Proceedings of the 2025 Winter Simulation Conference
E. Azar, A. Djanatliev, A. Harper, C. Kogler, V. Ramamohan, A. Anagnostou, and S. J. E. Taylor, eds.

A TUTORIAL ON DATA-DRIVEN PETRI NET MODEL EXTRACTION AND SIMULATION FOR DIGITAL TWINS IN SMART MANUFACTURING

Atieh Khodadadi[1], Ashkan Zare[2], Michelle Jungmann[1], Manuel Götz[1], and Sanja Lazarova-Molnar[1,2]

[1]Institute AIFB, Karlsruhe Institute of Technology, Karlsruhe, GERMANY
[2]The Maersk Mc-Kinney Moller Institute, University of Southern Denmark, Odense, DENMARK

ABSTRACT

The adoption of data-driven Digital Twins in smart manufacturing systems necessitates robust, data-driven modeling techniques. Stochastic Petri nets offer a formal framework for capturing concurrency and synchronization in discrete-event systems, making them well-suited for modeling smart manufacturing systems. This tutorial provides a hands-on introduction to extracting stochastic Petri net models from system event logs. Using our Python-based stochastic Petri nets library (PySPN), participants will learn how to: (1) define ground-truth models using the Petri Net Markup Language (PNML), (2) generate event logs by simulating these models, and (3) apply process mining techniques to automatically reconstruct stochastic Petri net models from real or simulated data. Two case studies illustrate the end-to-end workflow, from data generation to model validation, within the context of developing Digital Twins. By the end of the tutorial, participants will gain practical skills in data-driven stochastic Petri nets modeling for Digital Twins applications in manufacturing.

1 INTRODUCTION

The increasing adoption of Smart Manufacturing Systems (SMSs) is driving demand for robust modeling and simulation methods that can accurately represent system behavior and support data-driven decision-making. Digital Twins (DTs) are increasingly used to represent manufacturing systems virtually, enabling real-time monitoring, analysis, and optimization based on system data (Soori et al. 2023). Among the different modeling formalisms applicable to DTs, Petri Nets (PNs) are particularly well-suited due to their ability to capture concurrency, synchronization, and stochastic behavior in discrete-event systems (Desel and Reisig 2015; Peterson 1981). This combination of capabilities aligns directly with the operational characteristics of manufacturing systems. Real-world production environments frequently involve parallel processes, dependencies among tasks, and stochastic variations in processing times, arrivals, and failures. SPNs provide a formal and simulation-compatible language to represent such behavior by utilizing system data, enabling DTs to mirror the dynamics of the physical system. Furthermore, PNs have an intuitive graphical representation, which facilitates communication and collaboration with domain experts who may not have a background in computer science. This visual clarity supports model validation and co-development in interdisciplinary teams, which is often essential in manufacturing environments.

A challenge in developing accurate DT models is the automated and scalable extraction of these models from system data. Traditional modeling approaches often rely heavily on substantial expert knowledge and manual construction, which limits their applicability in dynamic and data-rich environments (Jungmann and Lazarova-Molnar 2024). To address these limitations, Process Mining (PM) has been increasingly adopted as a technique to derive behavioral models from system-generated event logs (Van Der Aalst 2011). By analyzing sequences of recorded events, PM techniques enable the reconstruction of process models that reflect actual systems' behaviors.

Stochastic Petri Nets (SPNs) offer a formalism capable of representing complex behaviors. They extend classical PNs with stochastic timing information, allowing for performance analysis under uncertainty.

However, integrating SPNs with PM workflows to support the extraction of DT models remains an open challenge. Existing PM tools are primarily focused on extracting control-flow models for business process management and often lack support for advanced PN constructs that are critical for simulation and validation in manufacturing contexts, such as inhibitor arcs and guard conditions.

This tutorial provides a hands-on guide for extracting simulation-ready SPN models from event logs and validating the models for use in DT applications. Participants are guided through each stage of the workflow: beginning with the definition of a ground-truth model, followed by the generation of synthetic event logs, the application of PM techniques for model extraction, and concluding with validation of the discovered model against the ground-truth. The tutorial is supported by our open-source Python library, PySPN (Friederich et al. 2025), which facilitates both the simulation of SPN models and the generation of event logs in a format suitable for PM.

The remainder of the tutorial paper is organized as follows: Section 2 provides background on the key concepts and technologies, including DTs, PM, SPNs, PNML, and relevant Python-based tools. Section 3 introduces SPNs as a modeling formalism for DTs in manufacturing. Section 4 details the tutorial setup, including ground-truth model generation, SPN model extraction, and validation of extracted models. Section 5 demonstrates the workflow through two case studies. Section 6 concludes the tutorial.

2 PRELIMINARIES

In this section, we provide the key concepts and tools that form the foundation of this tutorial. We begin by providing an overview of DTs in manufacturing, emphasizing their relevance and potential impact. Next, we explore PM as an essential technique for extracting models from event data, followed by an introduction to SPNs, which serve as a suitable formalism for representing the relevant models. We then describe PNML, the standardized format used for describing PN models. Finally, we provide an overview of the Python libraries that we use throughout the tutorial to implement the workflow.

2.1 Digital Twins in Manufacturing

DTs have gained attention in both academia and industry since the concept was formally introduced by Grieves in 2002 (Grieves and Vickers 2017). NASA further adapted the concept and refined the term "Digital Twin", focusing on the aerospace domain in 2010 (Piascik et al. 2010). In 2021, ISO 23247 was issued by the International Organization for Standardization as a DT manufacturing framework (International Organization for Standardization 2021). Despite these developments, a universally accepted definition of DTs remains lacking (Liu et al. 2023).

We define a DT as a virtual representation of a physical entity, for example, a smart factory, where both components are connected through bidirectional communication (VanDerHorn and Mahadevan 2021), as shown in Figure 1. The bidirectional communication link facilitates data collection, validation, knowledge extraction, and model validation, and furthermore, is the component that distinguishes a DT from the related concepts of Digital Models and Digital Shadows (Kritzinger et al. 2018). DTs are utilized in various areas such as manufacturing, smart cities, healthcare, automotive, renewable energy, and aeronautics (Fuller et al. 2020; Liu et al. 2023; Soori et al. 2023).

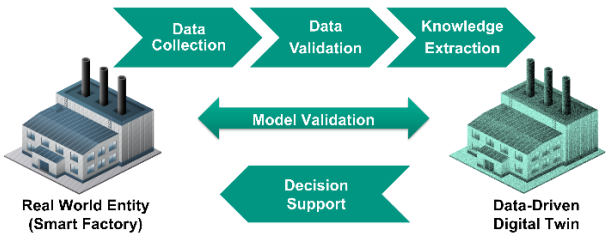

Figure 1: Constituent parts of a Digital Twins setup for Smart Manufacturing Systems (Khodadadi and Lazarova-Molnar 2023).

In the domain of manufacturing, SMSs, enabled by Industry 4.0, have become increasingly important as manufacturing processes have increased in their complexity, frequency of demand modifications, customization requirements, and interconnections (Leng et al. 2021; Tao et al. 2019). In SMSs, technologies such as IoT, software, artificial intelligence, automation, or big data are utilized to improve performance, quality, and efficiency within manufacturing processes. With this, the aim of SMSs is to enable highly adaptable environments that can swiftly respond to changing demands while maintaining high product quality and reducing production costs. To achieve this aim, both manufacturing processes and resources are digitally represented and managed in virtual environments (Soori et al. 2023; Leng et al. 2021).

