

An Object-centric Approach to Handling Concurrency in IoT-aware Processes

Florian Gallik¹[0000–0003–0861–275X], Yusuf Kirikkayis¹[0000–0001–6536–0785],
Ronny Seiger²[0000–0003–1675–2592], and Manfred Reichert¹[0000–0003–2536–4153]

¹ Institute of Databases and Information Systems, Ulm University, Ulm,
Germany{florian-1.gallik, yusuf.kirikkayis, manfred.reichert}@uni-ulm.de

² Institute of Computer Science, University of St.Gallen, St.Gallen, Switzerland
ronny.seiger@unisg.ch

Abstract. The increasing adoption of IoT in the context of Business Process Management (BPM) makes it necessary to efficiently coordinate concurrent processes and activities that involve physical resources. Traditional approaches to handling concurrency in BPM systems are not suitable for automating IoT-aware processes due to novel challenges raised by the IoT. We propose to handle concurrency in IoT based on object-centric processes implemented in the PHILharmonicFlows framework. The framework facilitates the data-driven modeling and coordination of object lifecycles and interactions, which are suitable to address concurrency in IoT-aware processes. The approach is demonstrated in a scenario from smart manufacturing. The results show that PHILharmonicFlows offers a flexible and comprehensible solution for coordinating concurrent activities in IoT settings with constrained physical resources.

Keywords: Internet of Things · Business Process Management · Concurrency · Object-centric Processes · Cyber-physical Systems

1 Introduction

The Internet of Things (IoT) is increasingly adopted in the context of Business Process Management (BPM) [10]. Data from IoT sensors can be used complementary to traditional event log data to improve process analysis and decision making [16,11]. In turn, BPM technologies experience a growing application in IoT use cases, e.g., for modeling and execution of *IoT-aware processes* [12]. BPM systems (BPMS) provide sophisticated means for modeling and automating processes involving IoT devices, digital resources, and humans. However, using BPMS for coordinating processes that involve the active control of IoT devices raises new challenges. The physical resources (actuators) are often constrained and unable to execute multiple activities concurrently [20,7]. This requires additional means for coordinating access to these resources [26]. Traditional techniques (e.g., worklists) are not efficient as they have not been designed for full automation, which may be required by the IoT use case (e.g., in smart manufacturing). In the context of IoT, related approaches either do not address

the aspect of handling concurrency in IoT-aware processes [12], or they rely on complex hard-wired logic implemented by an additional coordination component as part of a control system for the IoT devices [21]. Without proper coordination of concurrent activities, the execution of IoT-aware processes in the physical world might be interrupted and lead to unforeseen behavior and exceptions.

In this work, we propose an approach to handling concurrency in IoT-aware processes based on the object-centric process management framework *PHILharmonicFlows* [13]. This framework provides components for the modeling, execution, and coordination of data-driven object lifecycles. In particular, *lifecycle processes* can be leveraged for handling concurrency in combination with business processes. We use PHILharmonicFlows to model both the lifecycles of IoT devices and coordination processes enabling the proper management of concurrent process/activity executions while ensuring certain coordination constraints. A scenario from the domain of smart manufacturing is used as running example.

The paper is structured as follows: Section 2 introduces the research problem and research question. Section 3 presents the approach for handling concurrency using PHILharmonicFlows. Section 4 discusses our work in comparison to related work. Section 5 summarizes the paper and gives an outlook.

2 Problem Statement and Research Question

In traditional business processes not involving IoT, **digital** resources such as databases and web services can be accessed repeatedly and concurrently as the underlying software infrastructure (e.g., web servers or database management systems) provides scalability and coordination [9]. Usually, the coordination of **human** tasks relies on a worklist-based approach, i.e., process participants can view and retrieve open tasks from a list. This type of coordination is not suitable in an IoT context as the efficient automation is one of the main goals of IoT-aware processes, without involving human actors. In IoT-aware processes for which certain **physical** resources (actuators) exist only once and which cannot always be scaled up (e.g., a robot or an oven), multiple process activities may not be able to use the respective resources concurrently [20,7]. Moreover, the execution of activities that involve physical resources might be long-running and prevent the resources from being used by other activities. In traditional manufacturing setups, the handling of concurrent requests to the same machine is part of the Manufacturing Execution Systems (MES) which, for example use queues for each machine [25].

