



MDPNML: A Multidimensional Petri Net Markup Language Enabling Construction and Simulation of Comprehensive Digital Twin Models

Atieh Khodadadi¹^a and Sanja Lazarova-Molnar^{1,2}^b

¹*Institute AIFB, Karlsruhe Institute of Technology, Karlsruhe, Germany*

²*The Maersk Mc-Kinney Moller Institute, University of Southern Denmark, Odense, Denmark*

Keywords: Multidimensional Stochastic Petri Nets, Petri Net Markup Language (PNML), Multidimensional Petri Net Markup Language (MDPNML).

Abstract: A Digital Twin (DT) is a data-driven virtual representation of a physical system that updates in near-real-time and incorporates models of system physics and behaviors. Multidimensional Stochastic Petri Nets (MDSPNs), as an extension of traditional Stochastic Petri Nets (SPNs), provide an intuitive formalism for modeling and analyzing complex systems across multiple dimensions, enabling the development of comprehensive DTs. In MDSPNs, system objectives can be associated with different relevant dimensions, including time, energy, and waste. The Petri Net Markup Language (PNML), a standard XML-based interchange format, is widely used for sharing and executing Petri net models across tools. PNML, however, lacks multidimensional semantics. A PNML-compatible format for MDSPNs would enable portable model exchange across tools and allow conversion to time-oriented SPNs for generic PNML tools, supporting end-to-end traditional and comprehensive DT workflows. Such an extended PNML format also enhances model reproducibility and reduces the effort required to integrate dimension-specific behavior. In this paper, we introduce the Multidimensional Petri Net Markup Language (MDPNML) and identify necessary adaptations to the PNML format to represent MDSPNs. MDPNML supports multidimensional attributes, different dimensions in transitions, and simulation parameters for MDSPNs. Through an illustrative case study, we demonstrate MDPNML generation and execution for multidimensional simulation.

1 INTRODUCTION

Digital Twins (DTs) are an emerging technology that supports digital transformation and decision-making across industries (VanDerHorn & Mahadevan, 2021). As illustrated in Figure 1, a DT consists of three key elements: (1) a physical system, (2) its virtual counterpart, and (3) bidirectional data connections that facilitate continuous information exchange between the two. The virtual counterpart is often an executable simulation model calibrated with operational data to mirror the system's state and enable prediction and “what-if” analysis (Boschert & Rosen, 2016).

Discrete-event modeling is a widely used methodology for complex, event-driven systems,

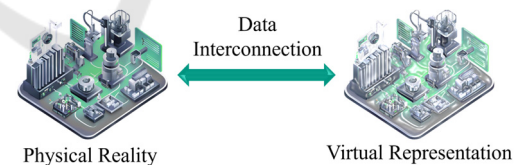




Figure 1: Digital Twin components (VanDerHorn & Mahadevan, 2021).

such as manufacturing systems, because it captures state changes at discrete points in time and supports performance analysis and control design (Cassandras & Lafortune, 2008). In event-driven systems, process mining plays a central role in DT model development by extracting process models from event logs generated from the real-world system (dos Santos, Micosky, de Freitas Rocha Loures, & Alves Portela

^a <https://orcid.org/0009-0001-5084-4856>

^b <https://orcid.org/0000-0002-6052-0863>

Santos, 2024; van der Aalst, 2018). Discovered models, serving as underlying DT models, can be represented in various modeling languages and formalisms, including Petri nets, Business Process Model Notation (BPMN), or process trees (van der Aalst, 2022).

Petri nets, first introduced by Carl A. Petri in 1962 (Petri, 1962), provide a powerful mathematical and graphical modeling formalism for the design and analysis of discrete-event systems. One of Petri nets' key strengths is their ability to use a single model for both behavioral and performance analysis, and support the systematic design of discrete-event controllers and simulators (Holloway, Krogh, & Giua, 1997). For instance, the same Petri net can be analyzed for safety, liveness, and nonblocking, and, once extended with timing or probabilities, simulated for performance evaluation.

Petri nets encompass several extensions, such as Colored Petri Nets (CPNs) (Gehlot & Nigro, 2010) and Stochastic Petri Nets (SPNs) (Bause & Kritzinger, 2002), which differ mainly in how they represent tokens, handle timing, and manage system complexity. A more recent development, Multidimensional Stochastic Petri Nets (MDSPNs) (Khodadadi & Lazarova-Molnar, 2025a), provides a formal basis for Comprehensive Digital Twins (CDTs) (Khodadadi & Lazarova-Molnar, 2025b) that can represent multiple impact dimensions within a single comprehensive model. CDTs simultaneously track, analyze, and predict system behavior across several dimensions such as time, energy, and emissions, supporting multi-objective decision-making. In CDTs, MDSPNs capture and update per-transition impacts on each dimension (such as time, energy, waste), making them well-suited for CDTs modeling and simulation.