DTs enhance SMSs by providing virtual replicas of physical systems that can be continuously updated with (near) real-time IoT data (Leng et al. 2021; Tao et al. 2019). DTs empower SMSs through capabilities such as monitoring, simulation, visualization, analysis, and optimization (Soori et al. 2023). Furthermore, DTs support more informed decision-making, reduce costs, reduce lead times, and improve efficiency and performance of production processes in terms of throughput and downtime (Kunath and Winkler 2018; Fuller et al. 2020; Soori et al. 2023; Greco et al. 2020). DTs are also utilized to track, test, and manage production systems as well as provide failure diagnosis (He et al. 2019). Another benefit of DTs in SMSs is the ability to execute virtual simulations of what-if scenarios without disrupting the system (Soori et al. 2023). Given their broad functionality, DTs can be implemented with different focuses, such as product monitoring, safety enhancement, downtime optimization, time-to-market reduction, and virtual commissioning (Soori et al. 2023). DTs can be further applied in different manufacturing phases such as design, (re-)configuration, and operation (Soori et al. 2023).

Numerous studies have explored applications of DTs in the context of SMSs. For instance, Leng et al. (2021), Friederich et al. (2022), Lazarova-Molnar (2024), and Mahmoud and Grace (2019) explore the implementation of DTs within SMSs. In our earlier work, we developed and extended a conceptual framework for data-driven DTs of SMSs (Friederich et al. 2022; Lazarova-Molnar 2024), shown in Figure 2. The framework defines the required components to extract SPNs as DT models of SMSs, which we demonstrate in this tutorial. The framework further shows how the real-world and its digital replications are interconnected. In this framework, the SMS is the real-world entity connected to a corresponding data-driven DT. The data-driven DT possesses an underlying SPN model that is automatically extracted from IoT data from the manufacturing system, in our case, event logs, and delivers decisions back to the real-world entity. The data-driven DT consists of six components: data collection, data validation, knowledge extraction, model development, model validation, and analysis, which aim to extract an underlying SPN model for the data-driven DT model and provide the DT capabilities that support decisions for the SMS.

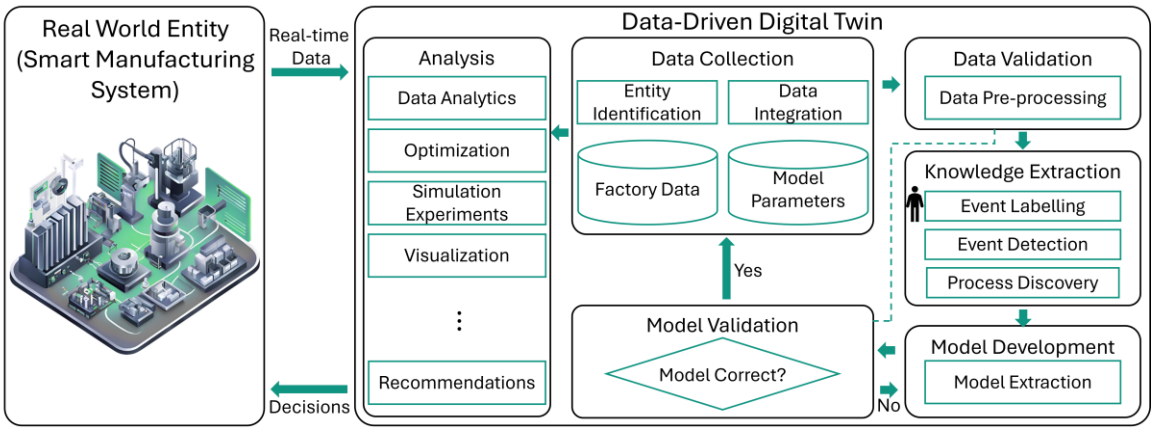

Figure 2: Framework for data-driven Digital Twins of SMSs (Lazarova-Molnar 2024).

2.2 Process Mining

PM enables the extraction of structured process models and performance insights from event logs readily available in modern production systems. PM aims to provide objective, data-driven insights into how processes are executed rather than how they are assumed to work (Van Der Aalst 2011).

Event logs form the foundation of PM, capturing ordered sequences of events typically annotated with a case ID, activity name, and timestamp. In the following, we provide a formal description of an event log (Friederich et al. 2025). An event log EL is defined as a set of event entries $EL = \{E_0, E_i, \dots, E_m\}, i = 1, \dots, m$. Each event log entry is defined as a tuple $E_i = (t, o, e)$, where:

- Timestamp (t): The exact point in time of event occurrence.
- Case ID (o): A unique identifier for the trace to which the event belongs.
- Event Identifier(e): Unique string that indicates the type of action that occurred in the beginning or end of an activity.
- The ordering of events in a trace ensures that if e_i and e_j are two events in EL where $i < j$, then $t(e_i) \leq t(e_j)$, maintaining a non-decreasing order of timestamps in a trace.

By analyzing event logs, PM techniques can reconstruct control flows of processes, evaluate processes' conformance to expected behaviors, and enrich models with performance-related data. These capabilities have led to the application of PM in domains such as manufacturing, healthcare, logistics, and business process management, where understanding real operational behavior is critical (Van Der Aalst 2011; Augusto et al. 2019). PM techniques are typically grouped into three main categories (Van Der Aalst 2011):

- **Process discovery**: This technique constructs a process model based on the data contained in the event log, without using any prior model. Algorithms such as Alpha Miner, Heuristic Miner, and Inductive Miner are commonly used to detect patterns and control-flow relations like sequences, choices, and parallelism (Augusto et al. 2019).
- **Conformance checking**: This technique compares an existing process model to an event log of the same process to identify deviations. It is particularly useful for auditing, compliance verification, and detecting discrepancies between prescribed and actual process executions.
- **Enhancement**: Also known as performance mining, this technique enhances a process model with additional information derived from the event log, such as activity durations, waiting times, and resource utilization. This enables a combined structural and performance-oriented analysis.

The effectiveness of PM relies on the availability of high-quality event data that accurately captures the underlying process behavior. Incomplete, noisy, or inconsistent data can result in incorrect or overly complex models. Moreover, the level of granularity must correspond to the logical structure of the process: insufficient detail may obscure control-flow dependencies, while excessive detail can introduce unnecessary complexity. Appropriate preprocessing is therefore essential to ensure the validity and interpretability of the extracted models (Van Der Aalst 2011).

We highlight three representative studies to demonstrate the adaptability and practical value of PM across diverse manufacturing scenarios. Friederich and Lazarova-Molnar (2021) extract reliability models from event data in a drone assembly lab using custom trace extraction and statistical parameterization. Lorenz et al. (2021) improve productivity in a make-to-stock environment by identifying inefficiencies through conformance checking and event log analysis with Disco and Python scripts. Birk et al. (2021) integrate ERP and SCADA data to analyze a multi-level valve production process, addressing challenges like hierarchical modeling and data heterogeneity. These studies underscore the versatility of PM in addressing a range of industrial challenges, including reliability, productivity, and system integration.

