduced in [3]. Compared to [3], it has been adapted for usability purposes and to facilitate a more straightforward translation into the simulation model of DORY. In particular, for simplicity, directional channels are established between transitions belonging to different object types. The direction of a channel encodes ordering constraints, specifying that certain transitions must be executed before others can be enabled and fired for the corresponding objects. The following paragraphs briefly describe the simulation parameters, first at the level of individual object types and then at the level of relationships between objects in the object-centric process model.

Object simulation parameters To define the behaviour of an individual object type, the Petri net describing its control flow must be complemented with a set of simulation parameters. These parameters are specified following the conventions adopted in traditional case-centric BPS models [7, 8]. Specifically, they cover the time perspective (i.e., transition processing times, object arrival rates),

[1]By default, the simulation produces an OCEL, adopting the so-called parameter semantics of [6], recording only event-to-object relationships. Optionally, object-to-object relationships can also be logged at each event adopting the snapshot semantics of [6].




```
"item":{
   "object_constraints": {"Delivered": {"obj": "truck", "trans": ["Ship"], "
        card": "All"}}
}
"truck": {
    "object_constraints": {"Load":{"obj":"item", "trans":["Packing"], "card":"
         CUSTOM"}},
    "create_relationship": {"Load": "item"},
    "destroy_relationship": {"Ship": "item"}
}
```

Figure 3: Create and Sync channels between item and truck and the corresponding section in `specifications.json`

the resource perspective (i.e., the resources that can perform activities), the data perspective (i.e., the attributes associated with the object type), and the decision perspective (i.e., the routing logic at decision points). As proposed in [8], users can define customizable functions for each simulation perspective, in `custom_function.py`, for example, to define object attributes during the simulation, specify custom processing times for transitions, and handle decision points according to user-defined logic.

Object Relationship Simulation Parameters Object relationships are modelled through channels connecting different object types, which regulate the synchronization of objects during the simulation. Specifically, in DORY we define two different types of channels, namely *Create* and *Sync*. *Create* channels can have cardinalities of 1 or N, while *Sync* channels can have 1, N, All, [min, max], CUSTOM, as they can represent a variety of scenarios with different requirements and constraints.

Create channels can generate new objects in the simulation and/or establish new relationships between existing objects. For instance, in Figure 1, the channel originating from Place Order points to the Select transition, capturing the fact that order triggers the creation of new *Item* objects and the new relationship with the corresponding order. This relation is labelled in the json file as "`task_generator`" within the *Order* object, which specifies the transition responsible for creating new objects of type *Item* (i.e. Place Order), along with the distribution parameters driving their generation. On the *Item* side, the relationship is represented through the field `"generator_by": ["order"]`, indicating that each item is created by a specific order.

In addition to generating new objects, *Create channels* can also define and instantiate relationships between existing objects within the simulation, as illustrated in Figure 3 for the case of *Item* and *Truck*. In this scenario, after order payment, picked items are packed, loaded onto a truck, shipped, and then marked as delivered. This scenario regards a *Create* channel between items and trucks to establish which items are loaded onto the truck, and then a *Sync* channel updates the status as delivered to the items. This relationship is specified through the "`create_relationship`" entry, paired with the transition (Load) that realizes this activity. On the *Truck* side, the execution of Load is constrained by the state of the associated *Item* objects. In particular, items must be packed before loading, and the grouping logic is controlled by a cardinality parameter, which may take values such as 1, N, All, or [min,max]. When `"CUSTOM"` is specified, the corresponding rule is implemented in `custom_function.py`, allowing users to define experiment-specific behaviour; in this example, items are grouped by destination city when being loaded onto trucks.

On the other hand, *Sync channels* are used to synchronize the execution of two transitions, leveraging an existing relationship. In Figure 3 this is illustrated by the channel between Ship and Delivered. In this case, once the truck has performed the shipment of its associated items, the Delivered transition in the item process can be fired to update item's status. This behaviour is specified within the "`object_constraints`" entry, which includes the full specification of the channel. More specifically, once the shipment is completed, two effects occur: the Delivered transition is triggered for the corresponding items, and the relationship between *item* and *truck* is removed, which is reflected on the truck side through the "`destroy_relationship`" entry.