To ensure that Petri net models are portable across tools and usable over time, the community relies on international standards and a common exchange format. Such a standardized format also facilitates the integration of DT workflows, enabling model sharing, calibration, and execution across various DT tools and platforms (Lagartinho-Oliveira, Moutinho, & Gomes, 2024). The International Standard for Petri nets (ISO/IEC 15909) consists of three parts (Hillah, Kordon, Petrucci, & Treves, 2010). Part 1 defines the fundamental concepts and notations for Place/Transition, Symmetric, and high-level nets. Part 2 (ISO/IEC 15909-2) specifies the XML-based Petri Net Markup Language (PNML) for model exchange. Part 3 (ISO/IEC 15909-3) targets extensions (such as modularity, time, probabilities) and small semantic differences (such as inhibitor arcs,

bounded places), requiring flexibility in the standard. PNML is an XML-based format designed to exchange Petri net models. PNML accommodates current and future Petri-net variants and serves as a flexible foundation for standardized model interchange (Billington et al., 2003).

However, PNML in its current form does not capture multidimensional attributes required by MDSPNs. We, therefore, propose MDPNML, an extension of PNML (Part 3) that introduces constructs for multidimensional metrics and their associated behaviors. The Multidimensional Petri Net Markup Language (MDPNML) enables portable exchange of MDSPN models and their conversion to time-oriented SPNs for use with generic PNML tools, all within a consistent, standardized format. MDPNML supports end-to-end CDT workflow (from process discovery to simulation and monitoring) and enables MDSPN simulation with dimensions aligned to system objectives, facilitating real-time analysis and CDT development. Furthermore, through an illustrative heat-sealing case study, we validate MDPNML via end-to-end parsing and simulation of the resulting multidimensional model.

We structure the paper as follows: Section 2 reviews the foundations of this work, including SPNs, MDSPNs, PNML, and CDTs. Section 3 introduces MDPNML, outlining the requirements and PNML adaptations for multidimensional semantics. Section 4 presents a case study and its MDPNML encoding and execution. Finally, Section 5 concludes with limitations and directions for future work.

2 BACKGROUND AND RELATED WORK

In this section, we outline the core concepts underlying this work. We first review SPNs, then introduce MDSPNs, and conclude with an overview of PNML and related work.

2.1 Stochastic Petri Nets

Petri nets capture key system properties such as synchronization, concurrency, asynchrony, resource sharing, and conflict, which are essential in industrial automation, communication, and computer-based systems (Zurawski & Zhou, 1994). These characteristics make Petri nets a powerful framework for managing the complexity of modern industrial systems, where accurate modeling is essential for efficient design, operation, and decision-making.

SPNs extend the basic Petri net formalism by assigning stochastic timing features to transitions, capturing temporal uncertainty, and enabling quantitative performance evaluation in systems with probabilistic behaviors. Following Lazarova-Molnar (Lazarova-Molnar, 2005), we define an SPN as $SPN = (P, T, A, G, m_0)$, where:

- $P = \{P_1, P_2, \dots, P_m\}$ is a finite set of places (drawn as circles).
- $T = \{T_1, T_2, \dots, T_n\}$ is a finite set of transitions (drawn as bars), each with either a firing-time distribution or, where applicable, a positive weight.
- $A = A^I \cup A^O \cup A^H$ is a set of arcs, with $A^I = \{a_1^I, a_2^I, \dots, a_j^I\}$ (input arcs that connect places to transitions), $A^O = \{a_1^O, a_2^O, \dots, a_k^O\}$ (output arcs that connect transitions to places), and $A^H = \{a_1^H, a_2^H, \dots, a_l^H\}$ (inhibitor arcs that disable a transition when a connected place holds at least a given token count). Each arc carries a (natural-number) multiplicity, and $A^O \subseteq (T \times P \times N)$ and $A^I, A^H \subseteq (P \times T \times N)$.
- $G = \{g_1, g_2, \dots, g_r\}$ is a set of guard functions attached to transitions; a guard must evaluate to true for the transition to be enabled.
- And m_0 is the initial marking of the Petri net (token distribution over places).

A transition in a Petri net is enabled when (i) every input place holds at least the number of tokens required by its incoming arc multiplicity, (ii) its guard condition evaluates to true, and (iii) no connected inhibitor arc blocks it. Arc multiplicity specifies how many tokens the transition consumes or produces, thereby supporting batch operations and modeling resource usage. When an enabled transition fires, it removes tokens from input places and adds tokens to output places according to the multiplicities and updates the Petri net's marking from M to M' . Each transition is represented as $T_i = (F, mp)$. The memory-policy indicator $mp \in \{enabling, age, immediate\}$ specifies the type of memory policy: for a timed transition, mp is either enabling or age; for an immediate transition, mp is immediate. The component F is the cumulative distribution function associated with the transition when it is timed. Immediate transitions do not have a distribution function; instead, they carry a constant value used to compute the firing probability when more than one immediate transition is enabled simultaneously.

