# OC-SimPN: Object-Centric Petri net Simulation

Tijmen Kuijpers[1,*], Karolin Winter[1] and Remco Dijkman[1]

[1]Eindhoven University of Technology, Groene Loper 3, 5612 AE Eindhoven, The Netherlands

**Abstract**

Object-centric process mining has developed into a well-established research stream and object-centric Petri nets as a means to model object-centric processes. However, tool-support for modeling and simulating object-centric Petri nets remains limited. Nonetheless, object-centric process simulation is important for many practical applications, such as event-log generation, what-if analysis, and simulation-optimization. To fill this gap, we introduce OC-SimPN, extending the Python-based simulation library SimPN with support for modeling and simulating object-centric Petri nets. In addition to the functionality provided by existing object-centric Petri Net simulation tools, this tool provides a means for expressive modeling of process behavior and optimization through integration with Python.

**Keywords**

Object-centric Petri nets, Simulation, OCEL 2.0

## 1. Introduction and Significance to BPM

Object-centric process mining techniques [1] have developed into a well-established research stream in the Business Process Management (BPM) community. Traditional process mining assumes that processes can be analyzed by tracking a single case id, and each event observed in a log corresponds to a single case. In processes where multiple objects interact, e.g., a supply chain with multiple items corresponding to multiple shipments and a single purchase order, this assumption is violated. To enable object-centric process mining, Object-centric Petri nets (OCPN) have been formalized [2], and the OCEL 2.0 log format [3] has been proposed as the standard for object-centric event logs. While these standards have enabled researchers and practitioners to study and apply object-centric process mining techniques, the simulation support for OCPNs remains limited. Tool support for simulation is beneficial as it enables simulation-optimization and what-if analysis. Furthermore, high data quality requirements and scarce availability of real-world logs motivate the need to develop simulated logs.

Currently, there exists only one tool for modeling and simulating OCPNs, OCPN Studio [4]. However, where OCPN Studio focuses on providing a means for easy graphical modeling and simulation, we provide a programmatic alternative. OC-SimPN allows for the integration with existing Python code, for example, for solving object-centric business process optimization problems. OC-SimPN also allows for the creation of higher-level language construct, thus facilitating, for example, the definition of an object-centric variant of BPMN with an underlying Petri-net semantics. Finally, Python allows for more convenient modeling of expressions, in particular for (timing) probability distributions, which are commonly used in simulation.

SimPN is a Timed-arc Colored Petri net (TA-CPN) simulation tool in Python [5]. In this paper, we extend SimPN to enable modeling, visualization, simulation and OCEL log reporting for OCPNs. Specific contributions with respect to the existing SimPN library are (1) variable arcs with a function for selecting which tokens to fire from the place(s) with the variable arc(s), (2) OCPN binding execution and well-formedness checking, (3) object-type based visualization of places and transitions, (4) object-centric reporters for Excel, OCEL and OCEL 2.0 files. In Section 2, we describe the functionality in OC-SimPN to support OCPN simulation. Basic elements of the OCPN simulator are demonstrated, and the use

*Proceedings of the Best BPM Dissertation Award, Doctoral Consortium, and Demonstrations & Resources Forum co-located with 24th International Conference on Business Process Management (BPM 2026), Toronto, Canada, September 27th to October 2nd, 2026.*

*Corresponding author.

t.p.kuijpers@tue.nl (T. Kuijpers); k.m.winter@tue.nl (K. Winter); r.m.dijkman@tue.nl (R. Dijkman)

of visualizations and log reporters is described. Next, the innovations and maturity of OC-SimPN are discussed in Section 3. Finally, we conclude with some final remarks in Section 4.

## 2. Functionality

### 2.1. Modeling

The main elements of an OC-SimPN model are *OCPNVars*, which represent places, and *OCPNEvents*, which represent transitions. They are abstractions of elements used for TA-CPN simulation in SimPN. These abstractions integrate object-type checking, well-formedness checking, and variable arcs. Before we specify these variables, we instantiate a *SimProblem*. Any functionality that is provided with SimPN is also applicable to the OC-SimPN models provided in this demo paper. We refer the interested reader to the original SimPN demo paper [5].

