technologies

MDPI

Article

A Data-Driven Framework for the Development of Reliability-Aware Business Process Digital Twins

Paolo Bocciarelli *, Manuel Fiorelli and Andrea D'Ambrogio

Department of Enterprise Engineering, University of Rome Tor Vergata, Via del Politecnico 1, 00133 Rome, Italy; manuel.fiorelli@uniroma2.it (M.F.); dambro@uniroma2.it (A.D.)

* Correspondence: paolo.bocciarelli@uniroma2.it

Abstract

Digital Twin (DT) technologies are increasingly adopted as valuable methods to support the analysis and optimization of Business Processes (BPs). The use of simulation-based techniques in the BPM context is not new. Nonetheless, existing approaches for BP simulation primarily focus on performance-related aspects. In this respect, this paper introduces a data-driven framework for the automated generation of reliability-aware DTs derived from event and state logs. The proposed method integrates process mining techniques, resource reliability modeling, and model-driven transformations to produce executable DT simulations capable of predicting the failure behavior of process resources. The proposed framework has been applied to the predictive maintenance domain, where DT-based simulations are used to estimate resource availability, identify possible failure scenarios, and support the timely scheduling of preventive maintenance interventions. A manufacturing case study shows that the proposed approach can enhance process operability and reduce downtime compared to conventional BP execution without DT-based reliability insights.

Keywords: BPM; BPMN; digital twins; model-driven development; predictive maintenance; process mining; simulation

1. Introduction

Modern Business Processes (BPs) are increasingly running on top of complex infrastructures that often include IoT-based and cyber–physical resources capable of detecting, storing, and processing large volumes of data. These infrastructures generate continuous data streams that constitute a valuable source of information for predictive analytics. The correct and efficient execution of a BP depends not only on its logical control flow but also on the behavior, availability, and health of the physical and software resources that constitute the underlying execution platform [1,2]. The availability of innovative strategies to deal with the reliability of execution platforms constitutes an essential requirement for executing BPs that orchestrate complex infrastructures such as production-critical equipment, redundant subsystems, and heterogeneous resources.

Business Process Management (BPM) provides methods and technologies to support stakeholders throughout the process lifecycle by offering tools for design, execution, monitoring, and continuous improvement [3]. Thus, innovative BPM approaches are required to explicitly integrate reliability predictions into process design and execution.

Data Science and Artificial Intelligence are increasingly reshaping enterprise systems and organizational practices from multiple perspectives, ranging from customer-oriented applications such as CRM [4] to the support of knowledge extraction [5]. In Business Process

Academic Editors: Sikha Bagui and Sotirios K. Goudos

Received: 19 January 2026
Revised: 13 February 2026
Accepted: 17 February 2026
Published: 20 February 2026

Management, execution data can be systematically analyzed to derive process models, performance indicators, and actionable insights on past and ongoing process behavior. When combined with simulation-based approaches, these technologies enable organizations to not only optimize the performance of BPs but also to adopt predictive strategies aimed at maintaining the reliability and availability of the underlying execution environment.

Furthermore, Digital Twins (DTs) offer capabilities that extend beyond traditional offline simulation or static process monitoring. A distinctive characteristic of DT lies in the continuous synchronization between the real system and its virtual counterpart. This enables executing effectively and timely *what-if* analyses based on real operational data, as well as the enactment of data-driven adaptation mechanisms. However, current DT research in BPM focuses primarily on *performance-centric* analysis—such as throughput estimation, bottleneck identification, and resource utilization optimization [6]. In contrast, the analysis of the *reliability* of the execution platform—crucial for processes that orchestrate physical devices, robotic resources, sensors, or distributed software components—remains comparatively underexplored.

This paper addresses this gap by proposing a data-driven framework for the automated generation of reliability-aware DTs of Business Processes. The proposed approach relies on process mining (PM) [7,8], which discovers and monitors processes by exploiting event logs collected from process-aware information systems (PAIS) [9], as well as state logs generated by the devices and resources that make up the execution platform [10]. To exploit this information for predictive purposes, the approach uses the DT paradigm, which is recognized as a promising solution to support the reliability analysis [11–13]. However, DT development requires systematic methods for the specification and evolution of the required process models. In this context, model-driven engineering (MDE) provides effective principles, methods, and standards to ease DT development [14,15].

In particular, the proposed approach exploits the Model-Driven Architecture (MDA) [16], i.e., the concrete realization of MDE principles promoted by the Object Management Group (OMG), which defines methods, tools, and standards for developing software systems through chains of model transformations that progressively refine higher-level abstract models into executable ones.

The primary contribution of this work consists of the specification of a data-driven methodology for building reliability-aware DTs, along with the design and implementation of a prototypal framework implementation. The proposed approach aims at supporting innovative performance- and reliability-oriented BPM approaches to be used in different domains, such as manufacturing, logistics, and service delivery. To illustrate the practical use of the framework, the proposed methodology is experimented in the context of predictive maintenance (PdM), a domain where reliability and failure prediction play a central role [17,18].

The remainder of this paper is structured as follows. Section 2 surveys the literature on predictive maintenance, digital twins, and data-driven process modeling and positions the proposed contribution. Section 3 introduces the necessary background on PyBPMN/eBPMN and DT modeling. Section 4 presents the proposed data-driven DT framework. Section 5 discusses the application of the approach in a manufacturing scenario. Finally, Section 6 summarizes the findings and outlines future research directions.

2. Related Work

In the context of BPM, this work proposes a framework for the low-code development of reliability-aware DT-based simulations that focuses specifically on the reliability evaluation. The objective of this section is to provide a comprehensive overview of the state of the art and to clearly position the contribution of this paper. This section is structured as

follows. Section 2.1 explores the role of DT in PdM efforts. Section 2.2 reviews model- and data-driven approaches to ease the development of DT. Section 2.3 discusses predictive maintenance from a general perspective and with a specific focus on its application to BPs. Section 2.4 illustrates how DT and AI can be integrated to enhance operational efficiency in industrial systems Finally, Section 2.5 summarizes the previous work of the authors and highlights the novel contribution proposed in this study.

2.1. Reliability Analysis and Digital Twins

DTs are increasingly being recognized as a powerful enabler for enhancing the reliability, resilience, and operational continuity of complex systems. The ability to perform real-time monitoring, scenario-based simulations, and data-driven forecasting makes DTs particularly well suited for reliability-aware applications [11,17,19–23].

In [17], an extensive literature review outlines the broad advantages of DT in real-time monitoring, failure prediction, asset life extension, and predictive control. This work also identifies key challenges, including the complexity of implementation and the dependency on infrastructure. The ability to analyze reliability and plan preventive interventions is one of the recognized advantages of using DT, as acknowledged in [19]. This work also highlights that DT acts as a powerful decision-making tool that supports operational and logistics management through continuous data acquisition from IoT devices. Such data is used in real time to configure a simulation model that mirrors the physical system, allowing predictive maintenance planning [20]. A DT enables the simulation-based comparison of different scenarios, thus allowing the identification of the most effective maintenance scenario. Furthermore, the continuous feedback loop between the DT and its physical counterpart enables ongoing validation and optimization of system operations [11].

The benefits of DT in the specific context of predictive maintenance have been empirically validated in [21], where such systems achieved fault detection accuracy up to 95% and operational cost savings of 25 to 30%, together with notable reductions in unexpected failures. The value of adopting DTs in the context of BPM is underlined in [22] where this paradigm is acknowledged.

A particularly relevant domain where DTs are gaining traction is Business Process Management (BPM), which includes both traditional organizational workflows and production processes governed by IoT-aware BPs [22]. In [23], a Digital Twin-based framework that enhances the entire BPM lifecycle is proposed. The framework addresses the process reliability by enabling runtime verification, anomaly detection, and proactive adaptation.

These contributions demonstrate that reliability analysis is increasingly recognized as a fundamental asset in the BPM domain for ensuring the operability of processes running on complex infrastructures. Building on these premises, this work advances the state of the art by introducing a low-code paradigm through standards and tools from the MDA. Moreover, unlike existing approaches, which are predominantly oriented toward production processes, the proposed method that leverages the data-driven generation of DTs from event logs and SysML models can be applied to any operational context, including organizational processes, where the use of advanced tools to analyze reliability and ensure process consistency is still underexplored.

An effective reliability analysis requires the adoption of appropriate methods and techniques. In this respect, various techniques can be employed, ranging from purely analytical approaches to those based on simulation. In particular, approaches exploiting Discrete Event Simulation (DES) are recognized as a robust and widely applied technique in the field of reliability engineering, thanks to its ability to effectively and flexibly model complex interactions among system components, non-linear dependencies, and dynamic maintenance policies [24].

Finally, a common reference architecture, often based on the IBM MAPE-K loop [24], includes six core subsystems: (i) a database, (ii) a process analytics component, (iii) a live monitoring component, (iv) a real-time analytical engine for model evaluation and KPI estimation, (v) a strategy evaluator to define corrective actions, and (vi) an executor to implement these actions. The architecture proposed in this work has been inspired by such contributions.

2.2. Data-Driven Approaches to Digital Twins

Model-Driven Development (MDD) and process mining (PM) can be effectively integrated in the design of DTs for BPM applications. In this context, MDD might play an essential role [25,26]. MDD can be effectively adopted not only for DT modeling but also for implementing the software components needed to automatically execute corrective actions on the physical system based on insights provided by the DT. Furthermore, PM techniques can be used to analyze event log repositories and generate BP models specifically suited for reliability evaluation [27]. MDD approaches, in turn, support the (semi)automated development of DT implementations starting from the extracted process model [25].

In complex systems, such as manufacturing environments, reliability is a critical factor. Traditional reliability analysis methods are static, require domain experts, and involve manually defined interventions that are typically performed after the occurrence of failure events [10]. More specifically, the need to overcome these limitations by using real-time data from physical systems to construct data-driven reliability models is underlined in [28].

In this regard, this work uses PMM techniques to build BP reliability-oriented models, which serve as a foundation for subsequent stages of analysis. Building on these models, MDD approaches are employed to systematically support the development of DT and to enable the automated update of real processes according to the insights derived from DT outcomes.

2.3. Predictive Maintenance and Business Processes

The relevance of PdM in both industry and academia is evidenced by the extensive literature available on the subject. In [17], a comprehensive and valuable systematic review of the literature emphasizes the importance of PdM by highlighting its motivations and providing a detailed taxonomy. Among the main challenges identified by this review, one of the most relevant is the lack of a reference architecture to guide the design of a system capable of predicting future failures and supporting the implementation of predictive maintenance actions.

In [29], a cyber–physical system (CPS) architecture is proposed for PdM in manufacturing domains. The paper also discusses a case study dealing with self-driving vehicles that operate in factories and warehouses and share transportation routes and tasks with human workers. The system leverages IoT middleware to collect sensor data in real time, enabling condition-based fault detection and prognosis. However, the main limitations of the proposed approach include infrastructure complexity, the need for stable network connectivity, and potential latency in data processing pipelines under high load conditions.

Although classical PdM is mostly applied to physical machinery, a growing trend involves predicting process-level issues through business process monitoring and mining. Although these approaches are promising for anomaly detection, the construction of realistic reliability models still depends heavily on the quality of the datasets analyzed [30]. Furthermore, the real-time applicability of such PdM techniques to running BP environments remains under investigation [31]. In this regard, this paper proposes a method based on DT to predict failures in production processes and possibly trigger automatic corrective actions or notifications for timely preventive maintenance.

2.4. Digital Twins, AI, and Optimization for Maintenance and Operational Efficiency

Recent studies have investigated the integration of Digital Twins (DTs), Artificial Intelligence (AI), and optimization techniques to enhance operational efficiency and maintenance strategies in industrial systems.

In [32], a comprehensive systematic review on the application of AI, machine learning, and deep learning techniques in machining operations is presented. The study highlights the role of DTs as AI-enabled enablers for predictive maintenance, process optimization, and energy efficiency, while also identifying key challenges related to data quality, scalability, explainability of AI models, and their integration within industrial systems. The authors emphasize the lack of unified and structured frameworks capable of combining DTs and AI-driven decision-making at scale.