2.2 Multidimensional Stochastic Petri Nets (MDSPNs)

While classical SPNs model stochastic discrete-event systems effectively, they mostly capture only the temporal dimensions and struggle to represent other operational dimensions, such as energy or waste, in a single comprehensive model. CPNs can present multiple dimensions (such as time, energy, waste) by adding tokens with colors and attributes (Zimmermann, 2008). For example, Kaiyandra et al. (2024) model a green supply chain including forward and reverse/recycling flows using a CPN, and Wang et al. (2021) use a real-time CPN within an Industrial Internet of Things (IIoT)/Cyber-Physical System (CPS) in a smart-factory setting to analyze and predict key performance indicators along two dimensions of time and energy consumption in a CNC manufacturing case study. In practice, however, these multidimensional aspects are hidden in the token colors and guards, and a static view of the net (without tokens or animation) reveals only the control flow and basic resource structure similar to a Petri net, not the underlying energy or waste flows. Indeed, most efforts to automatically discover CPN models from data such as logs have focused on capturing control flow and basic time/resource behavior, and any energy or cost dimensions have been incorporated manually in the token inscriptions (Rozinat, Mans, Song, & Van der Aalst, 2008). In contrast, MDSPN can be automatically discovered from multidimensional event logs (Khodadadi & Lazarova-Molnar, 2024a) utilizing Multi-flow Process Mining (MFPM) (Khodadadi & Lazarova-Molnar, 2024c). In an MDSPN, every transition presents its effects on all dimensions, so the multidimensional impact of each activity is built into the model's structure intuitively. Thus, a domain expert can immediately see where energy is consumed or waste is produced by looking at the MDSPN (even before simulation), making it more suitable for analyzing the system's multi-objective performance.

Case Study: Consider a heat-sealing packaging line with dimensions of time, energy consumption, and CO_2 emissions generation. Arrival of a batch occurs via T1, which is exponentially distributed with a rate $\lambda = 1.5$. The line runs a sealing cycle for a defined duration (T3) following a triangular distribution ranging from 3 to 4 minutes, with the most likely duration at 3.5. During sealing, the line draws electrical power (E3), with an energy consumption rate of 0.08 kWh per minute and generates a constant 0.004 grams of CO_2 emissions

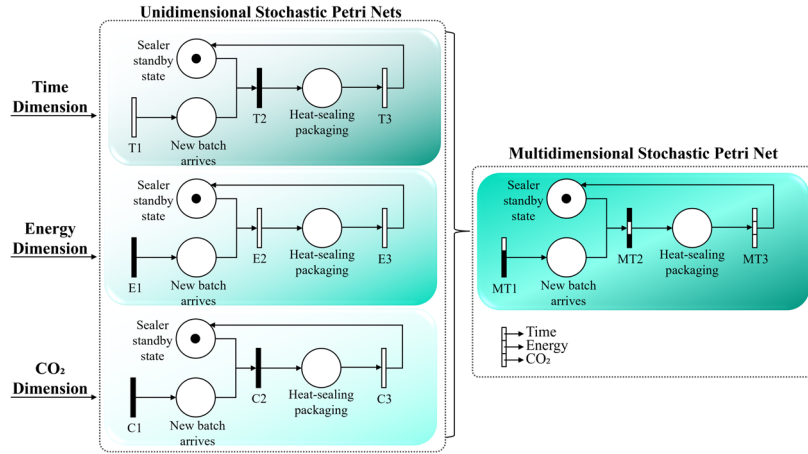


Figure 2: Unidimensional and multidimensional SPN models of a heat-sealing packaging line.

(C3) per minute. Between batches, the sealer remains at temperature: this standby step is an immediate transition in the time-oriented model (T2), contributes to energy use (E2) with an energy consumption rate of 0.01 kWh per minute, and is non-contributing in the CO_2 model (C2). In Figure 2 (left), the unidimensional models reflect this: arrival affects only time (T1), standby affects only energy (E2), and the sealing operation contributes to all three dimensions (T3, E3, C3). Unidimensional SPN models are building blocks for MDSPNs. Each unidimensional model is one SPN model focused on a single dimension and analyzes behavior in its own dimension, supports its dimension-oriented bottleneck identification, and enables dimension-specific optimization.

We then compose the unidimensional models into a single MDSPN that captures behavior across all dimensions (Figure 2, right). In the MDSPN, a transition may impact none, some, or all dimensions. Firing still creates/destroys tokens as usual, and additionally updates dimension-specific clocks, for example, time, energy, and waste. Unlike time, which advances monotonically, other dimensions may increase or decrease (such as net waste generation, which can be reduced through material reuse). We currently assume identical topologies across dimensions; transitions may be immediate in the time dimension or, in our terms, non-contributing in specific dimensions. To support multidimensionality, we extend the SPN formalism as follows:

- **Transition Definition:** We refine the transition tuple to $T_i = (dim, type, F)$ where dim identifies the affected dimension and $type \in \{contributing, noncontributing\}$.

- **Graphical Element:** We adopt a segmented transition symbol to encode per-dimension effects. As shown in Figure 2 (right) for MT1–MT4, the top segment denotes the time dimension, the middle segment the energy dimension, and the bottom segment the CO_2 dimension.
- **Distinct Clocks:** We maintain separate simulation clocks for each dimension. The temporal clock advances time (conventional simulation clock), while the other clocks track updates in their respective dimensions. Upon a transition firing, the relevant clocks are updated, ensuring a dimension-specific representation of system behavior.