**OCPNVar.** To create an *OCPNVar* we specify what *SimProblem* it should be added to, which is "model" in the example below. To ensure tokens of a certain object_type can only reside in places of that object type, we need to specify the parameter *object_type*. We also specify an *id* for internal reference. An optional *priority* parameter allows for specification of a priority function that sorts the tokens (default is time). In the example below, we created a *order_arrival* and *item_arrival* place, with object types "order" and "item", and initialized both with an object. To put objects in an *OCPNVar*, we need to specify at least the *object_type* to ensure place-token object type consistency.

```
#Create a simulation model
model = SimProblem()

# Create variables for different object types
order_arrival = OCPNVar(model, object_type="order", id="order_arrival")
item_arrival = OCPNVar(model, object_type="item", id="item_arrival")

# Put some objects in the system
order_arrival.put({"object_type": "order", "order_id": 1, "items": 2})
order_arrival.put({"object_type": "order", "order_id": 2, "items": 1})
item_arrival.put({"object_type": "item", "order_id": 1, "size": "small"})
...
```

**OCPNEvents.** To create an *OCPNEvent*, we must specify its *incoming_vars* and *outgoing_vars*, including if they are variable arcs (True) or not (False). Variable arcs can consume multiple tokens from the corresponding place when firing a transition once. This is specified by putting a tuple with (*OCPNVar*, True/False) for each incoming and outgoing place in the parameters *incoming_vars* and *outgoing_vars*. Since multiple tokens can be consumed when firing a transition with a variable arc, we need to select which tokens should be fired in an enabled binding. For this, we introduce a *selection* parameter that takes a function to determine what tokens are selected. the *guard* and *behavior* functions are called with the tokens passed by the *selection* function as arguments. Note how the integration in Python enables specifying a size-dependent delay probability distribution for the items. The *OCPNEvent* created is validated to check if objects of a certain type are only consumed from, and produced to, places corresponding to that object-type. The OCPNs well-formedness is checked by ensuring that an *OCPNVar* that is incoming with a variable arc can always be matched to an *OCPNVar* of the same object-type with an outgoing variable arc from the *OCPNEvent*. In the example below, a set of item-objects with the same "order_id" is fired together with the corresponding order-object.

```
def process_items_behavior(order, item_queue):
   put_inventory = []
   for item in item_queue:
      delay = np.random.exponential(scale=5) if item.value["size"]=="large" else 2
      put_inventory.append(SimToken(item.value, delay=delay))
```

```
    return [SimToken(order, delay=delay), put_inventory, []]

def process_items_selection(order, item_queue):
    selected_items = [item for item in item_queue if item.value["order_id"] == order["
        order_id"]]
    return [order, selected_items]

process_event = OCPNEvent(model=model,
                  incoming_vars=[(order_arrival, False),
                                 (item_arrival, True)],
                  outgoing_vars=[(pipeline_order, False),
                                 (pipeline_item, True)],
                  name="place_order",
                  guard=None,
                  behavior=process_items_behavior,
                  selection=process_items_selection)
```

## 2.2. Visualizing

After making an OC-SimPN model, the resulting OCPN can be used for visualization, step-wise animation, simulation, and OCEL log reporting. We illustrate an example used in [2] to formalize and discover OCPNs in Figure 1. The model shows a process with orders, consisting of multiple items, arriving simultaneously. The items need to be picked and shipped. For the orders an invoice must be sent and paid by the customer, possibly after some reminders. The order can be completed when it is paid, and all items corresponding to this order are shipped. The implementation shows places and transitions colored according to the object types related to them. Transitions with multiple incoming or outgoing object types are colored according to all those object types. Variable arcs are represented with double arrows.

