PySPN: An Extendable Python Library for Modeling & Simulation of Stochastic Petri Nets

Jonas Friederich[1(✉)] and Sanja Lazarova-Molnar[1,2]

[1] Mærsk Mc-Kinney Møller Institute, University of Southern Denmark, Odense, Denmark
jofr@mmmi.sdu.dk

[2] Institute AIFB, Karlsruhe Institute of Technology, Karlsruhe, Germany
lazarova-molnar@kit.edu

Abstract. Stochastic Petri Nets (SPNs) are a powerful formalism, widely used for modeling complex systems in various domains, ranging from manufacturing and logistics to healthcare and computer networks. In this paper, we introduce *PySPN*, a flexible and easily extendable *Python* library for Modeling & Simulation (M&S) of SPNs. *PySPN* aims to provide researchers, engineers, and simulation practitioners with a user-friendly and efficient toolset to model, simulate, and analyze SPNs, facilitating the understanding and optimization of stochastic processes in dynamic systems.

Keywords: Stochastic Petri nets · Modeling & Simulation · Python

1 Introduction

SPNs are a popular formalism for system M&S due to their ability to capture concurrency, synchronization, and stochastic behavior. SPNs are an extension of Petri nets, developed in the 1960s by Carl Adam Petri, and are used today in many domains such as manufacturing [12] and healthcare [14]. Existing tools for SPN M&S, however, often suffer from usability issues and limited extensibility.

GUI-based solutions like *GreatSPN* [1], *Oris* [9], or *CPN Tools* [11] offer user-friendly interfaces and facilitate quick modeling of Petri nets. However, they lack the flexibility and customization required for specific use cases. Conversely, command-line-interface-based solutions, such as *SNAKES* [10], *gspn-framework* [2], or *PNet* [4], are often domain-specific or have complex software architectures.

PySPN [6] fills this gap by offering an open-source, easy-to-use, and extensible library that integrates seamlessly with the *Python* ecosystem, empowering users to efficiently explore and analyze stochastic processes in a wide range of applications.

Our previous work on data-driven Digital Twins [7] and data-driven reliability modeling [8] in the manufacturing domain motivated the development of

J.-L. Guisado-Lizar et al. (Eds.): SIMUtools 2023, LNICST 519, pp. 70–78, 2024.
https://doi.org/10.1007/978-3-031-57523-5_6

this library. In the referenced work, we demonstrate the potential of SPNs to be used in combination with techniques such as Process Mining for data-driven model extraction and continuous validation.

The remainder of this paper is organized as follows: In Sect. 2, we provide a comprehensive technical overview of *PySPN*. Subsequently, in Sect. 3, we illustrate the application of our developed library using a two-server queuing system. Lastly, in Sect. 4, we summarize this work and discuss potential future expansions of *PySPN*.

2 Library Overview

Our library focuses on discrete-event systems, where system changes occur at discrete points in time as a result of completing particular activities in the system. These activities are associated with transitions, which can either fire instantly when certain conditions are met (immediate transitions) or have firing delays determined by probability distribution functions (timed transitions). Beyond transitions, a SPN comprises a finite number of places, a finite number of arcs, and an initial state, denoted by the initial marking of the Petri Net. Formally, the class of SPNs that can be modeled using the library is defined as:

$$SPN = (P, T, A, G, m_0)$$

where:

- $P = \{P_1, P_2, .., P_m\}$ is the set of places, drawn as circles;
- $T = \{T_1, T_2, .., T_n\}$ is the set of transitions along with their distribution functions or weights, drawn as bars;
- $A = A^I \cup A^O \cup A^H$ is the set of arcs, where A^O is the set of output arcs, A^I is the set of input arcs and A^H is the set of inhibitor arcs and each of the arcs has a multiplicity assigned to it;
- $G = \{g_1, g_2, .., g_r\}$ is the set of guard functions which are associated with different transitions;
- and m_0 is the initial marking, defining the distribution of tokens in the places.

Each transition is represented as $T_i = (type, F)$, where $type \in \{timed, immediate\}$ indicates the type of the transition, and F is either a probability distribution function if the corresponding transition is timed, or a firing weight or probability if it is immediate. *PySPN* currently supports empirical distributions, as well as several theoretical distributions such as *exponential*, *normal*, and *Weibull*. The sets of arcs are defined such that

$$A^O = \{a_1^o, a_2^o, ..., a_k^o\},\ A^I = \{a_1^i, a_2^i, ..., a_j^i\} \text{ and } A^H = \{a_1^h, a_2^h, ..., a_i^h\}$$

where:

$$A^H, A^O \subseteq P \times T \times \mathbb{N}^+ \text{ and } A^I \subseteq T \times P \times \mathbb{N}^+.$$