In the integrated model, tokens flow as in the temporal net, while transitions that affect other dimensions adjust the corresponding clocks. MDSPNs capture both process progression and cross-dimensional effects (such as energy use and CO_2 generation), which is essential for accurately representing real-world systems' dynamics where actions can consume resources or generate waste.

2.3 Petri Net Markup Language (PNML)

Researchers in the Petri net community have long emphasized the importance of enabling model transfer across tools developed in different parts of the world (Billington et al., 2003). This interoperability among Petri net tools supports collaboration among users worldwide and allows them to access advanced functionalities developed in other tools, including analysis, simulation, and implementation. PNML is an XML-based,

standardized interchange format for Petri net models that facilitates tool-to-tool exchange.

2.3.1 XML

XML is a meta-language for defining specialized markup languages. An XML document consists of elements (tags) enclosed in angle brackets (< >). Tags appear either as paired start/end tags (such as, <item>...</item>) or as empty-element tags (such as, <item/>) when no content is enclosed; both forms may include attributes of the form name="value". Defining a new markup language requires specifying the permitted tags, their attributes, allowable nesting, and ordering. Conformance can be checked automatically by validating documents against a machine-readable specification, such as an XML Schema (Stehno, 2002).

2.3.2 PNML Concepts and Structure

PNML classifies its concepts as core or tool-specific. The core concepts capture the common graphical and structural elements of Petri nets (Bonnefoi, Choppy, & Kordon, 2009). These elements are utilized by all classes of Petri net models and tools, such as places, transitions, arcs, and labels. In contrast, tool-specific concepts record information for particular tools or Petri-net subclasses, such as editor settings and analysis annotations. These tool-specific details are intended for those particular tools and nets only and are not meant for cross-tool reuse (Murata, 1989). In this paper, we focus on the PNML core concepts.

Each Petri net type is composed of a predefined set of components, which defines a new type that mainly involves selecting the elements and fixing

Table 1: Mapping PNML core-model concepts to XML elements and attributes (Billington et al., 2003).

Class	XML element	XML Attributes
PetriNetFile	<pnml>	
PetriNet	<net>	id: ID type: anyURI
Place	<place>	id: ID
Transition	<transition>	id: ID
Arc	<arc>	id: ID source: IDRef (Node) target: IDRef (Node)
Page	<page>	id: ID
RefPlace	<referencePlace>	id: ID ref: IDRef (Place/RefPlace)
RefTrans	<referenceTransition>	id: ID ref: IDRef (Transition/RefTrans)
ToolInfo	<toolspecific>	tool: string version: string
Graphics	<graphics>	

their order. The common graphical and structural elements in PNML (such as places, transitions, arcs, and labels) are identified by a special attribute and represented by a named tag, as we present in Table 1 (URI stands for Uniform Resource Identifier). These tags may include child tags, such as a name or initial marking (m_0). The base syntax also provides elements for graphical information (absolute or parent-relative positions) and for separating nodes across pages. More specific properties or additional net elements require newly defined tags. Although tags could be chosen arbitrarily, that would hinder interoperability across tools. Instead, the syntax and semantics of every tag should be specified in a shared document to ensure consistency across related net types. Unfortunately, the current PNML implementation lacks widely adopted schemas, so examples of different Petri-net types are often presented without them, which limits the language cross-tool consistency (Stehno, 2002).

In Table 2, we present the PNML syntax for the time-oriented SPN used in the heat-sealing packaging case study in Section 2.2. Transitions carry a timing flag via the type of attribute: type="T" denotes a timed transition, or type="I" which denotes an immediate transition (immediate transitions are

Table 2: PNML syntax for the time-oriented Petri net in the heat-sealing packaging case study (excerpt).

```

<pnml>
<net id="heatSeal_time">
  <place id="p1">
    <name>
      <text>Sealer standby state</text>
    </name>
    <initialMarking>
      <text>1</text>
    </initialMarking>
  </place>
  ...
  <transition id="T1" type="T">
    <name><text>New arrival</text></name>
    <distribution>
      <type>exponential</type>
      <parameters><a>1.5</a></parameters>
    </distribution>
  </transition>
  ...
  <arc id="a1" source="T1" target="p2">
    <type="output">
      <inscription><text>1</text></inscription>
    </arc>
  ...
</net>
</pnml>

```

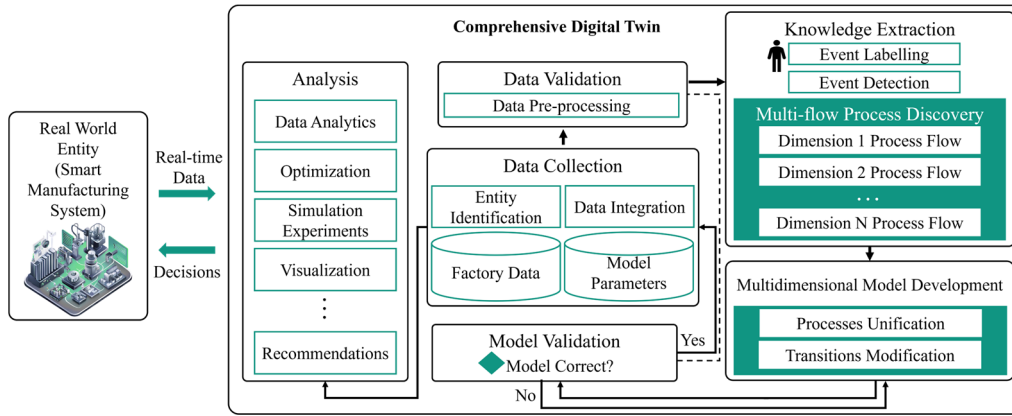


Figure 3: Comprehensive Digital Twin framework for smart manufacturing systems (Khodadadi & Lazarova-Molnar, 2025b).