2.3 Stochastic Petri Nets

PNs have been utilized for modeling, simulating, and assessing the performance of discrete event systems. The broad adoption of PNs is attributed to their features, such as visual (graphical) representation that closely mirrors real-world objects and their clear, formally established rules. Built on basic mathematical concepts (primarily basic algebra and graph theory), PNs facilitate comprehensive system analysis, enabling the examination of the system's entire behavior, pinpointing performance bottlenecks, and avoiding deadlocks (Peterson 1981). In our work, we use the stochastic variant of PNs, SPNs. SPNs extend classical PNs by incorporating stochastic timing information into transitions, enabling the modeling of temporal uncertainty and facilitating performance analysis in systems characterized by probabilistic behavior. The following provides the formal description of SPNs as outlined by Lazarova-Molnar (2005). An SPN is defined as $SPN = (P, T, A,\ G, m_0)$, where:

- P is a finite set of places, drawn as circles, $P = \{P_1, P, \dots, P_m\}$;
- T is a finite set of transitions along with their distribution functions or weights, shown as bars, $T = \{T_1, T_2, \dots, T_n\}$;
- A is a set of arcs $A = A^I \cup A^0 \cup A^H$. The sets of arcs are defined as input arcs $A^I = \{a_1^i, a_2^i, \dots, a_j^i\}$ that connect places to transitions; output arcs $A^0 = \{a_1^o, a_2^o, \dots, a_k^o\}$ that connect transitions to places, and inhibitor arcs $A^H = \{a_1^h, a_2^h, \dots, a_i^h\}$ that prevent a transition from firing if a certain number of tokens are present in its connected place. Each arc has a multiplicity assigned to it, and $A^I \subseteq (P \times T \times N)$ and $A^H, A^I \subseteq (P \times T \times N)$;
- G is the set of guard functions that are associated with different transitions, $G = \{g_1, g_2, \dots, g_r\}$;
- And m_0 is the initial marking of the PN.

To prepare SPNs for integration with PM, we define a transition as a tuple $T_i = (e, f, type)$, linked to an event $\{E.e\}$ for every trace $E \in EL$. Here, e supplies the transition's label, and f stores either the probability distribution function (when the transition is timed) or the firing weight (when it is immediate). The last element, $type$, specifies whether the transition is *timed* or *immediate*. One of the three conditions for a transition in a PN to be enabled is that all input places contain at least as many tokens as their arc multiplicities. The other two conditions are that no guard functions or inhibitor arcs are blocking the transition. Inhibitor arcs are used in PNs to represent conditions where the presence of one or more tokens in a place prevents a transition from firing. Multiplicity of an arc defines how many tokens are destroyed or created by a corresponding transition, supporting batch operations or resource usage. Guard functions attached to a transition define logical conditions that must evaluate to true for the transition to be enabled. These functions serve as additional constraints, allowing transitions to fire only when specific contextual criteria are met. Firing an enabled transition destroys tokens from input places and creates tokens in output places, according to arcs multiplicities, and correspondingly updates the marking of the net (from marking M to M' through the firing of a transition).

In Figure 3, we illustrate the four basic control-flow patterns for representing workflow models. PN(a) captures a simple sequence $(a \rightarrow b)$, PN(b) represents an XOR-split pattern ($a > b, a > c$ and $b \# c$), PN(c) represents an AND-split pattern ($a > b, a > c$ and $b || c$), and PN(d) represents an AND-join pattern ($a || b \rightarrow c$). A detailed discussion of the algorithm is provided in the work of van der Aalst et al. (2004).

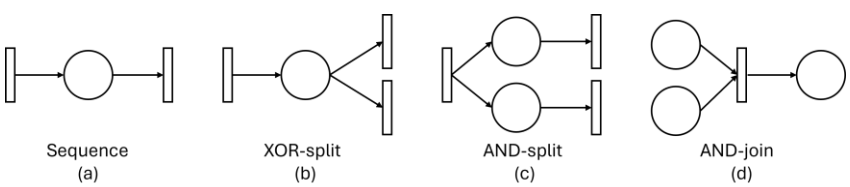

Figure 3: Examples of foundational Petri net control-flow patterns.

```
<?xml version="1.0" encoding="UTF-8"?>
<pnml xmlns="http://www.pnml.org/version-2009/grammar/pnml">
  <net id="SimplePNML" type="http://www.pnml.org/version-2009/grammar/ptnet">
    <name>
      <text>A Simple Petri net in PNML </text>
    </name>
    <page id="page1">
      <!-- Places -->
      <place id="P1">
        <name><text>A place</text></name>
        <initialMarking><text>0</text></initialMarking>
      </place>
      <!-- Transitions -->
      <transition id="T1" type="T">
        <name><text>A transition</text></name>
        <distribution>
          <type>exponential</type>
          <parameters>
            <a>0.5</a>
          </parameters>
        </distribution>
      </transition>
      <!-- Arcs -->
      <arc id="arc1" source="T1" target="P1" type="output">
        <inscription>
          <text>1</text>
        </inscription>
      </arc>
    </page>
  </net>
</pnml>
```

Figure 4: A simple Petri net model in PNML format.

2.4 Petri Net Markup Language

In this tutorial, we formalize the extracted models using Petri net Markup Language (PNML), an XML-based standard designed to represent PNs in a tool-independent manner. The design of PNML is focused on three main principles (Billington et al. 2003).

- Flexibility: PNML represents any type of PN, including its features and extensions. By considering a PN as a labelled graph, PNML can store all additional information of a PN.
- Unambiguity: By defining fixed PN types, the legal labels for a particular PN are determined. Petri net Type Definition (PNTD) ensures that each PN can be determined from its PNML representation, making the description unambiguous.
- Compatibility: To enable exchange between different PN types, PNML uses a conventions document that defines the syntax and the meaning of labels or extensions. This enables the interpretation of new PNs by different tools without knowledge of their specific types.

Objects of PNs (places, transitions, arcs) are translated into XML syntax using PNML keywords, also called PNML elements. Figure 4 shows an example of a PN model in PNML format. In the example, various PNML elements and their corresponding attributes are visible.

2.5 Relevant Python Libraries

In this tutorial, we present our research workflow for data-driven PN model extraction and simulation. We use several Python libraries to implement and support this workflow for simulation, event log generation, model extraction, data processing, and visualization. Below is a summary of the key libraries we use:

- Graphviz (Ellson et al. 2002): Used to visualize the extracted PN structure and support validation.
- Pandas (McKinney 2011): Utilized for data handling and preprocessing, including trace construction and filtering from raw logs.