In this work we assume the existence of BPMS used for orchestrating IoT-aware processes. Related approaches often rely on (1) the capabilities of BPMS to handle parallel activity executions [19]; (2) the services/control applications for the IoT devices to handle concurrent access [14], which is usually incomprehensible for the user; or (3) additional coordination components hard-coded with a programming language, which is also incomprehensible and inflexible [18,21].

Smart Manufacturing Example: Fig. 1 illustrates a simple scenario of a smart factory [12]. Two processes include the same type of activity executed by the

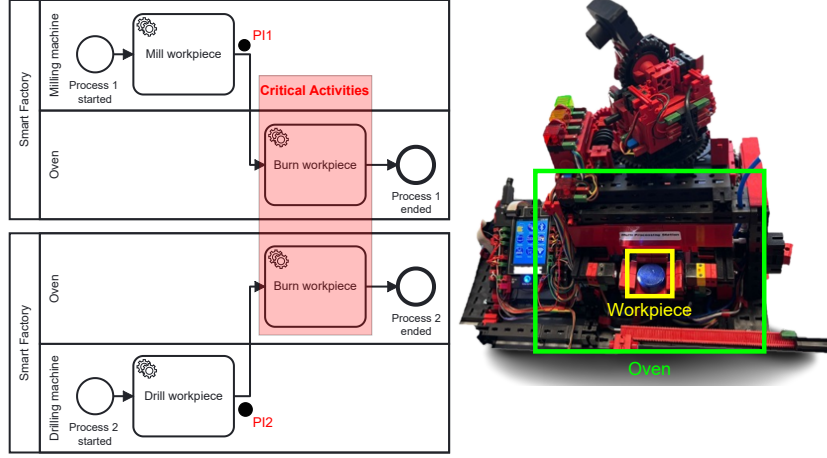


Fig. 1. Two IoT-aware process instances (in BPMN 2.0) accessing the same resource

same resource: in *Process 1*, the workpiece is milled first and then burned in the Oven; in *Process 2*, the workpiece is drilled first and then burned in the Oven. Critical here are the *Burn workpiece* activities in both processes as they require exclusive access the same physical resource *Oven* to execute the same activity. Similar problems can be found when executing the same activities concurrently in one process instance denoted as *batch activities* [18].

As shown, the use of BPMN 2.0 for modeling both processes does not make the problem of managing concurrency explicit. The depicted BPMN 2.0 models only represent a static view on the processes without any restrictions regarding their runtime behavior and involved resources [27]. As soon as two or more process instances of Process 1 and/or Process 2 concurrently reach the point of executing the *Burn workpiece* activity (*Multiple Instance Patterns* [2]), the runtime behavior cannot be reliably determined. Moreover, executing the activity *Burn workpiece* might take several hours, which blocks the resource for other process instances as only one workpiece can be burned at a time. *Petri nets* have been used for many years to make concurrency in processes explicit and analyzable [28]. However, Petri nets are mostly used to model and analyze the *control flow* of processes, which means that the executions of all process/activity instances have to be analyzed at runtime to identify potential conflicts [1]. Petri nets may be used complementary to processes to model the behavior of specific resources (e.g., production machines) when concurrency arises [28]. Nevertheless, the specification of constraints and dependencies with other resources, processes and materials as required by a state-of-the-art process management system fostering flexible IoT-aware processes is not well supported [10,3].

In summary, the coordination of concurrent access to physical resources that have limited parallel processing capabilities poses new challenges emerging with

the adoption of BPM in IoT. This paper investigates the following research question: **How to coordinate concurrently executed activities in IoT-aware processes with constrained physical resources?** We aim at developing a generic solution that can be used to flexibly and comprehensively coordinate concurrent process activities in IoT.

