

BPSimpy: A Python Library for WfMC-Standard Process-Simulation Specifications

Claudia Fracca^{1,2}, Angelica Bianconi¹, Francesca Meneghello¹,
Massimiliano de Leoni¹, Fabio Asnicar² and Alessandro Turco²

¹University of Padua, Italy

²ESTECO SpA, Trieste, Italy

Abstract

Business process simulation is a flexible technique that allows analysts to compare alternative scenarios and contribute to the analysis and improvement of business processes. This work presents BPSimpy, a business process simulation Python library, to generate a simulation specification file in a standard format that allows seamless integration with different simulators. BPSimpy can, e.g., be integrated with libraries for Data and Process Mining, thus bringing together the worlds of mining and business process simulation. For instance, event logs can be used to discover accurate business process simulation models consisting of a business process model and several simulation aspects that parametrize different process perspectives, such as the time perspective, the resource perspective, and the decision perspective.

Keywords

Business Process Simulation, Simulation Model, Business Process Simulation Standard (BPSim), Business Process Model and Notation (BPMN), Python.

1. Introduction


Business process simulation refers to techniques for the simulation of business process behavior on the basis of a *simulation model* consisting of a business process model extended with additional information for the definition of the different run-time *simulation aspects*: case arrival rate, task durations, routing probabilities, resource allocation and utilization, etc. Simulation provides a flexible approach to analyse and improve business processes [1]. The main idea of business process simulation is to generate a set of possible execution traces of a process, leveraging on the information a simulation model. The resulting execution traces allow companies to monitor the efficiency of their internal processes and to determine the critical aspects like bottlenecks, wastes, and over- and under-utilization of resources, and to verify the consequences of proposed process modifications before putting them in production. Through simulation experiments, various ‘what-if’ questions can be answered, and redesigning alternatives can be compared with respect to some key performance indicators.

Proceedings of the Demonstration Resources Track, Best BPM Dissertation Award, and Doctoral Consortium at BPM 2021 co-located with the 19th International Conference on Business Process Management, BPM 2021, Rome, Italy, September 6-10, 2021

✉ claudia.fracca@phd.unipd.it (C. Fracca); angelica.bianconi@studenti.unipd.it (A. Bianconi); francesca.meneghello.4@studenti.unipd.it (F. Meneghello); deleoni@math.unipd.it (M. d. Leoni); asnicar@esteco.com (F. Asnicar); turco@esteco.com (A. Turco)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

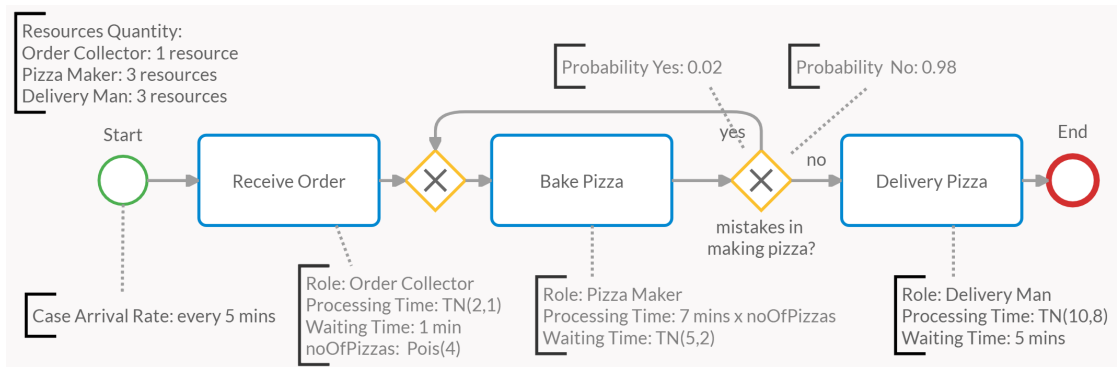


Figure 1: An example of BPMN model and some simulation parameters. Each task has a parameter indicating the role to be played by a resource to perform the task, and for each role the quantity of resources available are marked, and two time parameters, the processing time and the waiting time, parametrized as a constant duration or a truncated normal distribution with mean μ and standard deviation σ . Finally, the attribute 'noOfPizzas' is parametrized as a Poisson distribution with mean λ . For the start event and the sequences after the XOR gateway the case arrival rate and the conditional routing probabilities, parametrized with a constant value, are added respectively.

Typically, a business process simulation requires, as a first step, to extract a business process model from the process information (e.g. process documentation, stakeholders interviews, observational data, process data, event logs, etc.) such as a BPMN model [2] that covers the control-flow perspective (i.e. the ordering of activities) and may be enhanced with other perspectives, such as time, resource, and decision perspective [3]. The model is further extended by the definition of run-time simulation aspects, i.e. simulation parameters. The appropriate setup of these simulation aspects is often based on insights from process documentation, expert interviews and observations of the real process. These insights may be however inaccurate or, even, incorrect: they are indeed based on subjective opinions of process stakeholders and on the supposed process executions. It then follows that any conclusions drawn from these simulation results can be misleading or erroneous. Consequently, they can yield negative effects when the process is redesigned according to those conclusions. To analyze the actual process and not the supposed one, business process simulation should be merged with data and process mining: the simulation model should be derived from the actual objective facts recorded in the process event data, instead that from the subjective personal opinions [4].