part of the model encoding but are not shown in the excerpt). Timed transitions include a `<distribution>` block with parameters. In the excerpt, transition T1 uses an exponential distribution with parameter $a = 1.5$ (rate λ). We provide PNML code, and the corresponding parser for running in the PySPN (Friederich, Khodadadi, & Lazarova-Molnar, 2025), in our GitHub repository (Khodadadi, Lazarova-Molnar, 2025c).

2.4 Comprehensive Digital Twin

In Figure 3, we illustrate the Comprehensive Digital Twin (CDT) framework for the smart manufacturing system, which continuously collects data from IoT devices and controllers.

The workflow begins by collecting data, followed by identifying key entities (such as production systems and control technologies, and their corresponding dimensions) and storing their data in structured databases. We then validate the data through cleaning, preprocessing, and integration into a multidimensional event log. The validated data include information about critical factory events and their impact on the relevant dimensions. Unsupervised learning techniques, such as clustering, improve event detection and labeling (Vaarandi, 2003). Experts review the clusters to verify accuracy, add labels, and make corrections. Using the reviewed data, machine learning algorithms then continuously and automatically detect events and compile comprehensive multidimensional event logs. The labeled multidimensional logs support the discovery of processes by employing the Multi-Flow Process Mining (MFPN) algorithm, which forms the basis for building unidimensional process flows. In the next step, we integrate all targeted dimensions' unidimensional SPN into a single MDSPN.

Next, we simulate the MDSPN model, where time is the fundamental, mandatory dimension. We then validate the CDT model to ensure it accurately mirrors real-world behavior across all relevant dimensions. After validation, we archive the model parameters for future use. We then use the validated CDT to run simulations and conduct what-if analyses as part of a broader study. We assess these activities with predefined key performance indicators (KPIs) for each dimension of interest, enabling stakeholders to make informed, multi-objective decisions for system optimization.

3 MDPNML AS EXTENSION OF PNML

In this section, we introduce MDPNML, a PNML-compatible representation for MDSPNs. Our goal is to encode every modeling element required for MDSPN simulation while remaining compatible with standard PNML parsers and simulators. MDPNML preserves PNML's core structure (places, transitions, arcs, names, markings), while adding explicit dimension definitions and transition-level impact annotations for non-temporal dimensions, such as energy use and waste generation. MDPNML also records probability distributions, guard functions, and initial marking to ensure reproducibility and reliable model regeneration.

As illustrated in Figure 4, MDPNML supports four main uses after MDSPN model extraction from the real system (though it is not limited to these four):

1. **Tool-to-Tool Exchange:** MDPNML provides a PNML-compatible interchange artifact that enables model transfer and round-trip editing across tools.

2. **Unidimensional Model Analysis:** MDPNML enables per-dimension model conversion (time, energy, waste, CO₂), allowing analysts to isolate, inspect, and edit a single dimension independently of the others.
3. **Time-Oriented DTs:** MDPNML conversion can be exported as standard PNML and run unchanged on PNML-compliant simulators, supporting discrete event simulation and deployment as time-oriented DTs.
4. **CDTs:** The full MDPNML code preserves cross-dimensional behavior and transition impacts, enabling synchronized simulation and what-if analysis across multiple objectives.

In addition, MDPNML supports structural model validation by enabling comparison of the encoded model against the extracted MDSPN and by checking that places, transitions, arcs, and markings follow the same structure as the real-world system (when the system's ground truth model is available). Furthermore, MDPNML supports decision-making by allowing users to modify distributions, guards, and per-dimension impact parameters to evaluate single-objective and multi-objective what-if scenarios.

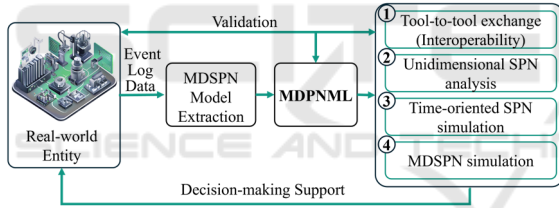


Figure 4: Uses of MDPNML after extracting MDSPN.