3 Handling Concurrency in IoT-aware Processes

3.1 Object-centric Processes

We investigate how concepts related to *Object-centric Processes* can be applied to handle concurrency in IoT-aware processes. Object-centric processes focus on the interactions and relations among objects [4]. These processes make the coordination of activities and interactions among objects explicit, which makes them suitable candidates for IoT settings where physical resources are constrained and need to be coordinated. *PHILharmonicFlows* is a process management framework focusing on object-centric processes [3]. It has proven to be well suited for modeling relations, interactions, and constraints among digital objects, data, and processes, with the main goal of achieving a high degree of flexibility [3,13]. *PHILharmonicFlows* leverages resource awareness and enables granular control and dynamic adaptation through a data-driven approach [3].

3.2 Relational Process Type Structure

In *PhilharmonicFlows*, the *Burn workpiece* process activity (cf. Fig. 1) consists of multiple *Objects* that need to be coordinated:

- Production: a representation of the burning process of workpieces.
- Workpiece: a representation of the physical item to be processed.
- Oven: a limited physical resource responsible for burning the workpieces.

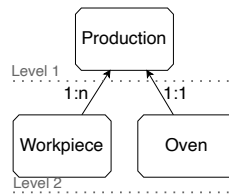


Fig. 2. Relational Process Type structure for the running example

A *Relational Process Type* structure is used to capture the involved objects and their interactions and relationships [22]. This structure serves as a foundation for coordinating the objects listed above (cf. Fig. 2). A Production object

instance has a relation to exactly one Oven object instance and at least one, but potentially multiple Workpiece object instances (indicated by the min:max cardinalities). The directed relations between objects create a hierarchy within the relational process type structure. As shown in Fig. 2, Production is a *higher level* object (Level 1), while Oven and Workpieces are *lower level* objects (Level 2).

3.3 Object Lifecycles

PHILharmonicFlows incorporates the concept of objects into its data-driven and data-centric process management. For each business object involved in a real-world process, a corresponding object exists in the system. In our running example, the Production, Oven, and Workpiece objects have distinct lifecycles.

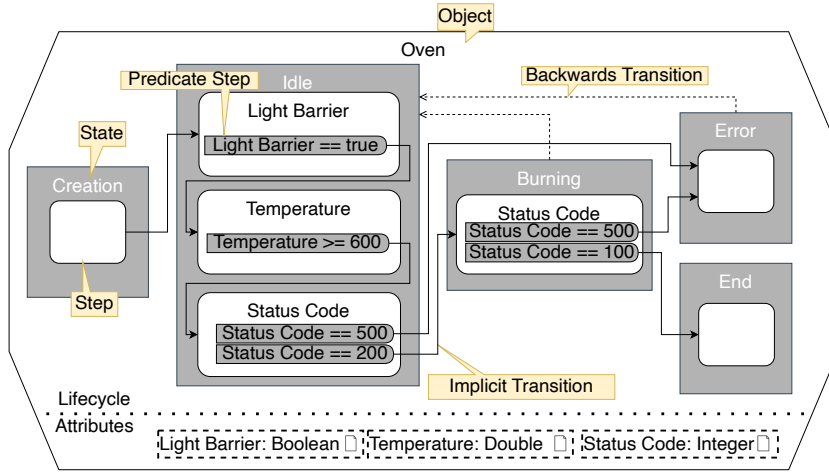


Fig. 3. Object Lifecycle of the Oven object (with annotations)

Conceptual Basis and Oven Object Fig. 3 shows the lifecycle of the Oven object which represents the machine used in the production process. An object encompasses data in the form of attributes and a state-based process model that details the object's lifecycle. This model shows the stages and transitions that the Oven object undergoes. The attributes capture relevant data points, which can be used for internal state comparisons. When an object is instantiated, it automatically enters its initial state (here: *Creation*). As this state only involves an empty step, a direct transition (*Implicit Transition*) to the *Idle* state occurs (represented by a solid arrow). An *Explicit Transition* (represented by dashed arrow) must be triggered explicitly by a user action or an external system [3].