This paper reports on BPSimpy, a business process simulation Python library to programmatically generate business simulation files. This way, one can write Python code that leverages on data- and/or process-mining libraries to obtain simulation models that accurately reflect the reality. In fact, the choice of Python is driven by the fact that this language is already equipped with libraries for process mining (e.g. PM4Py¹), machine learning, data science (e.g. Pandas, Numpy, Scipy, Tensorflow, Keras, etc.), and more.

One of the strongest points of BPSimpy is the ability to generate simulation models compliant with the Business Process Simulation Specification (BPSim), a standard format by the Workflow

¹<https://pm4py.fit.fraunhofer.de/>

```

1 import BPSimpy
2 example= BPSimpy.BPSim('BPMN.xml')
3 scenario=example.addScenario(id='S1', name='Pizza shop')
4 start=scenario.getElementParameters(example.getIdByName('Start'))
5 start.addInterTriggerTimer(value=datetime.timedelta(minutes=5))
6 receiveOrder=scenario.getElementParameters(example.getIdByName('Receive Order'))
7 receiveOrder.addProperty(name='noOfPizzas', nameDistribution='PoissonDistribution',
8     mean=4)
9 receiveOrder.addProcessingTime(nameDistribution='TruncatedNormalDistribution', min=0,
10     max=10, mean=2, standardDeviation=1)
11 receiveOrder.addWaitTime(value=datetime.timedelta(minutes=1))
12 bake=scenario.getElementParameters(example.getIdByName('Bake Pizza'))
13 bake.addProcessingTime(expression='`bpsim:getProperty(`noOfPizzas`)*7')
14 bake.addWaitTime(nameDistribution='TruncatedNormalDistribution', min=0, max=100, mean
15     =5, standardDeviation=2)
16 deliveryPizza=scenario.getElementParameters(example.getIdByName('Delivery Pizza'))
17 deliveryPizza.addProcessingTime(nameDistribution='TruncatedNormalDistribution', min=0,
18     max=120, mean=10, standardDeviation=8)
19 deliveryPizza.addWaitTime(value=datetime.timedelta(minutes=5))
20 YesSeq=scenario.getElementParameters(example.getIdByName('yes'))
21 YesSeq.addProbability(value=0.02)
22 NoSeq=scenario.getElementParameters(example.getIdByName('no'))
23 NoSeq.addProbability(value=0.98)
24 OrderCollector=getElementParameters(example.getIdByName('Order Collector'))
25 OrderCollector.addQuantity(value=1)
26 PizzaMaker=getElementParameters(example.getIdByName('Pizza Maker'))
27 PizzaMaker.addQuantity(value=3)
28 DeliveryMan=getElementParameters(example.getIdByName('Delivery Man'))
29 DeliveryMan.addQuantity(value=3)
30 receiveOrder.addSelection(expression='`bpsim:getResource(`OrderCollector`,1)')
31 bake.addSelection(expression='`bpsim:getResource(`PizzaMaker`,1)')
32 deliveryPizza.addSelection(expression='`bpsim:getResource(`DeliveryMan`,1)')

```

Figure 2: The Python code to implement, using BPSimpy library, the simulation scenario of the example in Fig. 1.

Management Coalition (WfMC) [5]. This enables a seamless integration with different simulation software products, such as Lanner² and Sparx³.

2. BPSimpy Library

The BPSimpy library takes a BPMN process model and the simulation parameters as input and generates a BPSim simulation model. The BPSim simulation model is organized in simulation scenarios. A simulation scenario specifies the initial conditions at the beginning of the situation and the different simulation parameters, necessary to set up a simulation run. Simulation parameters are also attached to the elements of the BPMN process model (events, tasks, gateways, sequences, etc.) to configure their simulation-time behavior. BPSimpy supports simulation parameters related to the various process perspectives: time, control, resources, cost, priority, and data. The definition of process' data objects is done via a set of process properties.

We describe the functionalities of the BPSimpy library through the example in Fig. 1. The

²<https://www.lanner.com/en-us/technology/l-sim-bpmn-simulation-engine.html>

³<https://sparxsystems.com/products/ea/trial/request.html>

different elements of the BPMN model are associated with some simulation parameters related to the perspectives on time, resources and control. Note how the example defines waiting times independently of the availability of resources: often, process tasks are not started as soon as they are enabled and resources are available, because resources work on multiple processes, take working breaks, execute duties that leave no trails in event logs, and other practical factors.