3.1 PNML Adaptation for MDSPNs

We adapt the elements and attributes of PNML to describe multidimensional models while preserving the PNML core. Our adjustments are guided by the MDSPNs semantics, which we detailed in (Khodadadi & Lazarova-Molnar, 2025a). In Table 3, we present the PNML extensions for MDSPNs, with all added or updated elements highlighted in green. In the following, we detail our adaptation of the main SPN elements for MDSPNs:

- **Design Scope and Requirements:** MDPNML encodes the control flow of the underlying SPN using PNML core constructs and introduces the additional information required to represent the MDSPN simulation. In particular, the extension supports explicit dimension declarations, optional place-level dimension tracking for idle accumulation, transition-level

impact annotations per dimension, and it preserves standard SPN features including transition timing distributions, guards, and arc multiplicity tag block.

- **Dimensions Definition:** We declare the modeled dimensions at the net level using a `<dimensions>` tag block, listing each as `<dimension name=" "/>`, such as time, Energy, CO₂ (Table 3, lines 3-5).
- **Places (P):** We extend place attributes for special places, such as idle states, by adding the optional `"dimensionTracked=" "`, which indicates the time duration that a token's presence in those places affects the defined dimension (Table 3, lines 6-11).
- **Transitions (T):** Each transition includes a structured `<impactedDimensions>` block. For every impacted dimension, an

Table 3: MDPNML code patterns for multidimensional extensions to PNML.

```

1 <pnml>
2 <net>
3 <dimensions>
4 <dimension name="Dimensions #N"/>
5 </dimensions>
6 <place id="pX" dimensionTracked="X">
7 <name><text>Name</text></name>
8 <initialMarking>
9 <text>N</text>
10 </initialMarking>
11 </place>
12 <transition id="tX" type="T/I">
13 <name><text>Name</text></name>
14 <distribution>
15 <type>Type</type>
16 <parameters><a>p</a></parameters>
17 </distribution>
18 <guard place="p_in" minTokens="1"/>
19 <weight>w</weight>
20 <!-- MD impacts -->
21 <!-- repeat per affected dimension -->
22 <impactedDimensions>
23 <impactedDimension name="Energy">
24 <impactType>rate|delta|dist</impactType>
25 <impactValue>"v"</impactValue>
26 </impactedDimension>
27 </impactedDimensions>
28 </transition>
29 <arc id="IDx" source="p or t" target=
  "p or t" type="input|output|inhibitor">
30 <inscription>
31 <text>multiplicity</text>
32 </inscription>
33 </arc>
34 </net>
35 </pnml>

```

`<impactedDimension name="...">` specifies an `<impactType>` that is either rate (applied over the firing duration), fixed (a one-off impact for that activity), or distribution (a stochastic impact). For rate and fixed, the impact is given as a numeric `<impactValue>` with an accompanying `<unit>`. For distribution, the impact is defined using a `<distribution>` element that specifies the distribution type and its parameters (Table 3, lines 12-28).

- **Arcs (A):** The standard PNML arc syntax is retained in MDPNML (Table 3, lines 29-33).

These adaptations enable multidimensional simulators to execute MDSPN models, while also maintaining the SPN control-flow skeleton's readability by PNML-compliant tools, and make it straightforward to convert the MDSPN model into time-oriented Petri nets, facilitating the exchange of models. In Table 4, we showcase the extended mapping of PNML core-model concepts to their corresponding XML elements and attributes in MDPNML, including support for multidimensional annotations.

3.2 Encoding Workflow

Our encoding methodology is focused on our developed MDSPNs simulation tool, MDPySPN (Khodadadi & Lazarova-Molnar, 2024b). The MDPNML parser builds the corresponding MDSPN model for simulation. The parser reads the dimension list, creates places, arcs, and transitions, with their types and distributions, attaches each `<impactedDimension>` as that dimension's effect, and wires input, output, and inhibitor arcs with their multiplicities. Missing optional fields will be set to the SPN defaults, such as `arc multiplicity = 1`, and `weight = 1`. The implementation of the MDPNML parser for MDPySPN runnable scripts is available in our GitHub repository (Khodadadi & Lazarova-Molnar, 2025c).

4 CASE STUDY

We demonstrate MDPNML of the heat-sealing packaging case study described in Section 2.2, modeled across three dimensions of time, energy consumption, and CO₂ emissions generation. We aim to demonstrate the application of MDPNML to convert the extracted MDSPN model into a PNML-

compatible representation and enable portable exchange of the MDSPN model. For this, we encode the discovered MDSPN model in MDPNML, load the model into MDPySPN, and simulate for 60 minutes to verify MDPNML executability, correct application of dimension-specific impacts, and reproducibility of the encoded model. The generated MDPNML file, its parser, and simulation scripts are available on our GitHub repository (Khodadadi & Lazarova-Molnar, 2025c).

Table 4: Mapping of MDPNML Extensions to XML Elements and Attributes.

Class	XML element	XML Attributes
PetriNet	<code><net></code>	PNML core: id: ID, type: anyURI MDPNML: dimensions: dimension name
Place	<code><place></code>	PNML core: id: ID; MDPNML: dimensionTracked: "N";
Transition	<code><transition></code>	PNML core: id: ID MDPNML: impactedDimensions, impactedDimension name: "Name"; impactType: "rate delta dist"; impactValue: "value";

4.1 MDPNML for the Case Study MDSPN Model

We collected the multidimensional event log generated by the case study system and used the MFPM to discover the system's MDSPN model. We then encoded the case study MDPNML using the profile introduced in Section 3. Table 5 presents an excerpt of the MDPNML code highlighting the key multidimensional constructions.