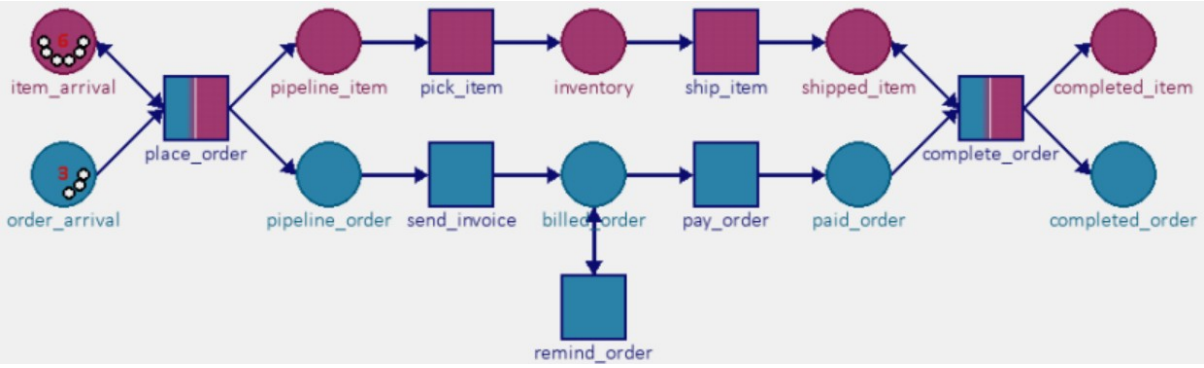


**Figure 1:** OC-SimPN visualization

## 2.3. Optimizing

OC-SimPN models can be embedded in simulation-based optimization approaches in Python. We show a simple example of an optimization heuristic applied in OC-SimPN for the *pick_item* transition. In the code fragment below, we apply a Shortest Processing Time (SPT) heuristic that prioritizes items belonging to a paid order object. The selection function passes the item that belongs to a paid order according to SPT logic based on the item's size. The behavior function adds a delay to the selected item corresponding to its size. If there are no paid orders waiting, the selection defaults to passing tokens with the lowest enablement time. This example illustrates how integration in Python enables optimization, and by no means provides the "best" solution to this problem. Depending on the objectives of the optimization problem, different solutions can be implemented.

```
def pick_item_selection(pipeline_item):
   paid_order_ids = [token.value["order_id"] for token in paid_order.marking]
   item_options = [item for item in pipeline_item if item.value["order_id"] in
       paid_order_ids]
   items = item_options if paid_order_ids and len(item_options) > 0 else list(
       pipeline_item)

   size_SPT = {"small": 0, "medium": 1, "large": 2}
   best_SPT_item = min(items, key=lambda x: size_SPT[x.value["size"]])

   return [[best_SPT_item]]

def pick_item_behavior(pipeline_item_queue):
   size_delays = {"small": 3, "medium": 5, "large": 7}
   picked_item = pipeline_item_queue[0]
   delay = size_delays[picked_item.value["size"]]
   return [[SimToken(picked_item.value, delay=delay)], []]

pick_item_event = OCPNEvent(model=model,
                 incoming_vars=[(pipeline_item, True)],
                 outgoing_vars=[(inventory, True)],
                 name = "pick_item",
                 guard = None,
                 behavior = pick_item_behavior,
                 selection = pick_item_selection)
```

### 2.4. Generating OCEL logs

We implemented an *OCELReporter* to output simulation logs in Excel, OCEL 1.0 and OCEL 2.0 format. An example of (part of) an event log in Excel format is given in Table 1. The Excel output provides a log in which each row records an activity, a timestamp, and the objects involved, grouped by object type. OCEL 1.0 extends this by structuring the same information as a standardized object-centric event log: events and objects are represented as separate, globally identifiable entities, linked through object maps. OCEL 2.0 further extends OCEL 1.0 by introducing an explicit type model for events and objects, event-to-object relationships, and time-aware object attributes. It also supports object-to-object relationships and event attributes, allowing richer descriptions of how objects interact and how their properties evolve. In our example, these latter elements are present as placeholders in the exported log: event attributes, object-to-object links, and semantically specific relationship qualifiers are not populated, but the format remains structurally compatible with the OCEL 2.0 standard.