- SciPy (Virtanen et al. 2020): Used for statistical analysis, including fitting probability distributions to observed transition durations with tests such as Kolmogorov–Smirnov (KS).
- XML ElementTree and XML DOM (Ogbuji 2007): Employed to parse, edit, and generate PNML files, enabling the conversion between PNML-formalized models and simulation-ready formats.
- PySPN (Friederich et al. 2025): PySPN is an open-source Python library designed for modeling, simulation, and event log generation of SPNs. PySPN is part of our previous work (Khodadadi and Lazarova-Molnar 2024) and is included in this tutorial as the core simulation library for modeling and executing SPNs. The PySPN library supports multiple probability distributions (for instance, exponential, normal, and Weibull), immediate and timed transitions, inhibitor arcs, and guard functions. PySPN provides a flexible environment for representing complex discrete-event systems, such as manufacturing, logistics, and healthcare, making it suitable for DT simulation models. A key feature of PySPN is its ability to generate synthetic event logs, which are essential for PM, validation, and further refinement of DT models. In scenarios where real-world data is limited or unavailable, such as in early system design and sensitive environments, PySPN-generated event logs can serve as an alternative. PySPN enables the extraction of SPN models that can be continually updated and analyzed utilizing (near) real-time data. As a result, PySPN can be used to develop DTs that enable system monitoring, support what-if scenario analysis, and assist in decision-making in discrete event systems.

3 STOCHASTIC PETRI NETS AS UNDERLYING MODELS OF DIGITAL TWINS FOR SMART MANUFACTURING SYSTEMS

Process flows in SMSs can be modeled using PNs and simulated using Discrete-Event Simulation (DES). As we aim to extract DT models in a data-driven manner, we utilize PM for the automated extraction process of discrete-event simulation models. PN is a modeling formalism that, on the one hand, is capable of modeling discrete-event simulations (Long et al. 2016) and, on the other hand, is one of the few available modeling formalisms that can be directly extracted using PM. Besides the compatibility of PNs with PM and discrete-event systems, PNs are intuitive and widely used to model, simulate, and assess SMSs (Latorre-Biel et al. 2018; Friederich 2023). Further, PNs are capable of modeling flexible, automated manufacturing processes experiencing parallelism, asynchrony, bottlenecks, conflicts, and events (Latorre-Biel et al. 2018; Zhang et al. 2015). This makes PNs especially suitable as underlying DT models for SMSs.

Multiple studies utilize PNs as underlying DT models for SMSs. Friederich et al. (2022) proposed a data-driven DT framework that extracts SPNs using PM. Seok et al. (2022) proposed an approach to check consistency in DTs based on timed PNs. Dai et al. (2020) proposed a DT model based on PNs that implements a control structure for the connection with industrial pipelines to enhance real-time simulations. Pei et al. (2021) proposed an application paradigm to extend DTs for monitoring quality in solar cell production processes implemented through a place refinement time layered PN. Tsinarakis et al. (2020) proposed a DT for the development process of electric vehicles based on PN models. There exist multiple variants of PNs, such as SPNs, fuzzy PNs, and colored PNs. We utilize SPNs for our work, as they can capture concurrency and synchronization of discrete-event systems, as well as complexity and uncertainty. In the following, we explain how we extract SPNs as DT models with the help of PM.

To utilize SPNs as underlying models for DTs of SMSs, a structured approach is required, which we outline in Figure 5 and implement throughout this tutorial. The process begins with the collection of event logs, which are either recorded from the real-world system or synthetically generated by simulation tools such as PySPN, utilizing the PNML file of the real-world system. Event logs must be preprocessed to remove incomplete traces and ensure a consistent event structure. Once prepared, PM and data analysis techniques are applied to extract the structural model in the form of an SPN. For this, we identify transitions, places, and their relationships based on event sequences. Furthermore, we identify inhibitor arcs and guard conditions by analyzing outlier behavior and token dependencies. We then simulate the extracted model and check if it is validated against the real-world system by comparing key performance indicators (KPIs) such number of outputs, and structural validation. The validated model that closely replicates the behavior

of the original system is recognized as a DT, which can be used for supporting monitoring and analysis in different what-if scenarios.

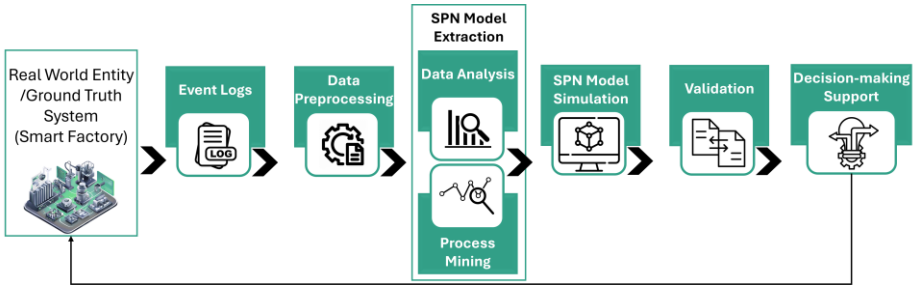

Figure 5: Methodology for utilizing SPNs as underlying models for Digital Twins of SMSs.

4 TUTORIAL SETUP

In this section, we outline the setup for the practical part of the tutorial. As illustrated in Figure 6, each case study begins with a ground-truth model described in PNML format. Subsequently, we run simulations using PySPN to generate event logs. The objective is to automatically extract the underlying SPN models from these event logs, capturing key features such as probability distributions of activity durations and inhibitor arcs. We then simulate the extracted models in PySPN to validate them and ensure their alignment with the ground-truth models. Once validated, the extracted models serve as the underlying models of DTs, which can be utilized for system analysis and decision-making support under various what-if scenarios.

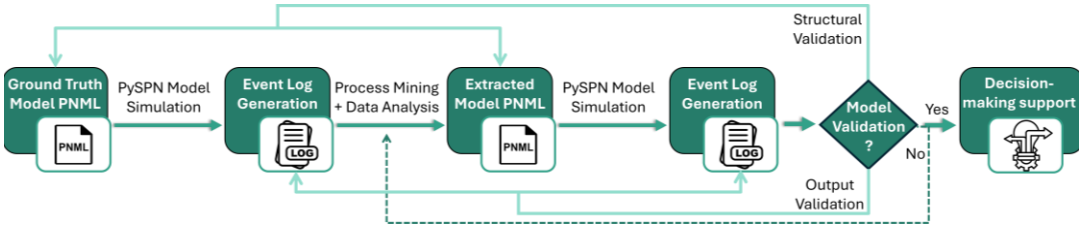

Figure 6: Steps of the tutorial setup.

4.1 Step 1: Ground-Truth Model Development and Event Log Generation

We begin with a ground-truth model that we use to generate event log data. The ground-truth model refers to a model that is accepted as an accurate representation of the real system for the purpose of comparison and evaluation. The ground-truth model is manually developed as an SPN, encoded using PNML. Through a custom parser, we import the PNML file directly into PySPN to simulate the ground-truth model and generate the corresponding event log. PySPN also provides the option to visualize the underlying SPN structure. The generated event log then serves as the input for the model extraction process.

4.2 Step 2: Model Extraction

The process of extracting an SPN model from event logs involves several key steps. In Algorithm 1, we outline the pseudocode for extracting a data-driven model from event logs, which we describe as follows:

1. **Trace Extraction and Data Pre-processing:** First, we sort the events by their case IDs (o) and timestamps (t). Second, we drop incomplete traces for each unique case ID that could mislead PM and result in invalid model structures. Next, we extract the underlying process flow of the system utilizing PM algorithms, such as Alpha Miner or Heuristic Miner (Premchaiswadi and Porouhan 2015), from which we extract the total available traces in the system (TR).