In [33], a Dynamic Reliability Digital Twin combined with AI techniques is proposed to support predictive maintenance of industrial steel components. The approach leverages physics-based reliability models to generate synthetic datasets used for training machine learning predictors of Remaining Useful Life (RUL). While this contribution effectively addresses data scarcity issues and demonstrates the benefits of coupling DTs with AI models, it focuses on asset-level reliability analysis and does not address the integration of DT-driven maintenance actions within business process execution and management platforms.

In [34], a machine learning-driven Digital Twin architecture for the full lifecycle management of complex equipment is introduced. The proposed architecture embeds ML models within DTs to support predictive maintenance, health management, and operational decision-making across different lifecycle phases. Although the approach shows the advantages of combining DTs and ML for equipment-level optimization, it primarily targets physical asset management and does not explicitly consider process-aware modeling, process mining, or the execution semantics of business processes.

Compared to these approaches, the present work addresses a complementary and less explored challenge, namely the data-driven and model-driven generation of Digital Twins for Business Processes. Rather than embedding AI or machine learning models within the DT, the proposed framework exploits process mining techniques to derive executable simulation models from event and resource logs and relies on an algorithmic evaluation of simulation outcomes to generate corrective maintenance actions. Predictive maintenance is therefore treated as an illustrative application scenario, while the main contribution lies in providing a unified, low-code framework that integrates DT generation, simulation, and feedback enactment directly within a BPM environment.

2.5. Past Work and Contribution Outline

This paper explores the use of DT to enable the predictive maintenance of BPs and introduces an architectural framework for the automated generation of DTs with reliability-driven data. In this respect, this paper exploits achievements and results discussed in the authors' previous work:

- The idea of jointly using process mining (PM) and simulation in a predictive process mining (PPM) approach has been introduced in [35].
- A low-code framework for the analysis of BP collaboration through simulation has been proposed in [36]. The framework, which relies on principles introduced in the MDE field, makes use of methods, standards, and technologies provided by the MDA.
- The java-based framework for the modeling and simulation of performance- and reliability-aware BPs, namely PyBPMN/eBPMN, has been introduced in [37].

Unlike previous approaches, this paper employs a full DT in which the simulation model is continuously synchronized with the real system. The insights obtained from the analysis of the simulation results are then used to adapt the physical system and prevent

service interruptions or critical failures. By integrating PM techniques to derive reliability models from XES-compliant event logs, the proposed approach addresses the limitations of traditional methods and enables a data-driven, adaptive PdM strategy.

3. Background

This section provides the necessary background for this work. Specifically, Section 3.1 summarizes PyBPMN/eBPMN, the framework for performance- and reliability-enabled BP modeling and simulation, while Section 3.2 briefly introduces the DT paradigm.

3.1. Performability-Enabled BPMN Extension (PyBPMN) and eBPMN

This section outlines the PyBPMN/eBPMN performability framework. PyBPMN is a performability-enabled BPMN extension, which addresses the specification of performance and reliability properties of BPs [37]. The PyBPMN extension is based on principles and standards introduced in the model-driven engineering field, specifically by the OMG's Model-Driven Architecture (MDA) [16]. Such an extension addresses the non-functional characterization of a BP according to four different dimensions:

- **Workload**: to model the workload for the tasks in the process;
- **Performance**: to specify the performance properties, i.e., efficiency-related properties such as the service time, associated with single tasks;
- **Reliability**: to specify the reliability properties of the resources that tasks depend on to carry out the work requests;
- **Resource management**: to describe the execution platform actually used for executing the BP.

Figure 1 shows the *PyBPMN metamodel*, i.e., a class diagram in which classes define the constructs to be instantiated at the model level. The figure also shows the relations between novel *PyBPMN metaclasses* (i.e., classes in the metamodel) and extended BPMN ones.

The PyBPMN-based modeling approach does not introduce a separate notation for representing the BP under study. Rather, the functional requirements of the process to be analyzed are initially used to specify a standard BPMN model. Then, the model is enriched with annotations that capture the non-functional requirements, according to elements (or) introduced in the PyBPMN metamodel.

As an example, the performance characterization of a BP is described in PyBPMN using the abstract metaclass `PaQualification`, which can be specialized as `PaService` to specify the service demand for executing a single request. Regarding the representation of the reliability of BP resources, the PyBPMN metamodel includes the abstract metaclass `DaQualification`, which can be specialized as `DaFailure`, for specifying the failure-related properties, e.g., the probability distribution of failure or repair events. A detailed description of the PyBPMN extension is given in [37].

PyBPMN provides a resource definition that is complementary to the standard BPMN one. In BPMN, a resource is an abstract role involved in an activity, whereas the PyBPMN resource definition allows the specification, at design time, of the real entities that perform the activities, along with their non-functional properties. A PyBPMN resource can model a human worker, a piece of equipment, a functional division of an organization, an autonomous system, a web service, or any other entity which can be used to carry out a service request.

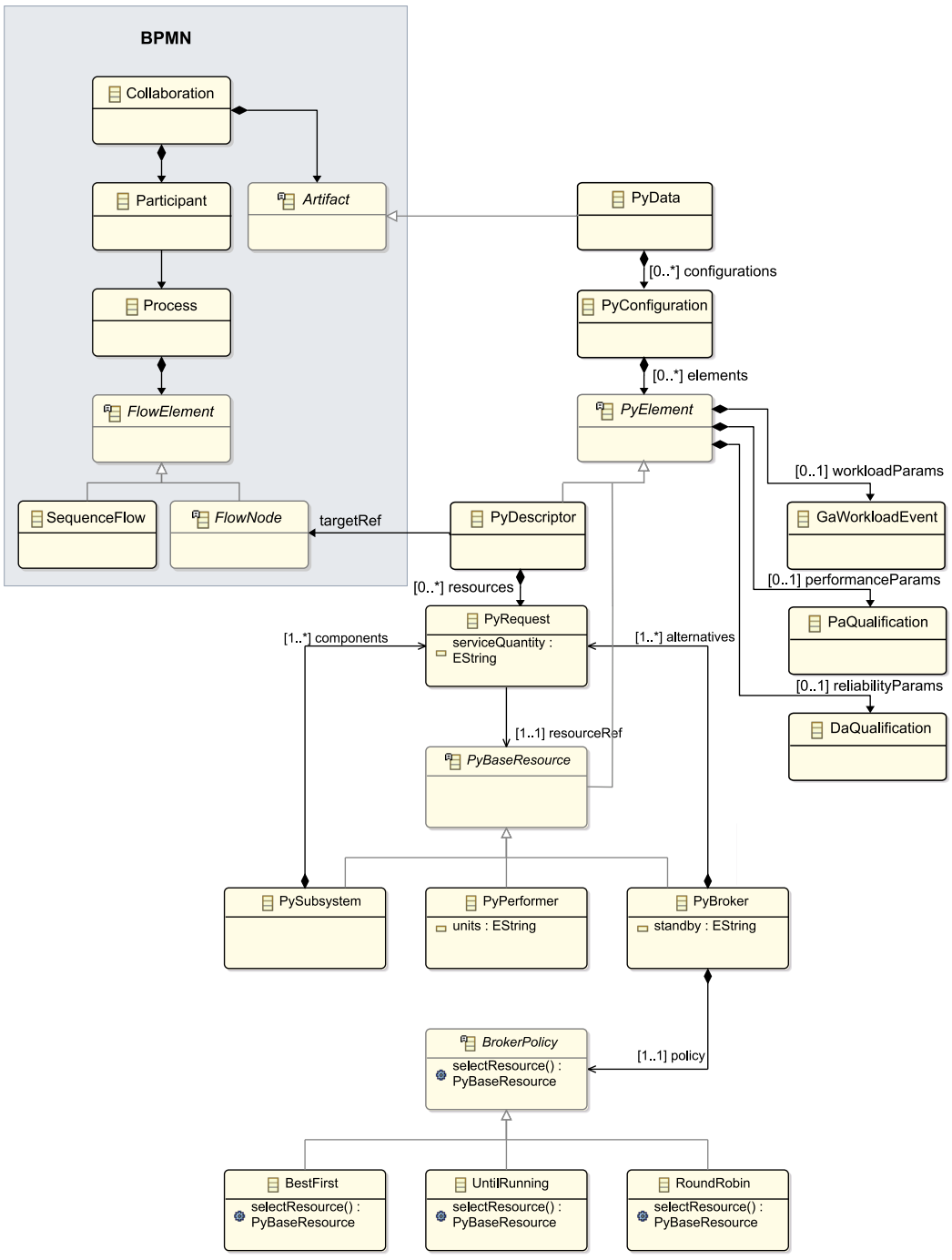


Figure 1. Relations between BPMN and PyBPMN metaclasses.

The capability of effectively modeling a complex execution platform composed of multiple and potentially redundant resources represents a central feature of PyBPMN. The core element is constituted by the `PyBaseResource` metaclass, which models a resource that is part of the execution platform. It can be specialized in terms of the concrete classes `PyPerformer`, `Broker`, and `PySubsystem`. The `PyPerformer` class represents the actual work performer that executes the service requests by selecting the required number of performers from a pool of available resources. The `PySubsystem` class allows the modeling of a complex resource defined in terms of its elementary components. Finally, the *PyBroker* class represents a resource manager responsible for dynamically selecting a resource from a set of candidate resources to execute a service request. The selection is performed according to configurable *policies* that determine how the current resource is initially selected and

replaced if needed, e.g., in case of unavailability. The broker supports different selection strategies, such as *round-robin*, *fail-over*, or *priority-based* selection, and explicitly models the management of unselected resources through standby policies (e.g., *hot* or *cold* standby). PyBPMN models can be directly simulated by using eBPMN, a Java-based domain-specific simulation framework introduced in [38]. The eBPMN framework has been explicitly designed according to the execution semantics defined in the BPMN 2.0 specification [39]. More specifically, BPMN flow elements such as tasks, gateways, events, and sequence flows are implemented as eBPMN entities that handle the execution of process instances through a token-based mechanism, ensuring compliance with BPMN execution rules. Indeed, eBPMN natively supports the simulation of complex process structures, including parallel branches, synchronization points, and concurrent task execution, which are directly mapped to corresponding eBPMN elements and execution behaviors.

3.2. Digital Twin

The Digital Twin (DT) concept has gained significant attention in recent years, particularly in the industrial and engineering domains. A DT is a virtual replica of a physical system that is continuously updated with real-time data to enable simulation, analysis, and optimization [11]. DTs have been adopted in various sectors, including manufacturing, healthcare, and infrastructure management. One of the most promising applications of DT technology is in predictive maintenance, where real-time monitoring and simulation-based techniques are jointly employed to prevent failures, reduce downtime, and optimize maintenance schedules.

A key distinction must be made between a Digital Twin and a Digital Shadow [40]. Although both concepts involve digital representations of physical assets, they differ in their level of interaction and data exchange. A Digital Twin establishes a bidirectional link between the physical and digital systems, enabling not only passive data collection but also active decision-making and control. The continuous feedback loop allows for dynamic adjustments and real-time optimization of the physical asset. In contrast, a Digital Shadow is a more limited form of digital representation in which data flows only in one direction, from the physical object to its digital counterpart, without active feedback or influence over the physical entity. Although a Digital Shadow can still provide valuable insights by analyzing historical and real-time data, it lacks the ability to directly interact with or modify the physical system based on predictive analytics.

DT adoption has emerged as a promising approach to support predictive maintenance, where it can be used to prevent system failures, optimize maintenance schedules [13], and also to enable proactive decision-making [10,17]. By integrating IoT sensor data, process mining algorithms, and simulation-based techniques, DTs forecast potential failures and recommend optimal maintenance actions [41], thus improving reliability and reducing operational costs.

3.3. Predictive Maintenance

Predictive maintenance (PdM) is an approach that takes advantage of data analysis to predict the failure of system components and schedule timely interventions before breakdowns occur. Its relevance in many operational fields is evidenced by the available literature that underlines the benefits and challenges of this technique.