Predicate Steps and Idle State In the *Idle* state, attributes are set by an automated retrieval of data from the physical machine following the specified sequence of steps (cf. Fig. 3). Once all values are set and the associated conditions, represented by the *Predicate Steps*, are met, a transition to the next state occurs. In the example, a transition to *Burning* takes place if: 1) the Light Barrier is interrupted (indicating the presence of a workpiece in the correct position); 2) the oven has reached the required Temperature; and 3) the oven reports a Status Code of 200 (indicating it is ready for the burning activity). The latter plays a crucial role as it represents the process activity that requires coordination. Here, we see a limitation of PHILharmonicFlows, which was originally designed for working with digital processes [13]. It is capable of retrieving and evaluating data only once and in sequence; not supporting IoT settings that rely on data streams that need to be continuously evaluated to derive higher level states [11].

Backwards Transitions In the *Error* state, there are two possible scenarios: 1) the error can be automatically resolved or 2) manual error handling is required. Once the error has been resolved, triggering the *Backwards Transition* (dashed backwards arrow) allows the machine to return to the *Idle* state, indicating that it is ready to proceed with normal operations. When the burning process is completed, the oven has two possible transitions, i.e., it may either move to the *End* state or return to the *Idle* state which is triggered by the machine itself. The transition to the end state indicates that the burning task was completed successfully and the lifecycle of the Oven terminated. Alternatively, the oven may transition back to the *Idle* state, signaling that it is ready for further processing.

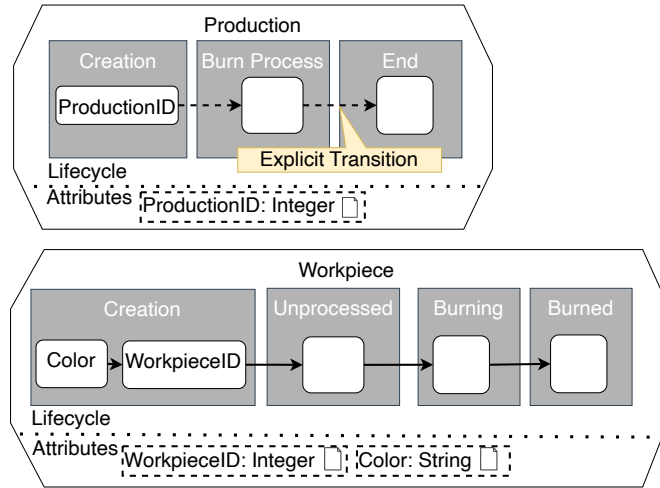


Fig. 4. Object Lifecycles for the Production and Workpiece objects

Production Object Fig. 4 shows the object lifecycles of the Production and Workpiece objects. The Production object represents the high-level production process that involves the Oven and Workpiece objects. For simplicity, in our example, the Production object only has one attribute, the ProductionID. When a Production object is instantiated, it enters the *Creation* state setting the *ProductionID* for that particular production instance. An transition to the *Burn Process* state takes place, which represents the crucial burn activity that has to be coordinated. This explicit transition is triggered either by an external system or by a user action, depending on the level of automation. Upon completion of the Burn process, the Production object transitions similarly to the *End* state indicating that the production process has reached its end.

Workpiece Object The workpiece object represents the item being processed in the production process. As the workpiece progresses through the production process, it primarily undergoes state changes and transitions based on the actions or states of other objects, such as *Oven* or *Production*. The Workpiece object in this example possesses two attributes: color and ID. The object initiates in the *Creation* state, where the attributes are populated with data. Through an implicit transition, the Workpiece object transitions to the *Unprocessed* state, followed by the *Burning* state, and it ultimately reaches the end state *Burned*.