In contrast to the time-oriented PNML code in Table 2, our encoded MDPNML file preserves the PNML core (places, transitions, arcs, markings), while adding explicit multidimensional semantics. For instance: (i) the `<dimensions>` block that declares the modeled dimensions (time, Energy, CO₂); (ii) an idle state place annotated with `dimensionTracked="Energy"`; (iii) an immediate transition along with dimensional impacts via `<impactedDimensions>`; (iv) and standard PNML arcs for control flow.

4.2 Encoding MDPNML of the Case Study

We automatically parsed the case-study MDPNML code to construct the corresponding MDSPN in MDPySPN and simulated the resulting MDSPN

Table 5: MDPNML Excerpt for the heat-sealing packaging case study.

```

1 <pnml>
2 <net>
3   <dimensions>
4     <dimension name="time"/>
5     <dimension name="Energy"/>
6     <dimension name="Co2"/>
7   </dimensions>
8   <place id="p1"
9     dimensionTracked="Energy">
10    <name><text>Sealer standby
11    state</text></name>
12    <initialMarking><text>1</text></initialMarking>
13  </place>
14  ...
15  <transition id="T2" type="I">
16    <name><text>MT2</text></name>
17    <impactedDimensions>
18      <impactedDimension name="Energy">
19        <impactType>rate</impactType>
20        <impactValue>0.01</impactValue>
21      </impactedDimension>
22    </impactedDimensions>
23  </transition>
24  ...
25  <arc id="a1" source="T1" target="p2"
26    type="output">
27    <inscription><text>1</text></inscription>
28  </arc>
29  ...
30 </net>
31 </pnml>

```

model for 60 minutes. As illustrated in Figure 5, the generated MDSPN follows the process flow and multidimensional behaviors described in the case study (structural validation), where places, transitions, and arcs map one-to-one. For improved readability, we provide a cleaned-up figure of the MDSPN exported by MDPySPN. Because MDPNML is an extension of PNML, the same artifact can be loaded by PNML-compliant tools. Therefore, by ignoring the multidimensional annotations, the time-oriented SPN can be processed by a generic PNML parser.

5 SUMMARY AND OUTLOOK

Petri Net Markup Language (PNML) ensures that Petri-net models remain portable across tools by supporting distinct net types, providing a flexible, future-proof foundation for standardized interchange. In this work, we introduce Multidimensional Petri Net Markup Language (MDPNML), an extension of

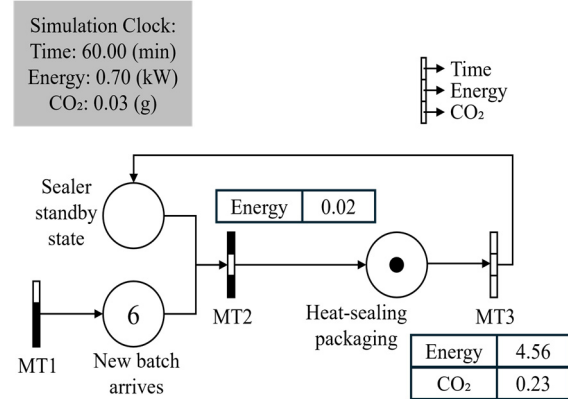


Figure 5: Case study MDSPN from executed MDPNML in MDPySPN at time 60 minutes.

PNML. MDPNML preserves the PNML core: places, transitions, arcs, and markings, while adding multidimensional semantics, such as energy consumption, and annotations that quantify each transition's impact on these dimensions. By standardizing multidimensional semantics and simulation parameters, MDPNML creates a direct path from model extraction to execution across MDSPN simulators. MDPNML also supports the conversion of Multidimensional Stochastic Petri Net (MDSPN) models into traditional (time-oriented) Stochastic Petri Nets (SPNs) for use with generic PNML-compliant tools, enabling deployment as both time-oriented and comprehensive Digital Twins.

Through an illustrative heat-sealing packaging case study with three dimensions of time, energy consumption, and CO₂ emission generation, we demonstrate MDPNML's applicability. We modeled the heat-sealing packaging line as an MDSPN and encoded the model in MDPNML. We then imported the MDPNML file into MDPySPN and executed the simulations, confirming that MDPNML preserves both structure and semantics of MDSPNs. Taken together, these results demonstrate that MDPNML provides a practical foundation for standardized exchange and simulation of MDSPNs. Moreover, MDPNML includes a systematic conversion of MDSPNs to (time-oriented) SPNs, enabling immediate reuse of established PNML-compliant toolchains when MDSPN is not required.

For future work, we intend to integrate process discovery with MDSPN feature extraction to automatically generate MDPNML code directly from real-world data. We, furthermore, aim to develop approaches to automatically update comprehensive Digital Twins as system objectives and their corresponding relevant dimensions change.