Specifically, PdM has emerged as a key paradigm to ensure the reliability and availability of complex systems, shifting the focus from reactive interventions to proactive strategies that anticipate failures and reduce downtime. Traditional PdM techniques are based on sensor monitoring, trend analysis, and fault prediction, providing significant benefits in terms of cost reduction, improved productivity, and lower downtime [42,43]. When applied

to organizational and operational Business Processes (BPs), PdM requires the capability to continuously monitor execution platforms, analyze the condition of heterogeneous resources, and adapt process execution to prevent failures.

Various systematic literature reviews have been proposed that underline the potential of PdM to reduce downtime and improve operational efficiency [44,45]. Such contributions also identified key challenges, such as data quality, interoperability, and the lack of standardized frameworks.

As underlined in [18], existing PdM approaches often rely on static models or require extensive manual calibration, which limits their adaptability and scalability. The adoption of data-driven techniques and DT constitutes a promising opportunity to enable proactive decision-making [10,17]. DTs, by using IoT data and simulation models, enable accurate failure prediction and optimal maintenance planning [41].

Further research is needed to develop integrated frameworks for DT-based predictive maintenance of BPs. This work contributes by proposing a model-based DT architecture specifically designed for reliability-aware BP monitoring and management.

4. Data-Driven Framework for the Reliability-Aware DT

The conceptual architecture of the proposed framework is shown in Figure 2. The objective of the architecture is to identify systems, actions and data flows for supporting the development of a DT focusing on the reliability analysis of BPs.

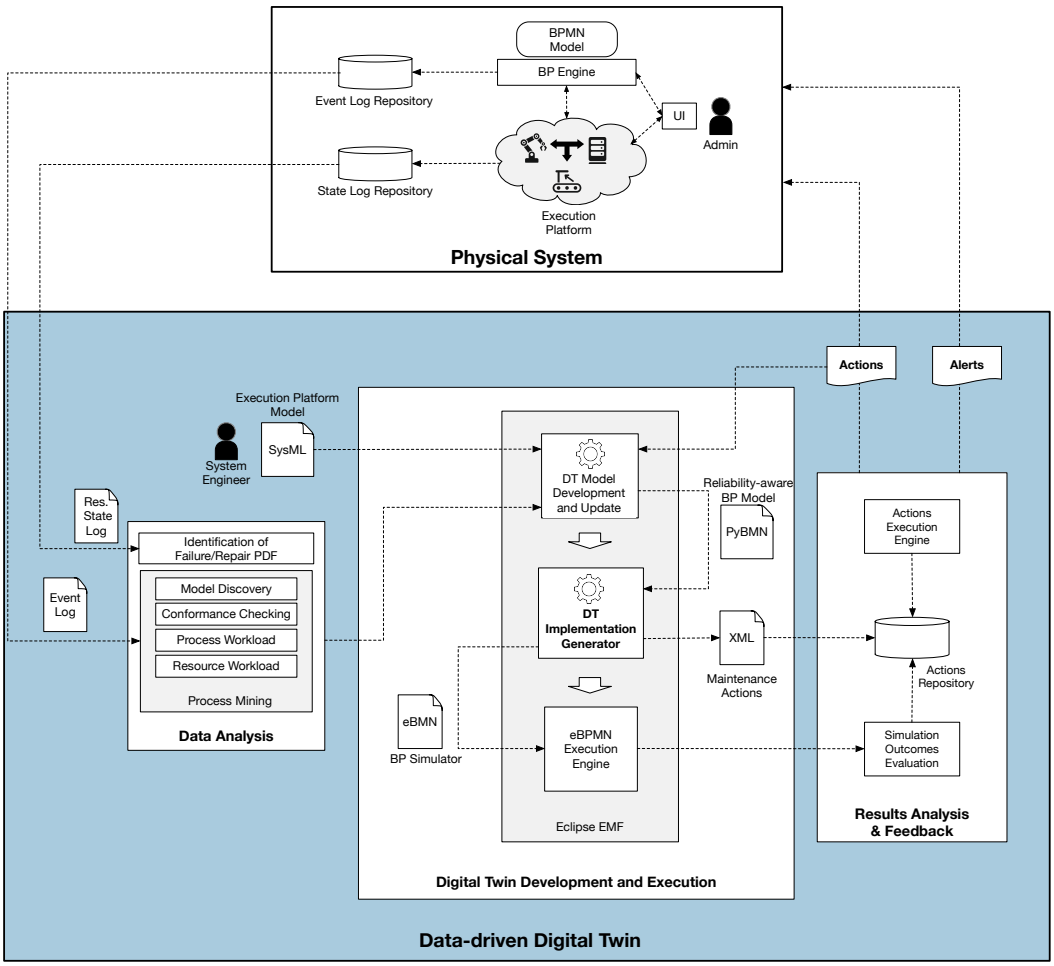


Figure 2. Architecture of the BP predictive maintenance framework.

The framework includes the physical system (PS) and the related data-driven DT. As noted in Sections 1 and 3.2, the proposed approach takes advantage of a complete DT

paradigm. More specifically, on the one hand, real-time data gathered from the physical system are used to develop a simulation model of the PS. On the other hand, simulation-based predictions are analyzed to generate appropriate actions that support variation in the PS configuration, to address reliability issues. Furthermore, the DT configuration is continuously updated to remain aligned with its physical counterpart.

The PS is structured into three main subsystems:

- **BP Engine**, which is a Process-aware Information System (PAIS) responsible for executing the BP that models the operational logic of the system and orchestrates the resources and equipment that make up the underlying execution platform. It is also capable of tracking all events that occur during the execution of the BP.
- **Execution Platform**, which is composed of a set of hardware resources (i.e., sensors, actuators, equipment in a production line, etc.) and/or software resources (i.e., web services).
- **User Interface**, which enables the system administrator to deploy the BP and manage its execution. Through the user interface, the administrator is also able to monitor the state of the execution platform's devices.
- **Event Log Repository**, which stores the event data captured by the BP engine. In the remainder of this work, we assume that the event log conforms to the IEEE eXtensible Event Stream (XES) standard [46]. It is worth noting that the use of a different formalism does not alter the overall structure of the framework. The format of the event log is discussed in Section 4.3.
- **State Log Repository**, which stores the operational state of the various devices in the execution platform. The different kind of information stored in the state log is illustrated in Section 4.3.

The data-driven DT component consists of three subsystems:

- **Data Analysis**: the subsystem responsible for analyzing the data contained in the logs to discover the information needed for the initial development of a reliability-aware BP model and for its update to ensure that it remains aligned with changes in the physical system. It is described in detail in Section 4.3.
- **Digital Twin Development and Execution**: the subsystem that constitutes the actual DT. Specifically, the information produced as output by the data analysis component is used to create (and update when needed) the reliability-aware BP model. The latter is then used to generate the implementation of the simulation system, whose execution allows one to obtain predictions on the availability of the physical system. The *Digital Twin Development and Execution* subsystem is described in detail in Section 4.4.
- **Results Analysis and Feedback**: the subsystem responsible for analyzing the data produced by the simulator and providing the necessary feedback to the physical system. This component is described in Section 4.5.

The proposed approach aims to analyze BPs from a reliability perspective and, when needed, to timely enact appropriate corrective actions. A key aspect for effectively achieving this objective lies in the data-driven nature of the framework: through data and models, the DT is built and constantly aligned with the corresponding physical system. In this regard, the availability of a suitable resource conceptual model enables designers to specify the structure of composite devices, thereby supporting effective predictions of resource reliability. In addition, automated methods are introduced to exploit simulation outcomes to specify, schedule, and execute appropriate actions aimed at mitigating reliability issues.

Specifically, the method relies on the following information sources:

- **State and event log repositories**, used for discovering BP models along with the relevant performance- and reliability-oriented characterizations.

- **A SysML model**, which specifies the allocation of process tasks to execution resources by use of a Block Definition Diagram (BDD), and the detailed structure of each resource composing the execution platform throughout an Internal Block Diagram (IBD).

These artifacts are processed through model transformations that generate (either fully automatically or with subsequent manual refinement):

(i) an intermediate reliability-aware simulation model;
(ii) the executable implementation of the simulation model;
(iii) the specification of corrective actions.

In this regard, the rationale for resource modeling is presented in Section 4.1, while the adopted strategy for corrective actions specification is discussed in Section 4.2. Finally, the three data-driven DT subsystems are described in Sections 4.3–4.5.

4.1. Execution Platform and Resource Modeling

In many real-world scenarios, the various resources orchestrated by a BP could be complex systems themselves. An execution platform can be made up of equipment structured into multiple subsystems or include multiple redundant devices that ensure operability in the event of failures.

To enable the appropriate representation of such scenarios with the required degree of flexibility, and according to the design principles at the basis of PyBPMN, an execution platform (EP) is a set defined as $EP = \{R_0, R_i, \ldots, R_N\}$

Each resource R_i with $i = 1, \ldots, n$ represents a hardware or software system that is orchestrated by a BP engine to fulfill a well-defined service request. More specifically, each resource R_i represents one of the following entities:

- **Atomic Resource**: the structure of R_i does not require further specification. It is treated as an atomic entity capable of processing a service request coming from the BP engine. In the PyBPMN framework, this kind of resource is identified as a *Performer*;
- **Structured Resource**: The resource R_i is structured in various subsystems. Formally, $R_i = \{R^s_{i1}, R^s_{i2}, \ldots, R^s_{ij}, \ldots, R^s_{iM}\}$, where each resource R^s_{ij} represents a subsystem in which R_i is structured. In order to process a service request, all M subsystems R^s_{ij} must simultaneously be in an *operational state*, otherwise the whole subsystem R_i is in a failed state and the service request cannot be processed. In the PyBPMN framework, such resources are referred to as *subsystems*.
- **Redundant Resource**: The resource R_i represents a collection of various redundant devices, $R_i = \{R^r_{i1}, R^r_{i2}, \ldots, R^r_{ij}, \ldots, R^r_{iK}\}$, where each resource R^r_{ij} is a redundant device that operates according to either a cold-standby or hot-standby paradigm: initially, one device, e.g., R^r_{i1}, is set to the *active state*, while the remaining $K - 1$ redundant devices are in an *inactive state*. The active resource is responsible for executing the service requests directed to the redundant subsystem R_i. In case of failure, the active resource enters a *failed state*, and a different inactive resource is selected and configured as the new active one. To process a service request, at least one of the K redundant devices R^r_{ij} must be in an operational state; otherwise, the redundant subsystem is considered to be in a failed state and the service request cannot be executed. In the PyBPMN framework, such resources are referred to as *brokers*;

The metamodel that conceptually specifies the structure of the resources is represented in the class diagram shown in Figure 3.

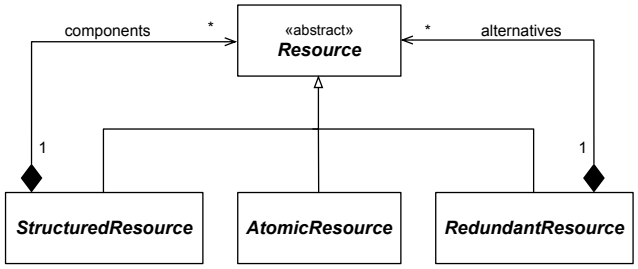


Figure 3. Conceptual model of resources.

The BP engine is therefore configured so that each task of the BP is associated with one or more resources R_i among those that make up the execution platform EP. It should be underlined that if R_i (or one of its subsystems) is a redundant subsystem consisting of K device resources, the BP maintains its operability through a reconfiguration operation in which the failed resource is replaced at runtime with an available backup.

Based on the resource metamodel mentioned above, the *SysML Resource Profile* provided in Figure 4 is introduced. The profile enables the characterization of the execution platform components according to the resource metamodel and, ultimately, provides the information required to drive the automated generation of the PyBPMN extended model, as discussed in Section 4.4.1.