3.4 Coordination Process for Handling Concurrency

Coordination Processes offer a flexible way for managing interdependent lifecycle processes by expressing semantic constraints through relationships, which are enforced at runtime [13]. This is achieved through a high-level graphical notation. Fig. 5 shows the coordination process for handling concurrent access to the Oven.

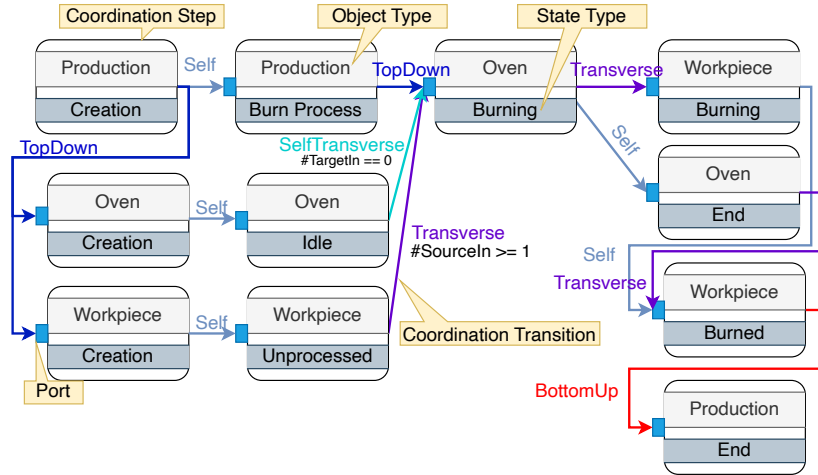
Coordination Transitions Coordination processes utilize the coordination transitions presented in Table 1 [23]. Coordination processes have the capability to either allow or disallow state activations. The actual activation of its states is determined by the object itself based on its lifecycle process. As a result of reaching a particular state in the object, it becomes feasible for a coordination process to allow the activation of this state well in advance, as the transition condition is already met. Conversely, a coordination process may disallow the execution until the specified coordination constraints are met, i.e., the coordination transition becomes enabled. To enable a coordination transition, it becomes necessary for all preceding transitions in the coordination process to have been enabled as well, e.g., the Production object must be in the *Creation* state to instantiate an Oven or a Workpiece object (cf. Fig. 5).

Ports A coordination step (excluding the start coordination step) may have any number of ingoing *Ports* to which coordination transitions are connected (cf. Fig. 5). All transitions connected to a single port correspond to a logical (AND) conjunction. Multiple ports in a coordination step are considered

Table 1. Types of supported coordination transitions in PHILharmonicFlows [23]

Name	Description of the coordination transitions
TopDown	The execution of one or more lower level processes is dependent on the status of a single higher level process.
BottomUp	The execution of one higher level process depends on the execution status of one or more lower level processes of the same type.
Transverse	The execution of one or more processes is dependent on the execution status of one or more processes of a different type. Both types of processes have a common higher level process.
Self	The execution of a process depends on the completion of a previous step of the same process.
SelfTransverse	The execution of a process depends on the execution of other processes of the same type. All processes have a common higher level process.

as logical (OR) disjunctions, i.e., the coordination step is activated when all coordination transitions connected to one port are enabled.

**Fig. 5.** Coordination Process for the Production, Oven and Workpiece objects

Coordination of Concurrent Activities The coordination process depicted in Fig. 5 is responsible for managing the processes that access the Oven resource concurrently. Once a Production object becomes instantiated, the oven and workpieces can be instantiated. The 1:1 relationship between the Oven and Production objects (cf. Fig. 2) ensures that only one oven may be instantiated, whereas there is no limit on the number of workpieces. The transition of the Oven from state *Creation* to *Idle* is guided by its lifecycle (cf. Fig. 3). Similarly, the