Algorithm 1: Extraction of a Stochastic Petri net and a corresponding PNML file from event log data.

Input: event log E
Output: SPN model PNML file

1. **Trace Extraction and Data Pre-processing**: Sort events by case ID and timestamps; remove incomplete traces, utilize the cleaned event log in PM, store ordered traces in the set TR.
2. **Petri-net Structure Construction:** For every trace $t \in TR$: for every pair of consecutive events mapped to transitions (T_i, T_{i+1}): Create a place P and add the arcs $T_i \rightarrow P_{i\_to\_i+1} \rightarrow T_{i+1}$; if a transition is an XOR-split: merge all of its outgoing places into a single place. Define input arcs A^I and output arcs A^O.
3. **Detection of Inhibitor Arcs and Transition's Types:**
   a. For every transition T_i and case j: compute $\Delta_i = \sum_{j=1}^{m} \mathrm{t_{j,i}} - \min(t_{j,i-1},\ t_{j\_enabled})$;
   b. If $\Delta_i = 0$: Add T_i to the immediate transitions' set T_I;
   Otherwise, remove outliers and set t_{med} ; compute the expected timestamp $t_{j_{expected}} = t_{j_{enabled}} + t_{med}$; If $\Delta_{i,j} > t_{j\_expected}$ and E_{i+1} is enabled: mark inhibitor arc from the place P(T_i_to_T_{i+1}) to T_i and remove the effect of inhibitor arc on time duration for E_i and return $\Delta'_{i,j}$; if $\Delta'_i = 0$, add T_i to T_I; else add T_i to T_T.
4. **Detection of Multiplicities**: In every trace $E_i \rightarrow E_{i+1}$: input arc multiplicity is the count occurrences of E_i before the first E_{i+1}; Output arc multiplicity is the count of occurrences E_{i+1} after the first E_i.
5. **Control-Flow Patterns Implementation:** Update SPN model by adding control-flow patterns, for instance:
   a. XOR-split: Identify any transition where all traces share the same prefix but branch at the next step; Integrate all of the outgoing places into one place and update the connecting arcs.
   b. AND-join: For every possible join: confirm that tokens arrive from all inputs at once and merge into the number of outputs; If not: merge the input places into a single place and update the arcs.
6. **Fitting of Durations Probability Distribution:** For every timed transition $T_i \in T_T$: Adjust the durations $\{\Delta'_i\}$ to candidate probability distribution families (such as normal, lognormal, or exponential) via MLE and retain the probability distribution with the highest KS p-value.
7. **Extracting Weights of Immediate Transitions**: For each immediate transition T_j involved in an XOR-split, count the occurrences of its event E_j and record this count as its weight f_j.
8. **PNML Generation:** Combine the extracted features into a complete SPN and export it in PNML format.

2. **Petri Net Structure Construction:** To construct the structure of PNs from event logs, we set each unique event identifier (E_i) as a transition (T_i). Next, from TR we identify the unique consecutive pairs of transitions (T_i, T_{i+1}), where for each such pair, we introduce a place P, labeled as $(T_{i\_to\_}T_{i+1})$. Finally, we define the sets of input arcs A^I, and output arcs A^O.
3. **Detection of Inhibitor Arcs and Transitions' Types:** Inhibitor arcs in the SPN model (and structures in the event log that lead to their extraction) can skew the timing distributions of timed transitions, as the delays they impose introduce outliers into the observed activity durations. To mitigate this, we employ a structured methodology to detect inhibitor arcs and eliminate their effects on activity durations. For this, we first record, for each activity's duration time, using:

$$\Delta_i = \sum_{j=1}^{m} \mathrm{t_{j,i}} - \min(t_{j,i-1},\ t_{j\_enabled}),$$

where $\mathrm{t_{j,I}}$ is the timestamp of the case ID_j for the event E_i, $t_{j,i-1}$ is the timestamp of the ID_j for E_{i-1} and $t_{j\_enabled}$ is the time that case ID_j is included in the activity (enabled corresponding transition T_i). If Δ_i for E_i is zero, we classify the corresponding transition T_i as an immediate transition and add it to T_I. Otherwise, we must confirm if the non-zero value for Δ_i is not related to the effect of an inhibitor arc condition. For this, we remove outliers of time duration values for Δ_i by discarding time durations outside the central 15–85 percentile interval (trimming the upper tail beyond the interquartile range) and computing the median delay t_{med}. For each entry with case ID_j={1, 2, …, m} for event E_i we predict the expected firing time (timestamp) as:

$$t_{j\_expected} = t_{j_{enabled}} + t_{med}.$$

If the $\Delta_{i,j}$ for a case ID for an event is more than $t_{j\_expected}$ and at the same time the subsequent event E_{i+1} is enabled, we introduce inhibitor arcs from the outgoing place to corresponding transition T_i for the event E_i. Finally, we recalculate the activity duration for E_i as $\Delta'_{i,j}$ where we remove the delay caused by the inhibitor arc and return Δ'_i. If Δ'_i is zero, then T_i is categorized as an immediate transition, and we add it to the set of immediate transitions T_I, otherwise T is assigned to the set of timed transitions T_T.

4. **Detection of Multiplicities:** We examine how many tokens (case ID) each transition (mapped to an event) consumes upon firing. Input multiplicity for an arc $E_i \rightarrow E_{i+1}$ is inferred by counting every trace that reaches E_i before the first occurrence of E_{i+1}, E_i appears. Conversely, output multiplicity for $E_i \rightarrow E_{i+1}$ is determined by counting, the occurrence of E_{i+1} after E_i's first occurrence in every such trace.
5. **Control-Flow Patterns Implementation:** To model system behavior accurately, we express control-flow patterns in the SPN and then update the resulting places, transitions, and arcs accordingly. For instance, for XOR-split to implement an XOR-split, we first identify transitions where traces share a common prefix but differ at the next event. In the corresponding SPN, we consolidate the multiple output places of such transitions (as assigned in Step 2) into a single place and draw arcs from this place to each possible subsequent transition. To model an AND-join, we detect points where traces with different prefixes meet on the same event. While the PN structure from Step 2 may already represent this, we verify its correctness by checking two conditions: (i) all required tokens from preceding events must arrive simultaneously before enabling the join transition, and (ii) firing the transition merges these tokens and generates new case IDs based on the output multiplicity. If these conditions are not met (indicating a false AND-join), we merge the transition's input places into one and update the arcs to reflect the corrected behavior.
6. **Fitting of Duration Probability Distribution:** For each timed transition T_i in the list of T_T, we fit Δ'_i by Maximum-Likelihood Estimation (MLE) using the SciPy library to a predefined family of distributions $\{e.g., norm, lognorm, expon\}$, estimating each candidate's parameters $\widehat{\hat{\theta}_T(f)}$. For this, we perform a one-sample Kolmogorov–Smirnov ($KS_t(f)$) test between the empirical cumulative distribution of Δ'_T and the theoretical Cumulative Distribution Function (CDF) $F(x;\ \hat{\theta}_T(f))$ for each f, and select the distribution f_t^* with the highest KS p-value as the time-feature.
7. **Extracting Weights of Immediate Transitions**: For every $T_i \in T_I$ with an XOR-split condition, we calculate the number of occurrences of the corresponding event E_i and set its value f_i for T_i.
8. **PNML Generation:** Once the SPN model structure and features are extracted, including duration probability distributions for timed distributions and weights for immediate transitions, the model is translated into code compatible with the PySPN simulation framework. To enable standardized exchange and reuse, the extracted PN model is translated to PNML. The resulting PNML file ensures interoperability and facilitates simulation within PySPN and other PN-compliant tools.