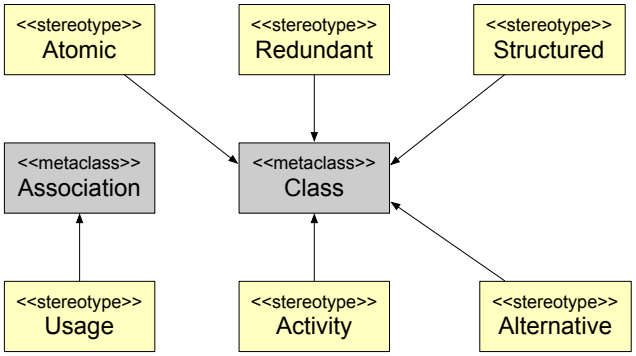


Figure 4. SysML resource profile.

4.2. Corrective Actions Modeling

The resources that compose the execution platform of a BP are subject to failure. The DT introduced in this work is not limited to predicting resource failures using simulation techniques. An essential aspect of DT lies in the ability to exploit predictions of system behavior and to provide feedback which impacts the system state. In this work, corrective actions aim to enable or support preventive maintenance, with the ultimate goal of minimizing downtime and preserving system operability. To this end, the metamodel shown in Figure 5 is introduced.

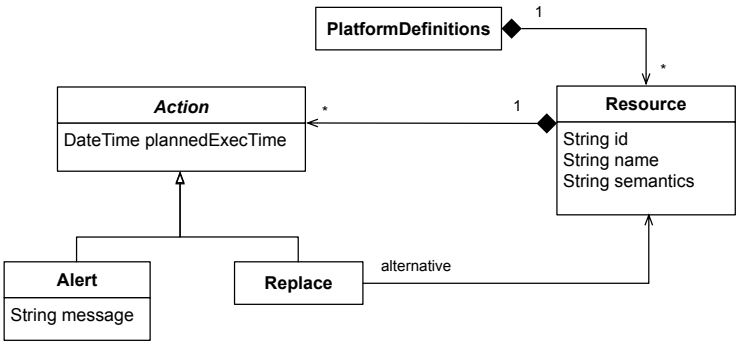


Figure 5. Conceptual model of corrective actions.

An execution platform is composed of n elements of type *Resource*, which owns three attributes: a unique identifier (`id` attribute), a name (`name` attribute), and the description of the resource semantics (`semantics` attribute).

For each resource, it is possible to specify an arbitrary set of corrective actions (meta-class *Action*), each to be executed at a given time (attribute `plannedExecTime`). The current framework supports two types of corrective actions, represented by the two subclasses of *Action*:

- **Alert** : specifies a text message sent to the administrator;
- **Replace**: identifies a backup resource, part of the execution platform, which can replace the active resource before its failure to preserve the system operability (`alternative` reference field).

The conceptual model of corrective actions hereby introduced enables effective system governance and ensures compliance with safety constraints. Only actions explicitly marked as automatically applicable, such as the replacement of redundant resources, can be enacted without human intervention (while still under human supervision). In all other cases, the framework generates notifications to inform system administrators about predicted failures, preserving a *human-in-the-loop* approach for safety-critical decisions.

Listing 1 shows an example of the XML description of preventive maintenance actions for the resource `RoboticArm-01`. Assuming that simulation analysis predicts a potential failure on 15 January at 17:23, three corrective actions are generated:

1. A first notification message is sent to the administrator, to inform in advance about the expected failure of a resource `RoboticArm-01`;
2. A second message is sent to notify the start of the replacement procedure;
3. The automatic replacement of resource `RoboticArm-01` with its backup `RoboticArm-01B` is enacted.

Listing 1. Example of corrective actions for a resource.

```
<?xml version="1.0" encoding="UTF-8"?>
<act:executionplatform xmlns:act="http://sel.uniroma2.it/actions">
  <act:resource id="RA-01" name="RoboticArm-01"
       semantics="Automated device that equips the assembly station">
    <act:alert
        plannedExecTime="2026-01-08T09:00:00"
        message="Expected failure on 15/1/2026 17:23:09"/>
    <act:alert
        plannedExecTime="2026-01-15T23:59:00"
        message="The resource RoboticArm-01 is being replaced with
    RoboticArm-01B"/>
    <act:replace
        plannedExecTime="2026-01-15T23:59:00"
        alternative="RA-01B"/>
  </act:resource>
  ...
</act:executionplatform>
```

4.3. Data Analysis

In the proposed framework, the adoption of a data-driven approach is essential to ensure that the DT is aligned with the current state of the PS, based on data collected at each analysis cycle. Such data, discovered from logs, captures task execution information and resource failure and repair events. Therefore, the availability of appropriate data is crucial to develop an effective simulation model which is able to capture the reliability-

oriented characteristics of the PS. As underlined in Section 4.1, in the addressed scenario, the resources that make up the execution platform are orchestrated by a BP engine.

According to a widely shared assumption [7,9,10],

- The BP engine (i.e., the PAIS) orchestrates the hardware and software resources capable of accomplishing the needed service requests, according to the execution flow specified by the BP model. The execution of each BP task generates appropriate events.
- The PAIS captures event-related data and records them in the Event Log Repository (see Figure 2).
- As generally each PAIS might capture various information, which varies depending on the specific operational context, we assume that the recorded data are compliant with the XES standard and include standard attributes that identify a *trace* and the *activity name*. In addition, there are attributes provided by three standard extensions [46]: `Start Timestamp` and `Completion Timestamp` from the *lifecycle* and the *time* extensions, and `Resources` and `Role` from the *organizational* extension. Figure 6 shows an example of an event log illustrating the various information recorded for each event.

We also assume that each resource can detect and record its own state in a state log [10]. Although, in general, resources can track different states (e.g., operational, busy, idle, failed, etc.), for the purposes of this research, we assume that the state log is defined by the following attributes:

- **Resource Id**, which uniquely identifies a resource $R_i \in EP$;
- **Timestamp**, in the format `YYYY-MM-DD HH:mm:ss.sss`;
- **State**, which can take values `operative` and `failed`;

Figure 7 illustrates the structure of the state log.

Start Timestamp	Complete Timestamp	Activity	Resource
2025/09/02 00:00:00.000	2025/09/02 00:01:48.000	Component Feeding	PickAndPlaceTool
2025/09/02 00:01:48.000	2025/09/02 00:02:58.000	Component Feeding	ComponentConvBelt
2025/09/02 00:00:06.000	2025/09/02 00:08:01.000	Component Feeding	PickAndPlaceTool
2025/09/02 00:08:01.000	2025/09/02 00:09:58.000	Component Feeding	ComponentConvBelt
2025/09/02 00:02:58.000	2025/09/02 00:12:23.000	Product Assembly	AssemblyStation
2025/09/02 00:09:58.000	2025/09/02 00:18:10.000	Product Assembly	AssemblyStation
...	...	...	...

Figure 6. Structure of the event log.

Timestamp	ResourceID	State
2025/09/02 00:00:00.000	PP-01	Operative
2025/09/02 00:00:00.000	RA-01	Operative
...	...	..
2025/09/05 07:29:12.000	RA-01	Failed
2025/09/05 015:12:05.000	RA-01	Operative
...	...	...

Figure 7. Structure of the state log.

As shown in Figure 2, the data analysis phase is structured into two main steps. The first step, performed by the *process mining* component, focuses on the analysis of the event log, while the second step processes the resource log through the *Identification of Failure/Repair PDF* component. The joint objective of this phase is to derive a simulation model of the process under study, consisting of a BPMN representation that captures the process flow together with the performance and reliability parameters required for accurate simulation. Distinct approaches are adopted for the analysis of the event and resource logs; further details are provided in Section 4.3.1 and Section 4.3.2, respectively. The information retrieved from the log repositories is then used by the *DT Development*

and Execution component to build the digital counterpart of the physical system and obtain predictions about the resource failure. This aspect is discussed in Section 4.4.

4.3.1. Event Log Analysis

For the event log analysis, PM tools are employed to discover information such as the structure of the BP model, the process workload, tasks' performance characterization (e.g., the average service time), routing probabilities, etc.

In this respect, it is worth noting that real-life logs might exhibit quality issues such as missing events, incomplete traces, and attribute sparsity (e.g., missing timestamps or resource identifiers). As extensively discussed in [7,8], several existing algorithms and approaches can be adopted to mitigate these quality issues, so as to discover process models that achieve an appropriate level of quality in terms of fitness, precision, generalization, and simplicity. As an example, missing lifecycle transitions (e.g., only completion timestamps being recorded) can be addressed through lifecycle repair and timestamp interpolation techniques, which reconstruct plausible start times and ensure a consistent ordering of events. Incomplete traces, where prefixes or suffixes of the execution are missing, can be mitigated by alignment-based repair methods and by exploiting behavioral constraints derived from the discovered process model to infer the most likely missing events. Attribute sparsity, such as missing resource or role annotations, can be reduced by adopting probabilistic assignment rules based on attribute similarity. Additional preprocessing steps, such as noise filtering and outlier removal, further improve the robustness of the discovered model. The use of such approaches to derive a high-quality BP model has already been discussed in previous works [35] and will not be further detailed here. Interested readers are referred to the relevant literature. Finally, as underlined in Section 5, the example application discussed in this work makes use of synthetic logs that are not affected by the above-mentioned quality issues. Nonetheless, appropriate preprocessing techniques are commonly provided by the ProM process mining tool that is part of the proposed framework prototype implementation.

As clarified in Section 5, the prototypal framework implementation includes the ProM tool [47] to perform event log analysis and ultimately discover the process structure, workload characteristics, and resource-related performance parameters.

The process structure is discovered through the IDV plugin, which derives a Petri Net from the event log and converts it into a BPMN model via the *Convert Petri Net to BPMN* plugin. This plugin also evaluates the fitness of the discovered model and provides token statistics for each node, which are exploited to compute routing probabilities for BPMN gateways. These parameters are incorporated into the PyBPMN model and used to configure the corresponding eBPMN *Gateway* elements of the executable simulation. Process workload parameters are obtained from the Trace Analysis view of the ProM Log Viewer to estimate the inter-arrival time of process instances and configure the eBPMN *Source* elements. Resource workload is derived from the Paths and Service Time view of the Inductive Visual Miner plugin, combining task service times with resource processing rates; the resulting parameters are used to configure the eBPMN *Task* elements.

4.3.2. Resource State Log Analysis

The state log analysis requires a dedicated component in charge of determining, for each resource, the mean values of failure (MTTF) and repair (MTTR) events, as well as their corresponding probability distribution functions (PDF). This component is referred to as *Identification of Failure/Repair PDF* in Figure 2.

In this respect, various methods can be adopted. Maximum likelihood estimation (MLE) consists of identifying the parameter values that maximize the probability of ob-

serving the empirical data, thus providing statistically grounded estimates of failure and repair distributions. Alternatively, Bayesian estimation methods can be employed to infer the distribution parameters by combining prior knowledge with observed evidence. The prototypal implementation used to analyze the case study discussed in Section 5 relies on MLE, as it offers a straightforward and well-established approach for parametric inference in the presence of sufficient historical data. The estimated parameters are included in the PyBPMN model and are finally used to configure the eBPMN *Performer* elements of the executable simulation. It should be underlined that the proposed Digital Twin framework is agnostic to the specific estimation technique adopted: the use of a particular estimator does not alter the conceptual architecture of the method, but only the way in which the PDF parameters are computed from the state log.

4.4. Model-Driven Digital Twin Development and Execution

The development of the simulation that constitutes the actual DT of the physical system is carried out in two steps. First, the data extracted by the *Data Analysis* component are used to build a reliability-aware model of the BP. Then, this model is used to generate the implementation of the executable simulation.

Furthermore, the DT implementation component is also responsible for the automatic generation of maintenance actions. The following Sections 4.4.1–4.4.3 discuss these steps.