Figure 1 illustrates the directory structure of *PySPN*. *spn.py* defines the necessary classes for constructing a SPN, including places, transitions, input and output arcs, inhibitor arcs, guard functions, memory policies, and the SPN itself. *spn_simulate.py* implements the SPN simulation algorithm based on the discrete-event simulation paradigm [3]. In order to retrieve firing times for timed transitions during simulation, *RNGFactory.py* includes a factory method returning a random sample of a probability distribution function. We have employed the scientific computing library *scipy* [13] for defining distribution functions and obtaining random samples. *scipy* defines theoretical distribution functions with a consistent set of parameters, making it easy to extend our tool with other, not yet implemented, distributions. Furthermore, *scipy*'s *.ecdf* function can be employed to define empirical cumulative distribution functions based on underlying data samples. *spn_io* defines methods for printing a textual description of a SPN, printing statistics, printing a simulation protocol, and importing and exporting SPNs using *Python's pickle* format. Lastly, *spn_visualization.py* includes a method to create a graphical representation of a SPN using the *graphviz* [5] library. For further information about the tool and how to get started, we refer the reader to the documentation on *GitHub* [6].

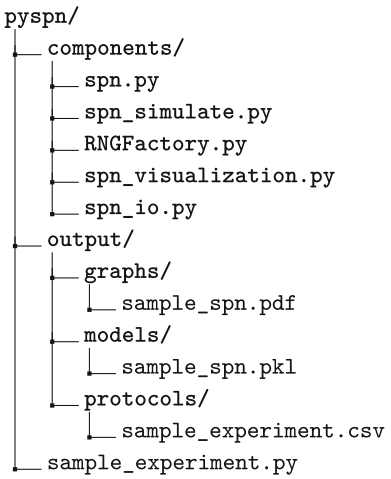

Fig. 1. Directory structure of *PySPN*.

3 Application Example

To illustrate the capabilities of *PySPN*, we use the popular model of a two-server queueing system (Fig. 2). In this particular model, *Server1* operates at a faster speed than *Server2*, but *Server2* has the capability to process two entities simultaneously, whereas *Server1* can only process one entity at a time. A guard

function is implemented on the *TApproach_Server2* immediate transition, only enabling the transition if there are at least two entities present in the queue.

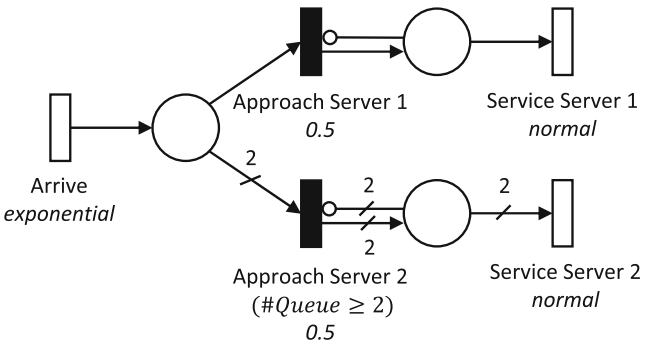

Fig. 2. SPN of a two-server queueing system.

By modeling, simulating, and analyzing the described model, several insights into the performance and behavior of the system can be gained, for example:

- **Performance Metrics:** throughput (rate at which entities are served and leave the system), queue length, and waiting times.
- **Resource Utilization:** utilization of each server (fraction of time a server is busy serving entities).
- **Sensitivity Analysis:** by varying the arrival times, service times, and number of servers the impact of these parameters on the system can be analyzed. This can help in optimizing resource allocation to minimize queue lengths, waiting times, or other performance metrics.
- **Bottleneck Identification:** Bottlenecks (model components that limit the overall system performance) can be identified and potentially mitigated.