| event_id | activity | timestamp | order | item |
|---|---|---|---|---|
| 20260201 | place_order | 0 | [1] | [1, 2] |
| 20260202 | place_order | 0 | [2] | [3, 4] |
| 20260203 | pick_item | 0 | [] | [1] |
| 20260204 | pick_item | 0 | [] | [2] |
| 20260205 | pick_item | 0 | [] | [3] |

**Table 1**
Generated event log using OC-SimPN model

## 3. Innovation and Maturity

To the best of our knowledge, the the OC-SimPN tool presented in this demo is the only Python tool that supports the simulation of OCPNs. We extended SimPN by implementing *OCPNVar*, *OCPNEvent* and

*OCELReporter*. This enabled the specification of variable arcs, a variable arc token selection function, OCPN binding execution checking, well-formedness checking, OCPN visualizations, and Excel, OCEL 1.0, OCEL 2.0 log reporting. Compared to OCPN studio, OC-SimPN, through its integration in Python, supports simulation-optimization and what-if analysis with expressive guard, behavior, and selection functions. Table 2 provides a comparison between OCPN studio, the SimPN library and OC-SimPN.

| | OCPN studio [4] | SimPN [5] | OC-SimPN |
|---|---|---|---|
| Object-centric Petri net (OCPN) modeling/simulation | ✓ | ✗ | ✓ |
| Transition guards/behavior/selection defined in Python | ✗ | ✓ | ✓ |
| Visualization of the simulation model | ✓ | ✓ | ✓ |
| Optimization in Python | ✗ | ✓ | ✓ |
| Reporter for OCEL event logs | ✓ | ✗ | ✓ |

**Table 2**
Table 2: Comparison of OCPN studio, SimPN, and OC-SimPN using criteria discussed in this paper.

The SimPN library that forms the basis for OC-SimPN has been successfully used for three years in the course 'Business Process Simulation' at Eindhoven University of Technology, with 89 students enrolled in 2026. OC-SimPN was tested by three researchers for early-stage work. The tool is demonstrated in a screencast [1] and publicly available on GitHub [2].

## 4. Conclusion

The OC-SimPN tool presented in this demo paper provides researchers and practitioners with the first Python-based library for the simulation of OCPNs. The tool can be used for visualization, step-by-step animation, simulation, and OCEL log reporting of object-centric processes. The implementation in Python allows for a high degree of expressiveness in defining guards, behavior, and token selection. Integration with other libraries facilitates the use of OC-SimPN for simulation-optimization and what-if analysis. Next steps are the use of the OC-SimPN tool for advanced analysis and optimization of event-to-object and object-to-object relations in object-centric process models.

## Declaration on Generative AI

During the preparation of this work, the authors used Cursor and Grammarly in order to assist with code writing, as well as grammar and spelling checks. After using these tools/services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

## References


[1] W. M. van der Aalst, Object-centric process mining: Unraveling the fabric of real processes, Mathematics 2023, Vol. 11, Page 2691 (2023) 2691.

[2] W. M. V. D. Aalst, A. Berti, Discovering object-centric petri nets, Fundamenta Informaticae (2020) 1–40.

[3] A. Berti, I. Koren, J. N. Adams, G. Park, B. Knopp, N. Graves, M. Rafiei, L. Liß, L. Tacke, G. Unterberg, Y. Zhang, C. Schwanen, M. Pegoraro, W. M. P. V. D. Aalst, Ocel (object-centric event log) 2.0 specification (2024) 1–48.

[4] I. Koren, Ocpn studio: Web-based modeling, simulation, and analysis of object-centric petri nets, Applications and Theory of Petri Nets and Concurrency (2026) 395–405.

[5] R. M. Dijkman, Simpn: A python library for modeling and simulating timed, colored petri nets, in: Proc. of BPM, CEUR Workshop Proceedings, CEUR-WS.org, 2024, pp. 71–75.


[1] https://youtu.be/X8T90W_m5MA
[2] https://github.com/TijmenKuijpers/OC-SimPN.git