4.4.1. Generation of the Reliability-Aware BP Model

The simulation model, as described in Section 3.2, is a BPMN model annotated according to the PyBPMN extension. This model is automatically generated through a `model-to-model` transformation implemented by the *Model Development and Update* component. The transformation takes as input the data extracted by the *Data Analysis* component and a SysML model that describes the execution platform and the allocation of BP tasks to resources. More specifically, the transformation takes as input the following information:

- A BPMN model serialized to an XMI document, discovered by the process mining tool. As discussed in Section 5, in the paper case, the IDV plugin provided by the ProM Tool has been used.
- The performance-related parameters that complete the BP model:
  - *Process Workload:* the PDF along with its characterizing parameters (e.g., mean value, variance, etc.) describing the requests' inter-arrival time.
  - *Task Service Time:* the PDF and the relevant characterizing parameters describing that models the service time of each task.
  - *Resource Cardinality and Routing Probabilities:* the number and type (role) of resources involved in each task.
  - *Resource Cardinality and Routing Probabilities:* the routing probabilities of each exclusive diverging gateway computed from the number of occurrences of task execution in each trace.
- The reliability-related parameters that characterize the execution platform, which are defined in terms of the mean values of failure (MTTF) and repair (MTTR) events, as well as their corresponding probability distribution functions (PDFs).
- The SysML model serialized to an XMI describing the execution platform.

The transformation rationale is outlined in Algorithm 1.

The Digital Twin is generated and subsequently updated through data analysis cycles triggered at configurable time intervals. At each cycle, the simulation model and its parameters can be regenerated based on newly collected logs, allowing the DT to reflect the current system behavior.

Algorithm 1 Generation of the Annotated Performance Model

Input: *bpmn*: process model in XMI format;
sysml: execution platform in XMI format;
exec_platform: list of resources composing the execution platform;
pef_parameters: set of performance parameters of each task;
rel_parameters: set of reliability parameters of each resource.
Output: *pybpmn*: reliability-aware bpmn model.

```
Init pybpmn ← bpmn.
for all resource r_i in exec_platform do
  retrieve b_r block in sysml BDD such that b_r.name = r_i.name
  if b_r is stereotyped as «atomic» then
    pybpmn ← new «PyPerformer» annotation where:
        serviceTime ← pef_parameters[r_i]
        MTTF,MTTR ← rel_parameters[r_i]
  else if b_r is stereotyped as «structured» then
    partSet ← parts composing b_r in sysml IBD
    pybpmn ← new «PySubsystem» annotation where:
        name ← b_r.name
        components ← partSet
  else if b_r is stereotyped as «redundant» then
    altSet ← parts composing b_r in sysml IBD stereotyped as «alternative»
    pybpmn ← new «PyBroker» annotation where:
        name ← ri.name
        alternatives ← altSet
  end if
end for
for all task t_i in pybpmn do
  for all a_i associations in sysml BDD do
    sBlock ← a_i.source
    if sBlock.name = t_i.name and sBlock is stereotyped as «activity» and a_i is stereotyped
    as «uses» then
      usageList ← a_i.target
    end if
  end for
  t_i in pybpmn ← new «PyDescriptor» annotation where:
      resources ← usageList
end for
for all Source nodes s_i in in pybpmn do
  s_i in pybpmn ← new «PyDescriptor» annotation where:
      workloadParams ← rel_parameters[s_i]
end for
for all diverging gateways g_i in pybpmn do
  rprob ← pef_parameters[g_i]
  for all outgoing edges e_j of g_i do
    e_j.routingProb ← rprob[e_j]
  end for
end for
```

4.4.2. Generation of the eBPMN Executable Simulation

As underlined in Section 1, the executable simulation is based on the eBPMN framework. The relevant Java-based implementation can be obtained straightforwardly from the PyBPMN model through a `model-to-text` transformation, as illustrated in [48]. The code generation is carried out by the *DT Implementation Generator* component. Both the PyBPMN metamodel and the eBPMN framework are built on top of the Eclipse Modeling Framework, which provides a concrete implementation of the MDA standard.

4.4.3. Generation of Maintenance Actions

The DT implementation component is also responsible for the automatic generation of maintenance actions according to the rules specified herein. For each atomic resource, the following actions are defined:

The ultimate objective of the DT is to specify, schedule, and execute appropriate maintenance actions based on the expected failures of the resources composing the execution platform. Specifically, such actions take into account the nature of each resource (e.g., whether it is redundant or not) and are determined according to the following rules:

- **Preventive maintenance notification:** atomic and structured resources do not have an available backup device that can be activated in case of failure. In this case, a *notification* action is scheduled to notify the expected failure event in advance. The administrator is then responsible for planning and executing the required maintenance intervention, coherently with the resources RUL, ensuring minimal impact on system operations.
- **Automated reconfiguration:** redundant resources are associated with a pool of backup resources. In this case, three actions are generated:
  - a *notification* informing about the expected failure time. This notification is scheduled prior to the failure event;
  - an automated *replacement* action to activate an available backup resource before the failure occurs. This action is scheduled prior to the failure event;
  - a *notification* informing about the actual execution of the automatic process reconfiguration.

The actions list is initialized by the DT implementation and subsequently updated after the simulation outcomes analysis, as illustrated in Section 4.5. Specifically, the initialization process is provided in Algorithm 2.

Algorithm 2 Initialization of Maintenance Actions

Input: *exec_platform:* list of resources composing the execution platform
Output: *maintenance_actions* in XML format
  for each r_i in *exec_platform* **do**
    create a `<resource>` entry with:
      `id` $\leftarrow$ r_i.id
      `name` $\leftarrow$ r_i.name
      `semantics` $\leftarrow$ r_i.semantics
    maintenance_actions $\leftarrow$ `<resource>` entry
  end for

Finally, the eBPMN-based simulation implementation is given as input to the *eBPMN Execution Engine* and relevant results are sent to the *Results Analysis and Feedback subsystem*.

4.5. Results Analysis and Feedback

This subsystem is responsible for two activities. First, the simulation outcomes are analyzed to predict resource failures and complete the specification of preventive maintenance actions. Then, such actions are executed to update the state of the physical system, thereby preventing (or mitigating) the expected downtime. More specifically, the simulation results are analyzed by the *Simulation Outcomes Evaluation* component, which determines the RUL of each resource in the hardware platform (i.e., the time interval between the current instant and the next predicted failure). Based on these results, the maintenance action list is updated, and for each action, the `plannedExecTime` attribute is assigned. These steps are illustrated in Algorithm 3.

Algorithm 3 Update of Maintenance Actions with Simulation Results

Input: *exec_platform:* list of execution resources
Input: *maintenance_actions:* action list in XML format
Input: *simulation_results:* simulation results
Input: *lead_time*: user-provided advance time for timely actions scheduling
Output: updated *maintenance_actions*

```
for each r_i in exec_platform do
  if ∃ failure f of r_i in simulation_results then
    failure_time ← f.time
    retrieve act <actions> entry from maintenance_actions such that: act.id = r_i.id
    if r_i has an available backup r_i_bck then
      add an <alert> action to act with:
        plannedExecTime ← failure_time −2· lead_time
        message ← "Expected failure on " + failure_time
      add a <replace> action to act with:
        plannedExecTime ← failure_time − lead_time
        alternative ← r_i_bck
      add an <alert> action to act with:
        plannedExecTime ← failure_time − lead_time
        message ← "The resource " r_i.name " is being replaced with "
      r_i_bck.name
    else
      add an <alert> action to act with:
        plannedExecTime ← failure_time −2· lead_time
        message ← "Expected failure on " + failure_time
    end if
    update act in maintenance_actions
  end if
end for
```

The actual execution of these actions at the scheduled times is managed by the *Actions Execution Engine* component. In particular,

- for each *notification* action, an alert message is sent to the Admin console;
- for each *replace* action, if at least one backup resource is available (i.e., in an `inactive state`), the BP engine is reconfigured so that the failed resource is replaced by its backup. The update is also reported to the *DT Model Development and Update* component, which updates the PyBPMN model to maintain alignment with the DT implementation.

When the physical system is modified following the enacted maintenance actions, the Digital Twin is updated accordingly (see Figure 2, where the *Action* feedback is sent both to the PS and to the *DT Model Development and Update* component).

If *Actions Execution Engine* cannot execute any of the planned actions (for example, if no backup resource is available), an error message is sent to the administrator, who can then plan manual countermeasures.

The next section discusses an application example analyzed through a prototypical implementation of the proposed framework.

5. Example Application

To assess the feasibility and effectiveness of the proposed approach, an experimental study has been conducted on a manufacturing process.

Specifically, a prototypical implementation of the DT framework has been applied to a process flow modeled using the BPMN notation. The experimental evaluation presented in this section does not rely on a real production system but on an emulated execution

platform designed to reproduce the behavior of a manufacturing process. The BPMN model has been executed on an infrastructure in which physical resources have been emulated through software components that reproduce the failure-prone behavior of real manufacturing equipment. Within this setup, the physical system is represented by a Camunda-based BP engine that executes the process model and invokes web services emulating the behavior of manufacturing devices. The Digital Twin, instead, includes ProM Tools, a plugin-based open-source framework which implements various process mining algorithms [47], the eBPMN-based BP simulation framework, and all the remaining data-driven components described in the architecture of Figure 2. Finally, the integration of the DT with the BP execution engine is implemented by leveraging the native APIs provided by Camunda [49]. These APIs enable the application of configuration changes resulting from DT-generated corrective actions, such as resource-to-task reassignment or process reconfiguration.

To assess the impact of the proposed framework on BP operations, the results obtained in two different scenarios have been compared. In the first, the feedback provided by the data-driven DT has been ignored and the BP has been executed on a conventional platform without any backup resources available. In the second case, redundant resources have been introduced in the execution platform and the physical system has been integrated with the data-driven DT.

The next section illustrates, in detail, the manufacturing process, whereas the experimentation is illustrated in detail in Sections 4.3–5.4.

5.1. Manufacturing Process and Execution Platform Description

The assembly and delivery of products follow the activity flow described herein. For each product to be manufactured, the required components are first retrieved and then transferred to the production line. To this end, a feeding station, equipped with an automated pick-and-place tool and a conveyor belt supplies the components to the subsequent station, where the product is assembled by a robotic arm. Subsequently, an automated quality inspection is performed. If quality requirements are met, a conveyor belt transfers the product to the packing and shipping station; otherwise, a human inspection is required to identify and possibly correct assembly defects. The manufacturing process is assumed to operate from 6:00 to 24:00, Monday through Saturday, on 6 h shifts. Periods of inactivity are used for potential maintenance activities. This assumption does not represent a constraint of the proposed framework and does not limit the applicability of the approach under different conditions (e.g., 24/7 operational processes), for which only different operating hours need to be considered in the execution of the simulation.

The execution platform is composed of the following devices:

- **Feeding Station**: a structured resource composed of the following subsystems:
  - *Component Conveyor Belt*: an atomic resource responsible for transferring the required components from the stock shelves to the assembly station.
  - *Pick-and-Place Tool*: a redundant resource consisting of two robotic picking units designed to retrieve specific items and place them on the conveyor belt. The two redundant picking units operate in cold standby.
- **Assembly Station**: a redundant resource composed of two robotic arms specifically designed for component assembly. The two redundant arms operate in cold standby.
- **Camera**: an atomic resource that enables the inspection of the final product and the detection of potential assembly defects through image analysis.
- **Product Conveyor Belt**: an atomic resource responsible for transferring the assembled product either to the packing and shipping station or to the inspection station, where

a human operator performs a detailed check, evaluates the need for manual corrective actions, or arranges for product rejection.

It should be underlined that, although the example application discussed in this section focuses on a relatively simple process structure and on cold-standby redundancy, as discussed in Section 3.1, the PyBPMN/eBPMN framework natively supports BPMN models of arbitrary complexity. This includes processes involving multiple participants, message exchanges, parallel branches, and synchronization points, as well as execution platforms characterized by heterogeneous and mixed backup policies. Accordingly, the presented case study should be regarded as a representative and illustrative example, rather than as a limitation of the proposed approach.

The following subsections illustrate the application of the framework depicted in Figure 2.

5.2. Data Analysis

The system log and the state log are analyzed to extract the necessary information to create the reliability-aware simulation model.