In Listing 1 we show, how to implement the SPN for the described model using *PySPN*. The process of modeling a SPN follows a clear structure: After importing the required components (lines 1–3), a SPN object is instantiated (line 5). Then, places and transitions are instantiated (lines 7–25) and added to the SPN object (lines 27–35). Next, the places and transitions are connected by arcs (lines 37–45). Finally, the modeled SPN can be simulated (line 47). The simulation function (i.e., *simulate*) takes as arguments the SPN object, the maximum time to run the simulation, the output verbosity, and whether to write a protocol file to store the marking for each discrete state transition during the simulation.

```
1 from components.spn import *
2 from components.spn_simulate import simulate
3 from components.spn_visualization import draw_spn
```

```
4
5  spn = SPN()
6
7  p1 = Place(label="Queue", n_tokens=0)
8  p2 = Place(label="Server1", n_tokens=0)
9  p3 = Place(label="Server2", n_tokens=0)
10
11 t1 = Transition(label="Arrive", t_type="T")
12 t1.set_distribution(distribution="expon", a=0.0, b=1.0/1.8)
13 t2 = Transition(label="Approach Server 1", t_type="I")
14 t2.set_weight(weight=0.5)
15 t3 = Transition(label="Approach Server 2", t_type="I")
16 def guard_t3():
17     if p1.n_tokens >= 2:
18         return True
19     else: return False
20 t3.set_weight(weight=0.5)
21 t3.set_guard_function(guard_t3)
22 t4 = Transition(label="Service Server 1", t_type = "T")
23 t4.set_distribution(distribution="expon", a=0.0, b=1.0/0.9)
24 t5 = Transition(label="Service Server 2", t_type = "T")
25 t5.set_distribution(distribution="expon", a=0.0, b=1.0/0.9)
26
27 spn.add_place(p1)
28 spn.add_place(p2)
29 spn.add_place(p3)
30
31 spn.add_transition(t1)
32 spn.add_transition(t2)
33 spn.add_transition(t3)
34 spn.add_transition(t4)
35 spn.add_transition(t5)
36
37 spn.add_output_arc(t1,p1)
38 spn.add_input_arc(p1,t2)
39 spn.add_input_arc(p1,t3,multiplicity=2)
40 spn.add_output_arc(t2,p2)
41 spn.add_output_arc(t3,p3,multiplicity=2)
42 spn.add_input_arc(p2,t4)
43 spn.add_inhibitor_arc(t2,p2)
44 spn.add_input_arc(p3,t5,multiplicity=2)
45 spn.add_inhibitor_arc(t3,p3,multiplicity=2)
46
47 simulate(spn, max_time = 100, verbosity = 2, protocol = True)
```

Code Listing 1. Two-server queue example.

We executed a simulation run of the described two-server queuing model using the arguments shown in Listing 1 (line 47). As can be seen form the listing, the simulation time is set to be 100 time units, the output verbosity is set to be 2 and a protocol file is to be written and stored.

In Fig. 3, we display an excerpt of the generated terminal output during the simulation. At a verbosity level of 2, basic information regarding the simulation run, such as the initial marking and the firing of transitions is returned. The initial marking describes the distributions of tokens in each of the three places (i.e., 0 for *Queue*, *Server 1*, and *Server 2*).

```
Starting simulation...
Simulation time limit = 100

Marking at time 0
Place Queue, #tokens: 0
Place Server 1, #tokens: 0
Place Server 2, #tokens: 0

Transition Arrive fires at time 0.08
Transition Approach Server 1 fires at time 0.08
Transition Service Server 1 fires at time 0.45
Transition Arrive fires at time 0.56
Transition Approach Server 1 fires at time 0.56
Transition Arrive fires at time 0.59
Transition Arrive fires at time 0.83
Transition Approach Server 2 fires at time 0.83
Transition Arrive fires at time 0.87
Transition Arrive fires at time 0.89
...
```

Fig. 3. Excerpt of the terminal output for the simulation run.

Table 1 shows an excerpt of the generated simulation protocol file. The protocol shows the changes of the markings at discrete points in time for the three places throughout the simulation. For this two-server queuing model, the marking of the place *Queue* corresponds to the number of entities waiting in the queue and the marking of the places *Server 1*, and *Server 2* corresponds to the number of entities being processed by the servers (i.e., either 0 or 1 for *Server 1* and 0 or 2 for *Server 2*).

Based on the generated protocol, Fig. 4, displays a graph depicting the discrete changes in queue size (i.e., marking of the place *Queue*) throughout the simulation.

4 Summary and Outlook

In this paper, we introduced *PySPN*, a *Python* library for M&S of SPNs. *PySPN* aims to provide a user-friendly and extensible toolset for researchers, engineers, and simulation practitioners to model, simulate, and analyze SPNs, facilitating the understanding and optimization of stochastic processes in dynamic systems.

PySPN offers a comprehensive set of features, including support for immediate and timed transitions, various probability distribution functions, inhibitor

Table 1. Excerpt of the generated protocol file.

Place	Time	Marking
...	...	...
Queue	5.6	2
Queue	5.6	0
Server 2	5.6	0
Server 2	5.6	2
Queue	6.05	0
Queue	6.05	1
Server 1	6.15	1
Server 1	6.15	0
Queue	6.15	1
Queue	6.15	0
Server 1	6.15	0
Server 1	6.15	1
...	...	...