Fig. 2 shows the Python code to implement this simulation scenario. After importing the BPSimpy library (line 1), we call the constructor of a BPSim object passing the BPMN specification file (line 2). The next step is to create a scenario (method `addscenario()`, line 3). Once the scenario is created, the simulation parameters are associated to the different elements of the BPMN model, such as tasks, events, sequences, etc. The method `getElementParameters(id)` (line 4,6,10,...) returns an handle object to invoke the methods to associate the parameters to the BPMN element with a given `id`. If this is not known the method `getIdByName(name)` (line 4) allows one to retrieve it by the element name. Line 5 associates a control parameter to the start event: the `interTriggerTimer`, which models the case arrival rate. Line 7 sets a property parameter, that models the attribute 'noOfPizzas' for the first task *Receive Order*. In line 17 and 19 the routing probabilities for the XOR gateway are added through the method `addProbability()`. Lines 22 to 30 associate the quantity of resources to each roles (method `addquantity()`). Each task is linked to the corresponding role that resources need to be able to play to perform it (method `addSelection()`).

Note how BPSim's simulation parameters are typed according to their nature and necessary configuration, which our library BPSimpy fully supports: constants, distributions, enumerations, and XPATH 1.0 expressions (see the BPMSim guide for details [5]). Our library allows setting up these parameters invoking methods of the same name. Calendars are also supported (cf. Section 3). It is also possible to request to a specific process model element a result request attribute to have in the output, e.g. the resulting processing time of a specific process task or the number of times a task is performed, (see [5] for the applicability).

3. A Case Study

This section reports a case study of a real business process. The event log is taken from the Second International Business Process Intelligence Challenge (BPI2012).⁴ The event log refers to an application process for a personal loan or overdraft within a global financing organization. The event log is a merger of three intertwined sub processes and we decided to showcase on an individual subprocess (subprocess A). We leveraged on the Pm4Py library and other statistic libraries in Python to first mine a fitting, precise BPMN process model and later complement with the other simulation parameters, namely: (i) the probability distribution function of the inter arrival rate, considering the case-arrival calendar from 8am to 11pm; (ii) the branching probabilities for each XOR split in the process model; (iii) the probability distributions of the processing time of every model task; (iv) the set of roles with the resources belonging to each; (v) the working calendar hours for each role, and (vi) the mapping of each activity *a* to each role that resources must play to be able to perform *a*. Once we discovered the BPMN process model and the simulation parameters, we employed our BPSimpy library to write a BPSim simulation

⁴<https://www.win.tue.nl/bpi/doku.php?id=2012:challenge>

file. As last step, this file was given as input to the Lanner simulator (L-sim) to carry out the simulation. L-sim returns a zip file as output that contains the simulated event log and a file with the simulation statistics of interests. The result-request configuration was employed through our library to analyze the performance the queue length and the waiting time for each task. This analysis can be useful to individuate process improvements. Note that the generation of an event-log file in XES format is supported by the Lanner simulator through a so-called vendor extension, a concept supported by the BPSim and configurable via our BPSimpy library.

Further information on the case study is available in the library Github repository.⁵ The repository also contains a tutorial based on a Jupiter Notebook on Google Colab that presents the case study along with a screencast demonstrating the usage of the BPSimpy library. This way, the Github repository incorporates a collaborative system where users can download and use the BPSimpy library, report problems and contribute to the code.

4. Conclusion

This paper reports on BPSimpy, a mature Python library that allows for programmatically creating business process simulation models in BPSim, a WfMC standard supported by multiple simulators. The library and the underneath standard supports several simulation parameters to enhance incorporate information related to time, control, resources and decision. The library is meant to enable the integration of the Process- and Data-Mining worlds to create accurate simulation models. These models enables monitoring processes, determining such critical aspects as resource over- and under-use, bottlenecks, cost and time wasting, to provide actionable insights to enable for a subsequent process improvement.

Acknowledgement. This research work is partly supported by the Department of Mathematics of University of Padua through the SID/BIRD 2020 project “Deep Graph Memory Networks”.

References

- [1] M. Dumas, M. La Rosa, J. Mendling, H. A. Reijers, *Fundamentals of Business Process Management*, 2nd ed., Springer Publishing Company, Incorporated, 2018.
- [2] BPMN 2013: ISO/IEC International Standard 19510: Information Technology – Object Management Group Business Process Model and Notation, Document Number ISO/IEC 19510:2013(E), 2013.
- [3] W. M. P. van der Aalst, *Process Mining: Data Science in Action*, 2nd ed., Springer Publishing Company, Incorporated, 2016.
- [4] N. Martin, B. Depaire, A. Caris, The use of process mining in business process simulation model construction, *Business & Information Systems Engineering* (2016) 73–87.
- [5] BPSim 2013: Workflow Management Coalition: BPSim– Business Process Simulation Specification, Document Number WFMC -BPSWG-2012-1, 2013.

⁵The Github Repository of BPSimpy - <https://github.com/claudiafracca/BPSimpyLibrary.git>