It is worth explicitly clarifying that, in the example application discussed in this paper, the event log and the state log are synthetically generated. As discussed in our previous works [50,51], the synthetic logs used in this study are generated through an additional feature of the eBPMN framework. The synthetic log generator requires, as input parameters, the probability distribution functions (PDFs) and the corresponding parameters that characterize the stochastic behavior of the modeled process and resources. The generated logs are then provided as input to the data analysis component of the proposed framework. With respect to process-related data, the choice of the probability distributions adopted to generate the synthetic event logs is grounded in well-established assumptions from the process mining and simulation literature. In particular, as discussed in [52], a negative exponential distribution can be realistically assumed for modeling the inter-arrival time of process instances, while a normal distribution can be adopted to model task service times in simulation models discovered from event logs. Regarding reliability-related data, it is widely acknowledged that, in manufacturing and production processes, the Mean Time To Failure (MTTF) and the Mean Time To Repair (MTTR) can be effectively modeled using Weibull and lognormal distributions, respectively [53]. In the considered case study, the Weibull distribution is adopted for modeling the MTTF, assuming a shape parameter $\beta = 1$, which corresponds to an exponential distribution. This assumption reflects a constant failure rate, coherently, with the adoption of a preventive maintenance strategy. The MTTR is instead modeled through a lognormal distribution, which is commonly used to represent repair processes.

Finally, it is important to underline that predictive maintenance represents only one illustrative application scenario used to demonstrate the feasibility and effectiveness of the proposed framework for the development of data-driven DTs. In this context, identifying the probability distribution function that best fits the data tracked in event and state logs remains a relevant and challenging research issue. Nevertheless, within the perspective adopted in this study, the specific choice of probability distribution functions does not affect the general validity of the proposed framework.

Table 1 shows the performance analysis results for the execution platform resources (derived from the event log), while Table 2 presents the reliability analysis results (from the state log).

Table 1. Performance analysis of execution platform's resources.

Atomic Resource	Service Time		PDF
	Mean Value	Variance	
Picking Unit 1-2	2 min	1.4	Normal
Component Conveyor Belt	2 min	0.5	Normal
Robotic Arm 1-2	10 min	3.6	Normal
Camera	4 min	0.5	Normal
Product Conveyor Belt	7 min	4	Normal

Table 2. Reliability analysis of execution platform's resources.

Atomic Resource	MTTF (Mean)	MTTF (var)	Failure PDF	MTTR (Mean)	MTTR (var)	Repair PDF
Picking Unit 1-2	90 days	6 days	Weibull	90 min	2 min	Lognormal
Component Conveyor Belt	250 h	50 h	Weibull	8 h	1.5 h	Lognormal
Robotic Arm 1-2	200 h	10 h	Weibull	8 h	1.2 h	Lognormal
Camera	50 h	1.3 h	Weibull	120 min	5 min	Lognormal
Product Conveyor Belt	250 h	50 h	Weibull	8 h	1.5 h	Lognormal

Regarding the process workload, the arrival time of production requests follows an exponential distribution with a mean of 20 min. Finally, with respect to the routing of products after the quality control phase, the log analysis reveals that 40% of the components are sent to manual inspection, while the remaining items are directly routed to the packaging station.

5.3. Digital Twin Development

The automated specification of the PyBPMN model performed by the *DT Model Development and Update* component takes advantage of the information extracted from the log repositories and requires a SysML model that specifies the structure of the execution platform and the allocation of process activities to the needed resources.

Specifically, the execution platform specification consists of two distinct SysML diagrams: (i) a Block Definition Diagram that illustrates the resource allocation, and (ii) an Internal Block Diagram that specifies the structure of non-atomic resources.

These models are shown in Figure 8 and Figure 9, respectively.

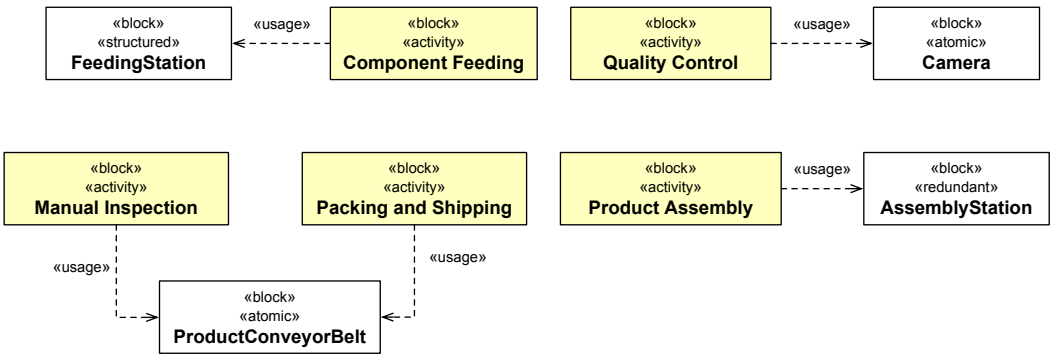


Figure 8. Block definition diagram specifying how the physical resources are used by the process activities.

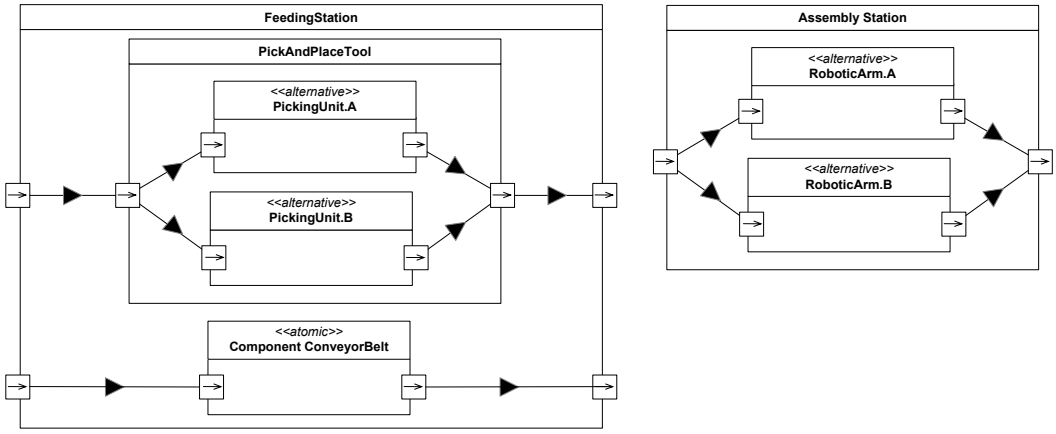


Figure 9. Internal block diagram showing the structure of the various complex resources composing the execution platform.

Figure 10 shows the PyBPMN model obtained.

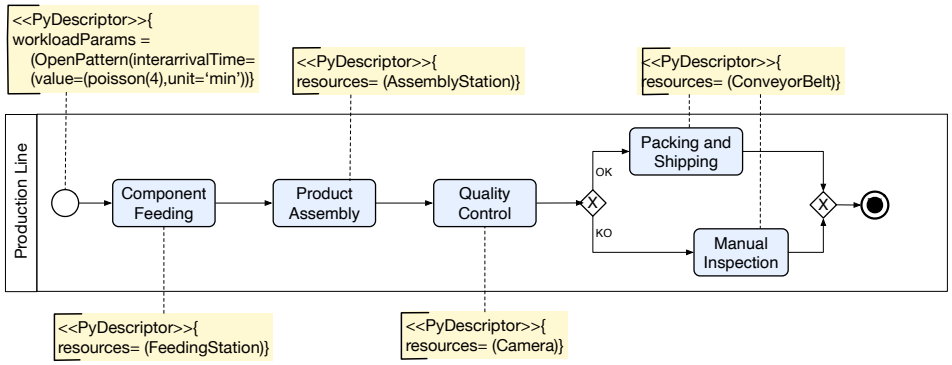


Figure 10. BPMN with performance and reliability PyBPMN annotations.

For the sake of conciseness, Figure 10 only includes the «PyDescriptor» annotation which provide the workload description and the resource allocation for the process tasks. The description of structured and redundant resources is provided by the use of «PySubsystem» and «PyBroker» annotations, respectively. Furthermore, the performance and reliability characterization of each atomic resource is given by the «Performer» annotation. As an example, the description of the feeding station is provided in Listing 2.

Listing 2. Annotations of the Feeding Station structured resource.

```
<<PySubsystem>>
{name=FeedingStation
components(PickAndPlaceTool, ComponentsConveyorBelt)}

<<PyBroker>>
{name=PickAndPlaceTool,
alternatives=(PickingUnit1,PickingUnit2),
standby=cold}

<<PyPerformer>>
{name=ComponentsConveyorBelt,
serviceTime=(value=(exp(2)), unit=min),
MTTF=(value=(normal(280), unit=250),
MTTR=(value=(normal(8), unit=hours)}
```

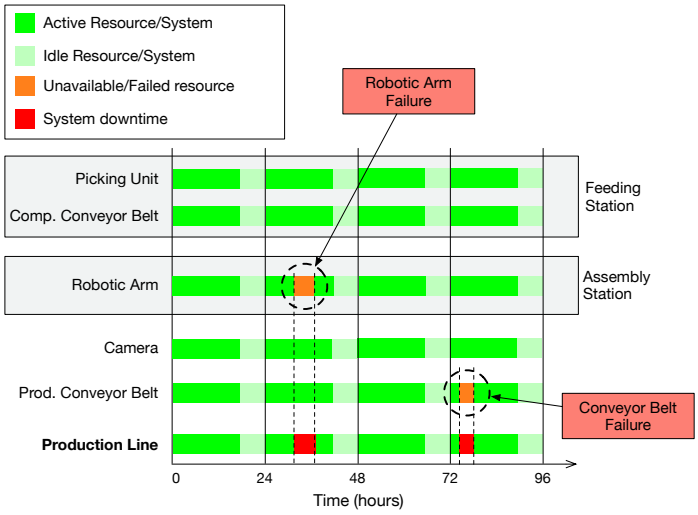
As illustrated in Section 4.4, together with the eBPMN code, the *DT Implementation Generation* component outputs an XML list of maintenance actions, which will subsequently

be completed with the execution times of the various activities after the simulation analysis carried out by the *Simulation Outcomes Evaluation* component.

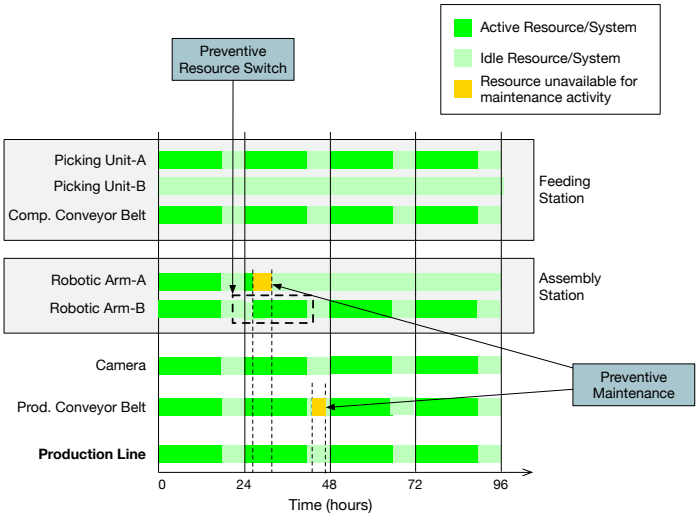
5.4. Results Evaluation and Actions on the Physical System

As discussed previously, the experimentation has considered and analyzed two different scenarios. To evaluate the effectiveness of the preventive maintenance actions provided by the DT, let us first consider the scenario where the execution platform does not include any backup resource, and predictions and preventive maintenance actions provided by the DT are initially ignored.

Figure 11a summarizes the actual state of the physical system during an observation period of 96 h. In such a time interval, two failures occur: the first one affects the Robotic Arm that equips the Assembly Station, while the second one is related to the Products Conveyor Belt. In this case, each resource failure directly affects the operability of the production process, which is subjected to downtime until the failed resource is repaired.



(**a**)



(**b**)

Figure 11. Manufacturing process operability with and without the DT. (**a**) When redundant resources are not available and any preventive maintenance actions are performed, resource failures lead to interruptions of the process operation. (**b**) Reliability predictions are used to schedule automated reconfiguration of the Assembly Station and on the Products Conveyor Belt to schedule a timely preventive maintenance.