4.3 Step 3: Model Validation and Simulation

To ensure the extracted model accurately reflects the ground-truth model, we assess its validity both in terms of structural and operational validity. For structural validity, we compare the PNML representations of the ground-truth model and the extracted model. The structural validity is verified by examining that both models contain an equal number of places, transitions, and arcs. While this comparison could be performed by directly counting the elements in the PNML files, we utilize the visualization capabilities of PySPN to facilitate this process. Furthermore, we compare the predefined KPIs of the ground-truth model with the KPIs of the extracted simulation model. In output validation, we check whether the 95% confidence intervals of the KPIs overlap after 100 independent simulation runs.

5 STEP-BY-STEP SIMULATION MODEL EXTRACTION FOR TWO CASE STUDIES

To demonstrate the practical application of our data-driven model extraction and simulation approach, we present two case studies: a simple sequential manufacturing system and a two-server queueing system. The

first case study models a sequential process, where products are made in a fixed series of steps from start to finish. This scenario provides a clear and intuitive setting for introducing the methodology, making it easier to follow the mechanics of event log generation, model extraction, and simulation. In contrast, the second case study involves a two-server queueing system, where incoming tasks are randomly assigned to one of two parallel servers. This setup introduces non-determinism and concurrency, offering a more complex and realistic environment to evaluate the robustness and accuracy of the proposed model extraction framework. It highlights the method's ability to capture stochastic behavior and structural variability, which are common in real-world manufacturing systems.

5.1 First Case Study: Simple Sequential Manufacturing System

The first case study models a simple manufacturing system structured as a sequential process flow. The process begins with the placement of a new order, which occurs according to a gamma distribution with an average duration of around 15 minutes. Subsequently, the order is transported through a conveyor system to the production stage, which has a fixed duration of 2 minutes. The production process involves the assembly of the product and takes a variable time governed by a triangular distribution, typically ranging between 10 and a maximum of 10.30 minutes, with the most likely duration being 10.10 minutes. A guard function is associated with the conveyor transition, allowing it to fire after a time duration and only when an order is available and the production station is not occupied. Once production is completed, the manufacturing task is done, and the system is ready for delivery or subsequent processes.

5.1.1 Step 1: Ground-Truth Model Development and Event Log Generation

The case study ground-truth model is described as a PNML file and input to PySPN. Figure 7 and Table 1 show the SPN ground-truth model and its generated event log in CSV format, respectively.

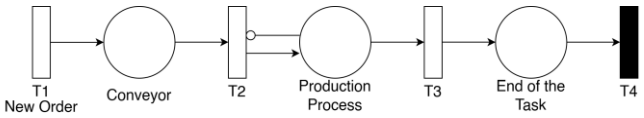

Figure 7: Stochastic Petri net of the simple sequential manufacturing system case study.

Table 1: The first case study (sequential manufacturing system) event log excerpt.

Timestamp	Order ID	Event
…	…	…
72.86	8	New Order
72.9	6	Conveyor End
74.23	4	Production Process End
74.23	4	End of the Task
…	…	…

5.1.2 Step 2: Model Extraction

Following the approach described in Section 4.2, we extract the underlying SPN model of the first case study in PNML format. The model extraction is based on the event log data generated by PySPN for simulating the ground-truth model, which is exported in CSV format.

5.1.3 Step 3: Model Validation and Simulation

Figure 8(a) presents the SPN generated from the PNML file using PySPN, which is redrawn for clarity. Comparing the structure of the extracted model to the ground-truth model, we observe that the extracted model preserves the same structure as the ground-truth model (same number of places, transitions, and arcs), demonstrating the structural equivalence of the model and validating the correctness of the extraction

process. In Figure 8(b), we present a quantitative comparison of the number of output products across 100 independent simulation replications of the first case study, showing strong correspondence between the extracted model and the ground-truth model.

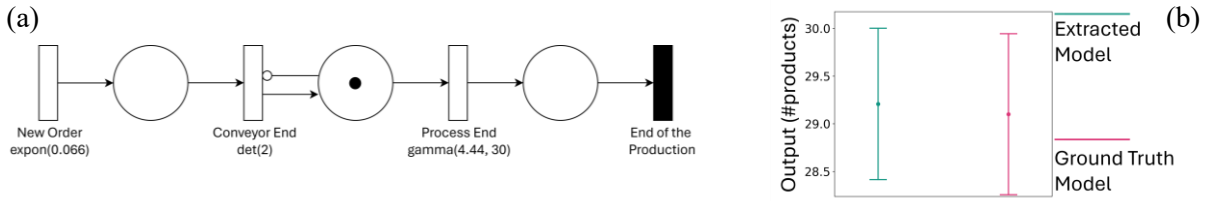

Figure 8: (a) Visual representation of the extracted Stochastic Petri net. (b) 95% confidence intervals for the KPI of the first case study.

5.2 Second Case Study: Two-Server Queueing System

The second case study models a two-server queueing system, designed to evaluate the proposed model extraction approach in a more complex and stochastic environment. In this system, new tasks (orders) arrive at variable intervals governed by an exponential distribution with a rate parameter of $\lambda = 1 / 7$. Each new task is first placed in the orders queue, from which it is routed to one of the two servers with equal probability (50%). Each server is able to process one entity at a time. The service time at each server follows a triangular distribution ranging from 20 to 22 minutes, with the most likely duration at 20.5 minutes. Comparably, the service time at Server 2 follows a triangular distribution ranging from 15 to 17 minutes, also with a mode of 15.5 minutes. A guard function is applied to the immediate transitions of both servers, allowing a task to proceed only if there is a task waiting in the queue and the server is available.

5.2.1 Step 1: Ground-Truth Model Development and Event Log Generation

We use PNML to describe the case study ground-truth model in PySPN. Figure 9 and Table 2 present the resulting SPN model and the event log generated by the PySPN simulation, respectively.

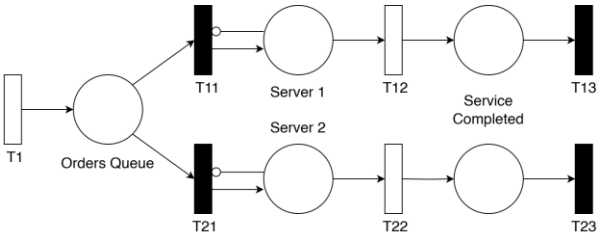

Figure 9: Stochastic Petri net of the two-server queueing system case study.

Table 2: Two-server queueing system event log excerpt.

Timestamp	Order ID	Event
…	…	…
75.12	10	New Order
89.8	7	Service Server 2 End
89.8	7	Order Completed 2 End
89.8	8	Direct to Line 2
…	…	…

5.2.2 Step 2: Model Extraction

We extract the model of the second case study by following the approach detailed in Algorithm 1. In the second case study, the extracted model includes both inhibitor arcs and transitions with fork conditions, where a single transition leads to multiple possible subsequent paths.