ACKNOWLEDGMENTS

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

REFERENCES

- Bause, F., & Kritzinger, P. S. (2002). *Stochastic Petri nets* (Vol. 1). Vieweg.
- Billington, J., Christensen, S., van Hee, K., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., & Weber, M. (2003). The Petri net markup language: Concepts, technology, and tools. In *International Conference on Application and Theory of Petri Nets*.
- Bonnefoi, F., Choppy, C., & Kordon, F. (2009). A discretization method from coloured to symmetric nets: Application to an industrial example. In *Transactions on Petri Nets and Other Models of Concurrency III* (pp. 159–188). Springer.
- Boschert, S., & Rosen, R. (2016). Digital twin—the simulation aspect. In *Mechatronic futures: Challenges and solutions for mechatronic systems and their designers* (pp. 59–74). Springer.
- Cassandras, C. G., & Lafortune, S. (2008). *Introduction to discrete event systems*. Springer.
- dos Santos, C. F., Micosky, A. L., de Freitas Rocha Loures, E., & Alves Portela Santos, E. (2024). A literature review of process mining and digital twin in the smart manufacturing context. In *International Conference of Production Research—Americas*.
- Friederich, J., Khodadadi, A., & Lazarova-Molnar, S. (2025). PySPN: A Python library for modeling, simulation, and event log generation of stochastic Petri nets. *Transactions of The Society for Modeling and Simulation International*.
- Gehlot, V., & Nigro, C. (2010). An introduction to systems modeling and simulation with colored Petri nets. In *Proceedings of the 2010 Winter Simulation Conference*.
- Hillah, L.-M., Kordon, F., Petrucci, L., & Treves, N. (2010). PNML framework: An extendable reference implementation of the Petri net markup language. In *International Conference on Applications and Theory of Petri Nets*.
- Holloway, L. E., Krogh, B. H., & Giua, A. (1997). A survey of Petri net methods for controlled discrete event systems. *Discrete Event Dynamic Systems*, 7(2), 151–190.
- Kaiyandra, D. R., Farizal, F., & Rakoto, N. (2024). Colored Petri nets for modeling and simulation of a green supply chain system. *IFAC-PapersOnLine*, 58(1), 306–311.
- Khodadadi, A., & Lazarova-Molnar, S. (2024a). Essential data requirements for industrial energy efficiency with digital twins: A case study analysis. *Procedia Computer Science*, 238, 631–638.
- Khodadadi, A., & Lazarova-Molnar, S. (2024b). *MDPySPN: Multidimensional stochastic Petri net simulation*. Retrieved December 2025, from github.com/atikh/MDPySPN
- Khodadadi, A., & Lazarova-Molnar, S. (2024c). Multi-flow process mining for comprehensive simulation model discovery. In *Proceedings of the 2024 14th International Conference on Information Communication and Management*.
- Khodadadi, A., & Lazarova-Molnar, S. (2025a). Multidimensional stochastic Petri nets: A novel approach to modeling and simulation of stochastic discrete-event systems. In *IEEE SMC*.
- Khodadadi, A., & Lazarova-Molnar, S. (2025b). Multi-flow process mining as an enabler for comprehensive digital twins of manufacturing systems. In *Winter Simulation Conference 2025*, Seattle, Washington.
- Khodadadi, A., & Lazarova-Molnar, S. (2025c). *MDPNML*. Retrieved December 2025, from github.com/atikh/MDPNML
- Lagartinho-Oliveira, C., Moutinho, F., & Gomes, L. (2024). Using Petri nets for digital twins modeling and deployment: A power wheelchair system case study. In *International Conference on Applications and Theory of Petri Nets and Concurrency*.
- Lazarova-Molnar, S. (2005). *The proxel-based method: Formalisation, analysis and applications* (PhD thesis). Otto-von-Guericke-Universität Magdeburg.
- Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4), 541–580.
- Petri, C. A. (1962). *Kommunikation mit automaten*.
- Rozinat, A., Mans, R., Song, M., & van der Aalst, W. M. (2008). Discovering colored Petri nets from event logs. *International Journal on Software Tools for Technology Transfer*, 10(1), 57–74.
- Stehno, C. (2002). Petri net markup language: Implementation and application. In *Promise 2002 – Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen*.
- Vaarandi, R. (2003). A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003)*.
- van der Aalst, W. M. (2018). Process mining and simulation: A match made in heaven! In *SummerSim*.
- van der Aalst, W. M. (2022). Foundations of process discovery. In *Process Mining Handbook* (pp. 37–75). Springer.
- VanDerHorn, E., & Mahadevan, S. (2021). Digital twin: Generalization, characterization and implementation. *Decision Support Systems*, 145, 113524.
- Wang, W., Zhang, Y., Gu, J., & Wang, J. (2021). A proactive manufacturing resources assignment method based on production performance prediction for the smart factory. *IEEE Transactions on Industrial Informatics*, 18(1), 46–55.
- Zimmermann, A. (2008). Colored Petri nets. In *Stochastic Discrete Event Systems: Modeling, Evaluation, Applications* (pp. 99–124).
- Zurawski, R., & Zhou, M. (1994). Petri nets and industrial applications: A tutorial. *IEEE Transactions on Industrial Electronics*, 41(6), 567–583.