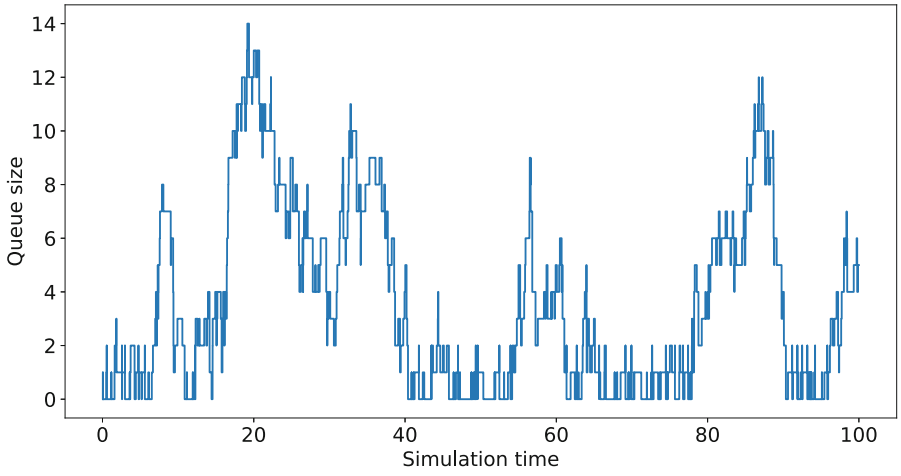

Fig. 4. Changes in queue size throughout the simulation.

arcs, guard functions, and more. It integrates seamlessly with the *Python* ecosystem, making it easy to extend and adapt for specific use cases.

Looking ahead, PySPN has potential for further development and expansion. Some avenues for future work include:

- **Parallel Simulation:** Implementing parallel and distributed simulation techniques to enable the efficient simulation of large-scale and highly concurrent systems.
- **Advanced Analysis Tools:** Developing additional tools and libraries for advanced analysis of SPNs, including sensitivity analysis, optimization algorithms, and statistical inference.
- **Enhanced Visualization:** Improving the visualization capabilities of PySPN to provide more detailed and customizable representations of SPNs, facilitating better insights into model behavior.

Furthermore, we are aiming to provide a more comprehensive review and comparison of related tools for SPN M&S in the future.

References

1. Amparore, E.G., Balbo, G., Beccuti, M., Donatelli, S., Franceschinis, G.: 30 years of GreatSPN. In: Fiondella, L., Puliafito, A. (eds.) Principles of Performance and Reliability Modeling and Evaluation. SSRE, pp. 227–254. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30599-8_9
2. Azevedo, C.: cazevedo/gspn-framework. https://github.com/cazevedo/gspn-framework
3. Banks, J., Carson, J., II., Nelson, B., Nicol, D.: Discrete-Event System Simulation, 5th edn. Pearson, Upper Saddle River (2009)
4. Chay, Z.E., Goh, B.F., Ling, M.H.: PNet: a Python library for Petri net modeling and simulation, February 2023. https://doi.org/10.48550/arXiv.2302.12054
5. Ellson, J., Gansner, E., Koutsofios, L., North, S.C., Woodhull, G.: Graphviz—open source graph drawing tools. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 483–484. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45848-4_57
6. Friederich, J.: PySPN (2023). https://github.com/jo-chr/pyspn
7. Friederich, J., Francis, D.P., Lazarova-Molnar, S., Mohamed, N.: A framework for data-driven digital twins of smart manufacturing systems. Comput. Ind. **136**, 103586 (2022). https://doi.org/10.1016/j.compind.2021.103586
8. Friederich, J., Lazarova-Molnar, S.: Data-driven reliability modeling of smart manufacturing systems using process mining. In: 2022 Winter Simulation Conference (WSC), pp. 2534–2545, December 2022. https://doi.org/10.1109/WSC57314.2022.10015301
9. Paolieri, M., Biagi, M., Carnevali, L., Vicario, E.: The ORIS tool: quantitative evaluation of non-Markovian systems. IEEE Trans. Software Eng. **47**(6), 1211–1225 (2021). https://doi.org/10.1109/TSE.2019.2917202
10. Pommereau, F.: SNAKES: a flexible high-level Petri nets library (tool paper). In: Devillers, R., Valmari, A. (eds.) PETRI NETS 2015. LNCS, vol. 9115, pp. 254–265. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19488-2_13
11. Ratzer, A.V., et al.: CPN tools for editing, simulating, and analysing coloured Petri nets. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 450–462. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-44919-1_28
12. Tüysüz, F., Kahraman, C.: Modeling a flexible manufacturing cell using stochastic Petri nets with fuzzy parameters. Expert Syst. Appl. **37**(5), 3910–3920 (2010). https://doi.org/10.1016/j.eswa.2009.11.026

13. Virtanen, P., et al.: SciPy 1.0: fundamental algorithms for scientific computing in Python. Nat. Methods **17**(3), 261–272 (2020). https://doi.org/10.1038/s41592-019-0686-2
14. Wang, J.: Patient flow modeling and optimal staffing for emergency departments: a Petri net approach. IEEE Trans. Comput. Soc. Syst. **10**(4), 2022–2032 (2023). https://doi.org/10.1109/TCSS.2022.3186249