5.2.3 Step 3: Model Validation and Simulation

Figure 10(a) presents the SPN generated using PySPN from the PNML file, redrawn for clarity. The result of the simulation model validation shows that both the extracted model and the ground-truth model have the same structure in terms of the number of places, transitions, and arcs. Therefore, the extracted model is deemed structurally valid. Furthermore, in Figure 10(b), we present comparisons of the output counts across 100 independent simulation replications. The results show an overlap between the extracted and the ground-truth models.

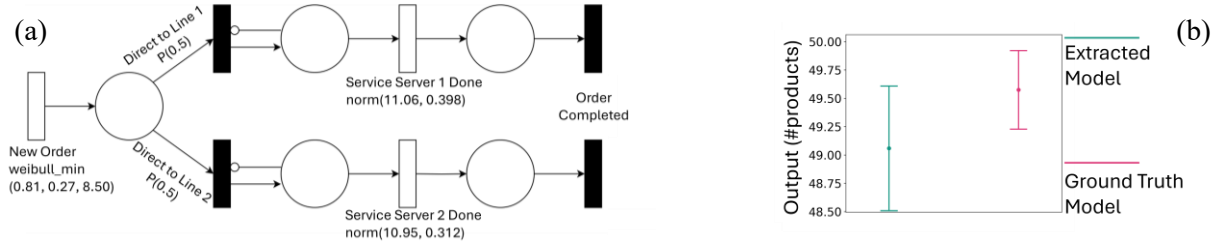

Figure 10: (a) Visual representation of the extracted PNML. (b) 95% Confidence intervals for the KPI of the second case Study.

6 SUMMARY AND OUTLOOK

In this tutorial, we present a workflow for the data-driven extraction and simulation of Petri net models, designed to support the development of Digital Twins for Smart Manufacturing Systems. We use Stochastic Petri nets as a formalism due to their expressive capacity for modeling discrete-event system behaviors and their compatibility with Process Mining techniques. The tutorial outlines a step-by-step methodology encompassing event log preprocessing, structural model extraction, probability distribution fitting for activity durations, multiplicities, and inhibitor arcs detection, and model validation. To simulate and validate the extracted Stochastic Petri net models, we utilize PySPN, an open-source Python library designed for stochastic Petri net modeling and simulation. We illustrate the workflow through two case studies involving simplified and fragmentary representations of manufacturing systems, chosen to demonstrate the feasibility and core principles of the approach. These case studies, while not exhaustive, show how Stochastic Petri nets can be derived from data to support key performance indicator analysis and inform decision-making processes. With this tutorial, we aim to offer a clear and practical foundation for researchers aiming to integrate data-driven Petri net modeling into Digital Twin development. Future work may explore extending this methodology to more complex systems, real-time data integration, and broader classes of Petri nets.

ACKNOWLEDGMENTS

The authors extend their thanks for the funding received from the ONE4ALL project (Grant Agreement No. 101091877) and the DMaaST project (Grant Agreement No. 101138648), both funded by the European Commission under the Horizon Europe Programme.

REFERENCES

Augusto, A., R. Conforti, M. Dumas, M. L. Rosa, F. M. Maggi, A. Marrella et al. 2019. “Automated discovery of process models from event logs: Review and benchmark.” *IEEE Transactions on Knowledge and Data Engineering* 31(4):686–705.

Billington, J., S. Christensen, K. Van Hee, E. Kindler, O. Kummer, L. Petrucci et al. 2003. "The Petri Net Markup Language: Concepts, technology, and tools." In *Applications and Theory of Petri Nets 2003*, 483–505. Berlin: Springer.

Birk, A., Y. Wilhelm, S. Dreher, C. Flack, P. Reimann, and C. Gröger. 2021. "A real-world application of process mining for data-driven analysis of multi-level interlinked manufacturing processes." *Procedia CIRP* 104:417–422.

Dai, Y., Y. Shi, Z. Zhang, R. Tao, and F. Fang. 2020. "Digital twins driving model based on Petri net in industrial pipeline." In *2020 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, 283–291. IEEE.

Desel, J. and W. Reisig. 2015. "The concepts of Petri nets." *Software & Systems Modeling* 14:669–683.

Ellson, J., E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull. 2002. "Graphviz—open source graph drawing tools." In *Graph Drawing*, 483–484. Berlin: Springer.

Friederich, J. 2023. *Data-driven assessment of reliability for cyber-physical production systems*. Ph.D. thesis, University of Southern Denmark.

Friederich, J., A. Khodadadi, and S. Lazarova-Molnar. 2025. "PySPN: A Python library for stochastic Petri net modeling, simulation, and event log generation." *SIMULATION: Transactions of the Society for Modeling and Simulation International*. https://doi.org/10.1177/00375497251343625.

Friederich, J. and S. Lazarova-Molnar. 2021. "Process mining for reliability modeling of manufacturing systems with limited data availability." In *2021 8th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, 1–7. Gandia, Spain: IEEE.

Friederich, J., D. P. Francis, S. Lazarova-Molnar, and N. Mohamed. 2022. "A framework for data-driven digital twins of smart manufacturing systems." *Computers in Industry* 136:103586.

Fuller, A., Z. Fan, C. Day, and C. Barlow. 2020. "Digital twin: Enabling technologies, challenges and open research." *IEEE Access* 8:108952–108971.

Greco, A., M. Caterino, M. Fera, and S. Gerbino. 2020. "Digital twin for monitoring ergonomics during manufacturing production." *Applied Sciences* 10(21):7758.

Grieves, M. and J. Vickers. 2017. "Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems." In *Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches*, edited by F.-J. Kahlen, S. Flumerfelt, and A. Alves, 85–113. Cham: Springer International Publishing.

He, R., G. Chen, C. Dong, S. Sun, and X. Shen. 2019. "Data-driven digital twin technology for optimized control in process systems." *ISA Transactions* 95:221–234.

International Organization for Standardization. 2021. *ISO 23247-1:2021 – Automation systems and integration – Digital twin framework for manufacturing – Part 1: Overview and general principles*. Geneva, Switzerland: ISO.

Jungmann, M. and S. Lazarova-Molnar. 2024. "Towards fusing data and expert knowledge for better-informed digital twins: An initial framework." In *15th International Conference on Ambient Systems, Networks and Technologies (ANT 2024) / 7th International Conference on Emerging Data and Industry 4.0 (EDI40 2024)*, 639–646. Hasselt, Belgium.

Khodadadi, A. and S. Lazarova-Molnar. 2024. "Essential data requirements for industrial energy efficiency with digital twins: A case study analysis." In *7th International Conference on Emerging Data and Industry 4.0 (EDI40 2024)*, 631–638. Hasselt, Belgium.

Khodadadi, A. and S. Lazarova-Molnar. 2024. "Data-driven extraction of simulation models for energy-oriented digital twins of manufacturing systems: An illustrative case study." In *2024 Winter Simulation Conference (WSC)*, 1669–1680. Orlando, Florida, USA. https://doi.org/10.1109/WSC63780.2024.10838974.