4. Maturity and Conclusion

DORY is a Python-based tool implemented by leveraging the Simpy[2] and the PM4Py library[3]. The documentation on how to install and run the tool, as well as a video and the code of DORY are available at the Github repository[4]. The code provided in this demo paper is tailored to the motivating example in Figure 1. As a preliminary stress test, we simulated up to 500 orders and 10 trucks (2152 objects), obtaining an execution time of about 30 seconds; however, we did not exhaustively explore the scalability limits of the simulator beyond this setting. As further limitations, we did not evaluate the simulator's performance and expressiveness against other existing tools, and we have not yet systematically tested it across the range of object-centric processes available in the community.

DORY is an object-centric process simulator that supports the dynamic evolution of relationships among object types. Its key contribution is the explicit handling of runtime-changing relationships and the incorporation of their effects into the progression of the simulation. In addition, DORY supports the definition of customizable simulation parameters for each object type, resulting in a flexible and highly expressive simulator. This allows the incorporation of predictive models, enabling a transition from a DES simulator to a hybrid approach, which we plan to explore as future work.

Acknowledgments

This work was partially supported by the Innovation Fund Denmark (IFD) under the project P3AI4 - Predictive and Prescriptive Process Analytics for Industry 4.0 (Grant Nr. 4355-00010B). The authors thank Prof. Chiara Ghidini and Prof. Marco Montali (Free University of Bolzano) and Prof. Chiara Di Francescomarino (University of Trento) for their support during the development of the paper.

Declaration of generative AI and AI-assisted technologies

During the preparation of this work, the authors used OpenAI[5] tools language editing and manuscript clarity. The authors reviewed and edited the content and take responsibility for the published article.

References

[1] B. Knopp, M. Pourbafrani, W. M. P. van der Aalst, Discovering object-centric process simulation models, in: 5th International Conference on Process Mining, ICPM 2023, Rome, Italy, October 23-27, 2023, IEEE, 2023, pp. 81–88.

[2] D. Moldt, J. Johnsen, R. Streckenbach, L. Clasen, M. Haustermann, A. Heinze, M. Hansson, M. Feldmann, K. Ihlenfeldt, RENEW: modularized architecture and new features, in: Proceedings of PETRI NETS, volume 13929 of *LNCS*, Springer, 2023.

[3] D. Cosolo, C. Di Francescomarino, C. Ghidini, F. Meneghello, M. Montali, M. Ronzani, Simulating dynamic relationships in object-centric processes, in: BPM Forum, 2026. To appear.

[4] D. Fahland, Describing behavior of processes with many-to-many interactions, in: Proceedings of PETRI NETS, volume 11522 of *LNCS*, Springer, 2019.

[5] A. Gianola, M. Montali, S. Winkler, Object-centric processes with structured data and exact synchronization - formal modelling and conformance checking, in: Proceedings of CAiSE, volume 15702 of *LNCS*, Springer, 2025.

[6] A. Gianola, Z. Hameed, M. Montali, A. Seidel, M. Weske, S. Winkler, Detecting dynamic relationships in object-centric event logs, CoRR abs/2604.13053 (2026).

[7] D. Chapela-Campa, O. López-Pintado, I. Suvorau, M. Dumas, Simod: automated discovery of business process simulation models, SoftwareX 30 (2025) 102157.

[8] F. Meneghello, C. D. Francescomarino, C. Ghidini, Rims_tool: a hybrid simulator for business processes, in: Doctoral Consortium and Demo Track at the International Conference on Process Mining (ICPM), CEUR Workshop Proceedings, 2023.

[2] https://simpy.readthedocs.io/en/latest/

[3] https://pm4py-source.readthedocs.io/en/stable/

[4] https://github.com/francescameneghello/object_centric_simulator

[5] https://openai.com/