In the second experiment, the same BP was executed over an execution platform which includes redundant devices. Preventive maintenance actions scheduled according to the DT predictions are also considered. Consistently with what has been observed in the first scenario, the process simulation predicts the occurrence of two failures: the assembly unit fails after 29.1 h (h) of operating time, corresponding to 35.1 h of elapsed time (i.e., including nighttime idle periods). The product conveyor belt fails after 49.5 h (corresponding to 61.5 h of elapsed time). Such predictions are used to generate the following corrective actions:

- Failure of the *Robotic Arm* (part of the Assembly Station):
  - a notification of the upcoming failure;
  - an automated reconfiguration of the execution platform, to replace RoboticArm-A with RoboticArm-B before the failure event, so as to ensure the continuity of the production process;
  - a second notification message, sent when the automated reconfiguration starts.
- Failure of the *Product Conveyor Belt*: a notification of the upcoming failure.

As no backup resource is available for the Conveyor Belt, the corrective action consists of a notification sent to the administrator, so that appropriate maintenance could be arranged to minimize the impact on the production line (e.g., during the night as in this case, or by planning downtime when lower productivity is expected).

To evaluate the impact of failures on system productivity, the manufacturing times obtained in the two scenarios are compared. Figure 12 shows the production times for 1000 items. In the *Conventional Process* case, noticeable peaks are observed, corresponding to failures that occurred during production.

Specifically, four failures were observed for the conventional scenario, resulting in a total production downtime of 15.51 h, corresponding to the time required to restore the affected resources. In contrast, in the DT-based scenario, the automatic reconfiguration of redundant resources and the scheduling of preventive maintenance actions during non-operational periods, as illustrated in Figure 11b, prevent the occurrence of production downtime. As a result, the production process remains continuously operational over the same time period.

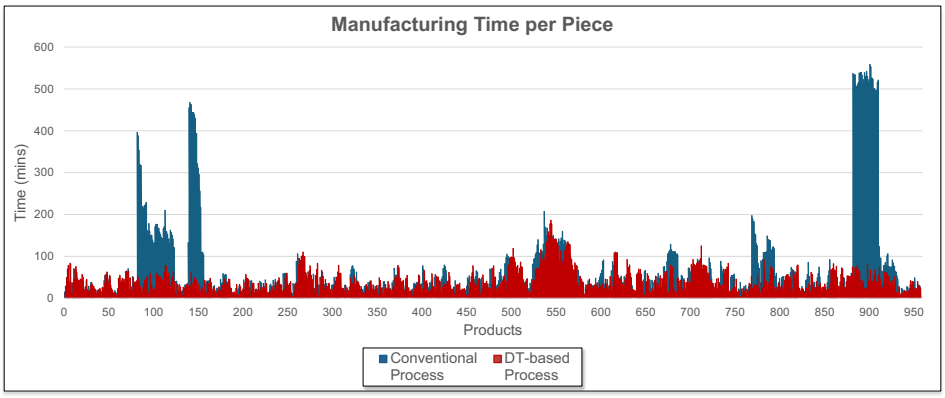


Figure 12. Manufacturing time per product (sample of 1000 items).

Over the considered observation period, the performance indicators reported in Table 3 have been computed. We observe an approximate 57% reduction in the average manufacturing time. This improvement is mainly enabled by the Digital Twin, which allows maintenance actions to be enacted in advance of predicted failures, thereby stabilizing the production flow and significantly reducing process variability, as also confirmed by the observed variance reduction. Moreover, the DT-based process achieves a throughput increase of about 19.2%. A sliding-window analysis performed over 100 consecutive items

reveals that the throughput improvement is not uniform over time and can reach peak values close to 30% during specific operating periods. This result highlights that the benefits of failure mitigation are particularly significant in phases where failures induce congestion and propagation effects along the production line.

Table 3. Manufacturing process performance indicators.

Scenario	Manufacturing Time (Mean)	Manufacturing Time (Variance)	Throughput (Items/h)
Conventional Process	69.52 min	10,293.60	2.51
DT-based Process	39.86 min	744.43	3.01

Furthermore, a detailed analysis has been performed on the production of a sample of 100 items, from number 50 to number 150. Figure 13 compares the time required to complete the production of this sample in the two scenarios. The results highlight the impact of the DT on overall process productivity.

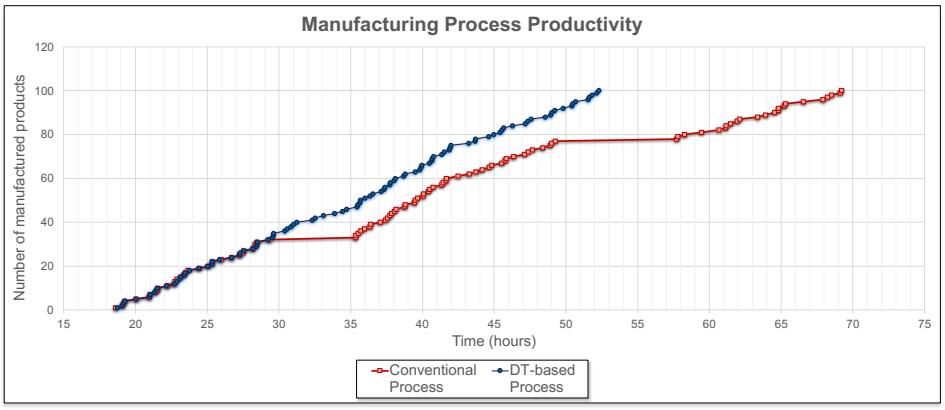


Figure 13. Process productivity in terms of the cumulative number of manufactured products (sample of 100 items).

The experimental results summarized in this section are based on measurements collected on the physical system (PS) under two different scenarios, namely with and without the presence of the DT, with the specific aim of assessing the actual and effective applicability of the proposed data-driven framework. As illustrated in Section 5.2, the simulation model at the core of the DT is derived from synthetically generated logs under realistic modeling assumptions. Nonetheless, since the data analysis component enables the discovery of both the simulation model and its parameters from real execution data, the modeling assumptions introduced for the case study do not affect the general validity of the proposed approach.

5.5. Discussion

This section discusses the lessons learned from the analysis of the experimental results and outlines advantages and limitations of the proposed method. The application of a prototypical framework to the proposed case study made it possible to verify the actual feasibility of the method and its effectiveness. The availability of a DT constantly aligned with the operational configuration of the PT enabled the prediction of failures in the execution platform and the execution of appropriate countermeasures: automatic reconfiguration (where possible) and the timely scheduling of preventive maintenance actions when a backup resource was not available. These actions helped prevent resource downtime, thereby ensuring higher productivity. The use of an innovative approach based on process mining techniques and model-driven technologies enabled the automatic generation of

an executable simulation, reducing the effort and expertise required for the development and subsequent implementation of a reliability-aware simulation model. On the other hand, experimentation with the prototypical framework also revealed the limitations of the proposed approach:

- **DT latency and real-time applicability:** in the considered evaluation setting, reconfiguration and maintenance actions are scheduled during the system's non-operational or low-load periods. The reconfiguration operations executed through the invocation of BP engine APIs have been carried out without significantly affecting productivity, and the overhead introduced by DT analysis and system reconfiguration is therefore negligible. The Camunda execution engine enables workflows to react to events with minimal latency, since workflow deployment and configuration updates can be performed through API invocations that streamline model updates without significant overhead with respect to typical business process execution time scales. Consequently, in the examined scenario, network communication delays between the PAIS and the DT, as well as the time required to apply reconfiguration decisions, can be considered negligible. In real-time or safety-critical domains, however, the end-to-end latency of the DT feedback loop becomes a key factor and mainly depends on the time required to reconfigure the involved resources. Such latency is negligible for software-based resources, while it may require explicit evaluation for physical appliances that involve manual intervention or human-in-the-loop actions. Future work will therefore include a quantitative assessment of these aspects.
- **Synthetic logs and validation on industrial datasets:** the evaluation presented in this paper relies on synthetic event and state logs. Furthermore, an emulated execution platform has been used in the addressed case. While this choice enables controlled experimentation and repeatability, it does not fully capture the complexity and the size of real industrial data. Future work includes applying the framework to a real case with more extensive datasets, also to assess the scalability of the proposed approach.
- **Automation and Availability of Toolchains**: the prototypical implementation of the framework has been based on the use of different tools developed and executed in heterogeneous operational environments (ProM tools, EMF-based model transformations, BP engines, etc.). The complexity of the framework would highly benefit from the development of an integrated system in which the generation and deployment of the DT, its execution, the implementation of appropriate countermeasures, and the alignment of the simulation model with the state changes of the physical system can all be handled in a unified manner. In this respect, future work includes the implementation of an integrated Digital Twin management tool capable of supporting the full DT lifecycle, from its development to the management of relevant adaptive feedback mechanisms. Such an approach is required to improve scalability and maintainability and ensure an effective synchronization between the simulation model and the evolving state of the physical system, thereby improving the effectiveness of decision-support activities.

6. Conclusions

This paper introduced a data-driven framework for the development of Digital Twins of BPs. The approach relies on the joint use of SysML and extended BPMN models and exploits process mining techniques and model-driven engineering principles to generate a realibility-aware DT. The framework provides a structured method to integrate reliability analysis into process models, allowing for the anticipation of resource failures and the scheduling of preventive maintenance actions.

The case study highlighted the effectiveness of the approach in a manufacturing scenario, showing how predictive maintenance based on DT simulations can significantly reduce downtime and improve productivity.

Work is in progress to complete the implementation of a full-fledged tool and carry out its validation in various application scenarios.

Author Contributions: Conceptualization, P.B., M.F. and A.D.; methodology, P.B. and A.D.; software, P.B.; validation, P.B. and M.F.; investigation, P.B.; data curation, M.F.; writing—original draft preparation, P.B. and M.F.; writing—review and editing, P.B., M.F. and A.D.; visualization, P.B., M.F. and A.D.; supervision, A.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Dataset available on request from the authors.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
API BDD	Block Definition Diagram (SysML)
BP	Business Process
BPM	Business Process Management
BPMN	Business Process Model and Notation
CPS	Cyber-Physical System
DT	Digital Twin
eBPMN	Executable BPMN
IBD	Internal Block Diagram (SysML)
IoT	Internet of Things
PyBPMN	Performability-enabled BPMN
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MDE	Model-Driven Engineering
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
PAIS	Process-aware Information System
PdA	Predictive Maintenance
PDF	Probability Distribution Function
PM	Process Mining
RUL	Remaining Useful Life
SysML	Systems Modeling Language
WS	Web Services

References

1. Van der Aalst, W.M. Business process management: A comprehensive survey. *Int. Sch. Res. Not.* **2013**, *2013*, 507984. [CrossRef]
2. Bocciarelli, P.; D'Ambrogio, A.; Giglio, A.; Paglia, E. Modeling resources to simulate business process reliability. *ACM Trans. Model. Comput. Simul.* **2020**, *30*, 14. [CrossRef]
3. van der Aalst, W.M.P.; ter Hofstede, A.H.M.; Weske, M. Business process management: A survey. In *Proceedings of the 2003 International Conference on Business Process Management*; BPM'03; Springer: Berlin/Heidelberg, Germany, 2003; pp. 1–12.
4. Ozay, D.; Jahanbakht, M.; Shoomal, A.; Wang, S. Artificial Intelligence (AI)-based Customer Relationship Management (CRM): A comprehensive bibliometric and systematic literature review with outlook on future research. *Enterp. Inf. Syst.* **2024**, *18*, 2351869. [CrossRef]
5. Tiftik, M.N.; Erdogan, T.G.; Tarhan, A.K. A Framework for multi-perspective Process Mining into a BPMN Process Model. *Math. Biosci. Eng.* **2022**, *19*, 11800–11820. [CrossRef]