Kritzinger, W., M. Karner, G. Traar, J. Henjes, and W. Sihn. 2018. "Digital twin in manufacturing: A categorical literature review and classification." *IFAC-PapersOnLine* 51(11):1016–1022.

Kunath, M. and H. Winkler. 2018. "Integrating the digital twin of the manufacturing system into a decision support system for improving the order management process." *Procedia CIRP* 72:225–231.

Latorre-Biel, J.-I., J. Faulín, A. A. Juan, and E. Jiménez-Macías. 2018. "Petri net model of a smart factory in the frame of Industry 4.0." *IFAC-PapersOnLine* 51(2):266–271.

Lazarova-Molnar, S. 2005. *The Proxel-Based Method: Formalisation, Analysis and Applications*. Ph.D. thesis, Otto-von-Guericke University Magdeburg.

Lazarova-Molnar, S. 2024. "A vision for advancing digital twins intelligence: Key insights and lessons from decades of research and experience with simulation." In *Proceedings of the 14th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2024)*, 5–10. Dijon, France: SCITEPRESS.

Leng, J., D. Wang, W. Shen, X. Li, Q. Liu, and X. Chen. 2021. "Digital twins-based smart manufacturing system design in Industry 4.0: A review." *Journal of Manufacturing Systems* 60:119–137.

Liu, X., D. Jiang, B. Tao, F. Xiang, G. Jiang, Y. Sun, J. Kong, and G. Li. 2023. "A systematic review of digital twin about physical entities, virtual models, twin data, and applications." *Advanced Engineering Informatics* 55:101876.

Long, F., P. Zeiler, and B. Bertsche. 2016. "Modelling the production systems in industry 4.0 and their availability with high-level Petri nets." *IFAC-PapersOnLine* 49(12):145–150.

Lorenz, R., W. Senoner, W. Sihn, and T. Netland. 2021. "Using process mining to improve productivity in make-to-stock manufacturing." *International Journal of Production Research* 59(16):4869–4880.

Mahmoud, M. A. and J. Grace. 2019. "A generic evaluation framework of smart manufacturing systems." *Procedia Computer Science* 161:1292–1299.

McKinney, W. 2011. "Pandas: A foundational Python library for data analysis and statistics." *Python for High Performance and Scientific Computing* 14(9):1–9.

Ogbuji, U. 2007. "Python and XML." In *XML Prague: A Conference on XML*, 53–54. Charles University, Prague.

Pei, F.-Q., Y.-F. Tong, M.-H. Yuan, K. Ding, and X.-H. Chen. 2021. "The digital twin of the quality monitoring and control in the series solar cell production line." *Journal of Manufacturing Systems* 59:127–137.

Peterson, J. L. 1981. *Petri net theory and the modeling of systems*. Englewood Cliffs, NJ: Prentice-Hall.

Piascik, R., J. Vickers, D. Lowry, S. Scotti, J. Stewart, and A. Calomino. 2010. *Technology Area 12: Materials, Structures, Mechanical Systems, and Manufacturing Road Map*. Washington, DC: NASA Office of Chief Technologist.

Premchaiswadi, W. and P. Porouhan. 2015. "Process simulation and pattern discovery through alpha and heuristic algorithms." In *2015 International Conference on Industrial Engineering and Operations Management (IEOM)*, 60–66. IEEE.

Seok, M. G., W. J. Tan, W. Cai, and D. Park. 2022. "Digital-twin consistency checking based on observed timed events with unobservable transitions in smart manufacturing." *IEEE Transactions on Industrial Informatics* 19(4):6208–6219.

Soori, M., B. Arezoo, and R. Dastres. 2023. "Digital twin for smart manufacturing: A review." *Sustainable Manufacturing and Service Economics* 2:100017.

Tao, F., Q. Qi, L. Wang, and A. Y. C. Nee. 2019. "Digital twins and cyber–physical systems toward smart manufacturing and Industry 4.0: Correlation and comparison." *Engineering* 5(4):653–661.

Tsinarakis, G. J., P. S. Spanoudakis, G. Arabatzis, N. C. Tsourveloudis, and L. Doitsidis. 2020. "Implementation of a Petri-net based digital twin for the development procedure of an electric vehicle." In *2020 28th Mediterranean Conference on Control and Automation (MED)*, 862–867. IEEE.

Van der Aalst, W., T. Weijters, and L. Maruster. 2004. "Workflow Mining: Discovering Process Models from Event Logs". *IEEE transactions on knowledge and data engineering* 16 (9):1128-1142.

Van Der Aalst, W. M. P. 2011. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Berlin: Springer.

VanDerHorn, E. and S. Mahadevan. 2021. "Digital twin: Generalization, characterization and implementation." *Decision Support Systems* 145:113524.

Virtanen, P., R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, and J. Bright. 2020. "SciPy 1.0: Fundamental algorithms for scientific computing in Python." *Nature Methods* 17(3):261–272.

Zhang, Y., W. Wang, N. Wu, and C. Qian. 2015. "IoT-enabled real-time production performance analysis and exception diagnosis model." *IEEE Transactions on Automation Science and Engineering* 13(4):1318–1332.

AUTHOR BIOGRAPHIES

ATIEH KHODADADI is a Ph.D. student at the Institute of Applied Informatics and Formal Description Methods, Karlsruhe Institute of Technology, Germany. Her research concentrates on using Digital Twins to improve energy efficiency in manufacturing systems, specifically through modeling, simulation, and data analysis. Her email address is atieh.khodadadi@kit.edu.

ASHKAN ZARE is a PhD student at the Mærsk Mc-Kinney Møller Institute, University of Southern Denmark. His research interests concern Modeling and Simulation, Data Analytics, and Machine Learning. His PhD project focuses on continuous validation of Digital Twins in manufacturing systems. His email address is zare@mmmi.sdu.dk.

MICHELLE JUNGMANN is a Ph.D. student at the Institute of Applied Informatics and Formal Description Methods at the Karlsruhe Institute of Technology. Her research is focused on the extraction of data-knowledge fused models from both expert knowledge and data in the context of Digital Twins. Her email address is michelle.jungmann@kit.edu.

MANUEL GÖTZ is a Ph.D. candidate at the Institute of Applied Informatics and Formal Description Methods at Karlsruhe Institute of Technology in Germany. His research focuses on extracting Digital Twins of labor-intensive manufacturing systems with a particular focus on integrating human factors of operators. His email address is manuel.goetz@kit.edu.

SANJA LAZAROVA-MOLNAR is a Professor at both the Karlsruhe Institute of Technology and the University of Southern Denmark. Her research focuses on data-driven simulation, Digital Twins, and cyber-physical systems modeling, with an emphasis on reliability and energy efficiency. She develops advanced methodologies to optimize complex systems and leads several European and national projects in these areas. Prof. Lazarova-Molnar holds leadership roles in IEEE and The Society for Modeling & Simulation International (SCS), where she currently serves as SCS Representative to the Winter Simulation Conference (WSC) Board of Directors. She was Proceedings Editor for WSC in 2019 and 2020 and serves as Associate Editor for *SIMULATION: Transactions of The Society for Modeling and Simulation International*. Her email address is lazarova-molnar@kit.edu.