6. Kumbhar, M.; Ng, A.H.; Bandaru, S. A digital twin based framework for detection, diagnosis, and improvement of throughput bottlenecks. *J. Manuf. Syst.* **2023**, *66*, 92–106. [CrossRef]
7. van der Aalst, W. *Process Mining: Data Science in Action*, 2nd ed.; Springer Publishing Company: Berlin/Heidelberg, Germany, 2016.
8. van Der Aalst, W. Process mining: Overview and opportunities. *ACM Trans. Manag. Inf. Syst.* **2012**, *3*, 7. [CrossRef]
9. Martin, N.; Depaire, B.; Caris, A. The use of process mining in a business process simulation context: Overview and challenges. In *Proceedings of the 2014 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*; IEEE: Piscataway, NJ, USA, 2014; pp. 381–388.
10. Friederich, J.; Lazarova-Molnar, S. Data-driven reliability modeling of smart manufacturing systems using process mining. In *Proceedings of the 2022 Winter Simulation Conference*; Feng, B., Pedrielli, G., Peng, Y., Shashaani, S., Song, E., Corlu, C., Lee, L., Chew, E., Roeder, T., Lendermann, P., Eds.; Institute of Electrical and Electronics Engineers, Inc.: Piscataway, NJ, USA, 2022; pp. 2534–2545.
11. Singh, M.; Fuenmayor, E.; Hinchy, E.P.; Qiao, Y.; Murray, N.; Devine, D. Digital twin: Origin to future. *Appl. Syst. Innov.* **2021**, *4*, 36. [CrossRef]
12. van der Aalst, W.M. Concurrency and objects matter! Disentangling the fabric of real operational processes to create digital twins. In *Proceedings of the International Colloquium on Theoretical Aspects of Computing*; Springer: Cham, Germany, 2021; pp. 3–17.
13. Rossini, R.; Conzon, D.; Prato, G.; Pastrone, C.; dos Reis, J.P.C.; Gonçalves, G. REPLICA: A Solution for Next Generation IoT and Digital Twin Based Fault Diagnosis and Predictive Maintenance. *SAM IoT* **2020**, *2739*, 55–62.
14. Eramo, R.; Bordeleau, F.; Combemale, B.; Brand, M.v.d.; Wimmer, M.; Wortmann, A. Conceptualizing Digital Twins. *IEEE Softw.* **2022**, *39*, 39–46. [CrossRef]
15. Michael, J.; Cleophas, L.; Zschaler, S.; Clark, T.; Combemale, B.; Godfrey, T.; Khelladi, D.E.; Kulkarni, V.; Lehner, D.; Rumpe, B.; et al. Model-Driven Engineering for Digital Twins: Opportunities and Challenges. *Syst. Eng.* **2025**, *28*, 659–670. [CrossRef]
16. Ojbect Management Group. MDA Guide, Revision 2.0 (ormsc/14-06-01). 2003. Available online: https://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf (accessed on 11 February 2026).
17. Van Dinter, R.; Tekinerdogan, B.; Catal, C. Predictive maintenance using digital twins: A systematic literature review. *Inf. Softw. Technol.* **2022**, *151*, 107008. [CrossRef]
18. Aivaliotis, P.; Georgoulias, K.; Chryssolouris, G. The use of Digital Twin for predictive maintenance in manufacturing. *Int. J. Comput. Integr. Manuf.* **2019**, *32*, 1067–1080. [CrossRef]
19. Rasheed, A.; San, O.; Kvamsdal, T. Digital twin: Values, challenges and enablers from a modeling perspective. *IEEE Access* **2020**, *8*, 21980–22012. [CrossRef]
20. Friederich, J.; Francis, D.P.; Lazarova-Molnar, S.; Mohamed, N. A framework for data-driven digital twins of smart manufacturing systems. *Comput. Ind.* **2022**, *136*, 103586. [CrossRef]
21. Chen, C.; Fu, H.; Zheng, Y.; Tao, F.; Liu, Y. The advance of digital twin for predictive maintenance: The role and function of machine learning. *J. Manuf. Syst.* **2023**, *71*, 581–594. [CrossRef]
22. Lê, L.S. Pr-dash: A dashboard-constructing tool for twinning business processes. *SN Comput. Sci.* **2024**, *5*, 550. [CrossRef]
23. Giussani, S.; Perez-Palacin, D.; Caporuscio, M.; Edrisi, F. Business Process Lifecycle Enhancement via Digital Twin and Model-Driven Engineering. In *Proceedings of the 2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C)*; IEEE: Piscataway, NJ, USA, 2025; pp. 300–309.
24. Qian, C.; Liu, X.; Ripley, C.; Qian, M.; Liang, F.; Yu, W. Digital Twin—Cyber Replica of Physical Things: Architecture, Applications and Future Research Directions. *Future Internet* **2022**, *14*, 64. [CrossRef]
25. Brockhoff, T.; Heithoff, M.; Koren, I.; Michael, J.; Pfeiffer, J.; Rumpe, B.; Uysal, M.S.; Van Der Aalst, W.M.; Wortmann, A. Process prediction with digital twins. In *Proceedings of the 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*; IEEE: Piscataway, NJ, USA, 2021; pp. 182–187.
26. Lehner, D.; Zhang, J.; Pfeiffer, J.; Sint, S.; Splettstößer, A.K.; Wimmer, M.; Wortmann, A. Model-driven engineering for digital twins: A systematic mapping study. *Softw. Syst. Model.* **2025**, *24*, 1339–1377. [CrossRef]
27. Macak, M.; Daubner, L.; Sani, M.F.; Buhnova, B. Process mining usage in cybersecurity and software reliability analysis: A systematic literature review. *Array* **2022**, *13*, 100120. [CrossRef]
28. Friederich, J.; Lazarova-Molnar, S. Process mining for reliability modeling of manufacturing systems with limited data availability. In *Proceedings of the 2021 8th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*; IEEE: Piscataway, NJ, USA, 2021; pp. 1–7.
29. Pereira, A.; Thomas, C. Challenges of machine learning applied to safety-critical cyber-physical systems. *Mach. Learn. Knowl. Extr.* **2020**, *2*, 579–602. [CrossRef]
30. Sang, G.M.; Xu, L.; De Vrieze, P.; Bai, Y.; Pan, F. Predictive maintenance in industry 4.0. In *Proceedings of the 10th International Conference on Information Systems and Technologies*; ACM: New York, NY, USA, 2020; pp. 1–11.

31. Abd Wahab, N.H.; Hasikin, K.; Lai, K.W.; Xia, K.; Bei, L.; Huang, K.; Wu, X. Systematic review of predictive maintenance and digital twin technologies challenges, opportunities, and best practices. *PeerJ Comput. Sci.* **2024**, *10*, e1943. [CrossRef] [PubMed]
32. Kaur, R.; Kumar, R.; Aggarwal, H. Systematic Review of Artificial Intelligence, Machine Learning, and Deep Learning in Machining Operations: Advancements, Challenges, and Future Directions. *Arch. Comput. Methods Eng.* **2025**, *32*, 4983–5036. [CrossRef]
33. D'Urso, D.; Chiacchio, F.; Cavalieri, S.; Gambadoro, S.; Khodayee, S.M. Predictive maintenance of standalone steel industrial components powered by a dynamic reliability digital twin model with artificial intelligence. *Reliab. Eng. Syst. Saf.* **2024**, *243*, 109859. [CrossRef]
34. Ren, Z.; Wan, J.; Deng, P. Machine-learning-driven digital twin for lifecycle management of complex equipment. *IEEE Trans. Emerg. Top. Comput.* **2022**, *10*, 9–22. [CrossRef]
35. Bocciarelli, P.; D'Ambrogio, A. Simulation-based Predictive Process Mining with eBPMN: Methods, Challenges and Opportunities. In *Proceedings of the 2024 Annual Modeling and Simulation Conference (ANNSIM)*; IEEE: Piscataway, NJ, USA, 2024; pp. 1–14.
36. Bocciarelli, P.; D'Ambrogio, A. A Low-code Approach for Simulation-based Analysis of Process Collaborations. In *Proceedings of the 2023 Winter Simulation Conference*; Corlu, C.G., Hunter, S.R., Lam, H., Onggo, B.S., Shortle, J., Biller, B., Eds.; Institute of Electrical and Electronics Engineers, Inc.: Piscataway, NJ, USA, 2023; pp. 2530–2541.
37. D'Ambrogio, A.; Paglia, E.; Bocciarelli, P.; Giglio, A. Towards performance-oriented perfective evolution of BPMN models. In Proceedings of the 6th International Workshop on Model-Driven Approaches for Simulation Engineering, Pasadena, CA, USA, 3–6 April 2016.
38. Bocciarelli, P.; D'Ambrogio, A.; Paglia, E. A Language for Enabling Model-Driven Analysis of Business Processes. In Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development, Lisbon, Portugal, 7–9 January 2014; MODELSWARD'14.
39. Ojbect Management Group. Business Process Model And Notation (BPMN) Version 2.0.2. 2014. Available online: https://www.omg.org/spec/BPMN/ (accessed on 11 February 2026).
40. Kritzinger, W.; Karner, M.; Traar, G.; Henjes, J.; Sihn, W. Digital Twin in manufacturing: A categorical literature review and classification. *Ifac-PapersOnline* **2018**, *51*, 1016–1022. [CrossRef]
41. Abdullahi, I.; Longo, S.; Samie, M. Towards a distributed digital twin framework for predictive maintenance in industrial internet of things (IIoT). *Sensors* **2024**, *24*, 2663. [CrossRef]
42. Alam, M.; Islam, M.R.; Shil, S.K. AI-Based predictive maintenance for US manufacturing: Reducing downtime and increasing productivity. *Int. J. Adv. Eng. Technol. Innov.* **2023**, *1*, 541–567.
43. Kamat, P.; Sugandhi, R. Anomaly detection for predictive maintenance in industry 4.0-A survey. In *Proceedings of the E3S Web of Conferences*; EDP Sciences: Les Ulis, France, 2020; Volume 170, p. 02007.
44. Zonta, T.; Da Costa, C.A.; da Rosa Righi, R.; de Lima, M.J.; Da Trindade, E.S.; Li, G.P. Predictive maintenance in the Industry 4.0: A systematic literature review. *Comput. Ind. Eng.* **2020**, *150*, 106889. [CrossRef]
45. Zhang, W.; Yang, D.; Wang, H. Data-driven methods for predictive maintenance of industrial equipment: A survey. *IEEE Syst. J.* **2019**, *13*, 2213–2227. [CrossRef]
46. XES Working Group. IEEE Standard for eXtensible Event Stream (xes) for Achieving Interoperability in Event Logs and Event Streams. *IEEE Std* **2023**, *1849*, 1–50.
47. ProM Tools. ProM Process Mining Toolkit. 2022. Available online: https://www.promtools.org (accessed on 11 February 2026).
48. Bocciarelli, P.; D'Ambrogio, A.; Giglio, A.; Paglia, E. BPMN-Based Business Process Modeling and Simulation. In *Proceedings of the 2019 Winter Simulation Conference*; Mustafee, N., Bae, K.H.G., Lazarova-Molnar, S., Rabe, M., Szabo, C., Haas, P., Son, Y.J., Eds.; Institute of Electrical and Electronics Engineers, Inc.: Piscataway, NJ, USA, 2019; pp. 1439–1453. [CrossRef]
49. Camunda Inc. Camunda Version 8. 2025. Available online: https://camunda.com/ (accessed on 11 February 2026).
50. Bocciarelli, P.; D'Ambrogio, A. Data-driven Simulation-based Analysis of Collaborative Business Processes in Distributed Environments. In *Proceedings of the 2025 Annual Modeling and Simulation Conference (ANNSIM)*; IEEE: Piscataway, NJ, USA, 2025.
51. Bocciarelli, P.; D'Ambrogio, A. An XES Extension for the Distributed Simulation of Process Collaborations. In *Proceedings of the 2024 IEEE International Symposium on Systems Engineering (ISSE)*; IEEE: Piscataway, NJ, USA, 2024; pp. 1–8.
52. Rozinat, A.; Mans, R.S.; Song, M.; van der Aalst, W.M. Discovering simulation models. *Inf. Syst.* **2009**, *34*, 305–327. [CrossRef]
53. Elmahdy, E.E. Modelling Reliability Data with Finite Weibull or Lognormal Mixture Distributions. *Appl. Math. Inf. Sci.* **2017**, *11*, 1081–1089. [CrossRef]