workpiece transitions from *Creation* to *Unprocessed* (cf. Fig. 4). The oven may enter the *Burning* state when the production process explicitly transitions to the *Burn Process* state triggered by an external signal. At the same time, the oven must be in *Idle* state. This is defined by the *SelfTransverse* transition in combination with condition $\#TargetIn == 0$, which ensures that there are no ovens in state *Burning*. Additionally, there must be at least one unprocessed workpiece, as defined by the transition *Transverse* with condition $\#SourceIn \geq 1$. Once the oven enters the *Burning* state, the corresponding workpiece transitions to the *Burning* state as well. After completing the burning process, the oven transitions to the *End* state, and the workpiece transitions to the *Burned* state. Consequently, the coordination process for the production terminates (cf. Fig. 5). In the manufacturing scenario, the coordination transitions are not fulfilled if the oven is occupied and another process seeks access. This request is then queued (e.g., in a FIFO queue). The queued requests are activated in new instances of the coordination process once the oven is idle again (cf. Backwards Transition in Fig. 3) and there is at least one unprocessed workpiece.

4 Related Work & Discussion

4.1 Related Work

First, we review related work that deals with concurrency handling in general: [6] presents a software architecture for IoT applications that use *accessors* as proxies for IoT devices and a coordination framework to handle concurrency. However, the focus of this work is on timestamps and sequential execution rather than on coordinating different object instances. [5] analyzes the challenges of coordinating concurrent IoT-aware processes, exploring complexities related to resource sharing, synchronization, and scalability. While it provides valuable insights into managing concurrent processes, it falls short in providing solutions to address these challenges. [28] presents a Petri net based method for scheduling resources and controlling IoT-aware processes. This approach allows analyzing and optimizing of complex shop floor processes. However, this approach lacks scalability and might be accompanied with a suboptimal resource utilization and reduced responsiveness to changes in respect to resource availability.

Second, we review related approaches from the BPM field: [17] introduces a framework for enabling adaptive, cyber-physical processes that involve physical entities and IoT devices. While the framework allows for monitoring, exception detection, and automated resolution strategies, it emphasizes process adaptation based on IoT data, rather than activity coordination. [14] introduces semantic web services to achieve easily configurable and flexible manufacturing systems. The proposed approach uses pre- and post-conditions for verifying service requests, which may be also used to handle and queue concurrent requests [21]. [24] proposes a method for synchronizing manufacturing processes based on BPMN 2.0, with a focus on optimizing production efficiency through buffering and unbundling operations. This approach dynamically adjusts process synchronization according to the available real-time data. As a drawback, however, only

modeling patterns in BPMN 2.0 are used, and without these recipes, it is not exactly clear how process instances are coordinated [24]. *Batch activities* involving the same activities to be executed in different processes that lead to concurrency are discussed in [18]. The authors propose a modeling and execution approach for batch activities in BPMN-based business processes. As this problem resembles the concurrency issues addressed in our work, we adapt similar concepts but put a stronger focus on the data-driven lifecycle models and processes of objects involved in the executions. [15] introduces an approach for synchronizing process activities in workflow systems, enabling their dynamic coordination based on pre-specified synchronization rules [15]. While the approach enhances process control and sequencing, it lacks a comprehensive implementation of synchronization patterns. [8] categorizes various types of instance-spanning constraints (e.g., for coordination) and introduces a formal notation to represent and enforce these constraints within process models. A shortcoming is the lack of assurance that future instances will not breach inter-scenario constraints currently satisfied.

4.2 Discussion

With our work, we effectively address the lack of support for managing concurrency in IoT-aware processes [10]. IoT environments include numerous devices, objects, materials, and resources that are involved in executing IoT-aware processes. Existing approaches fail to express the complex relations and constraints among these entities to achieve the necessary coordination while maintaining flexibility. This paper discusses an answer to the research question on how to coordinate concurrent process activities in IoT environments with constrained physical resources (cf. Sect. 2). By proposing an approach based on object-centric processes in PHILharmonicFlows, we demonstrate how to model and manage concurrency in IoT-aware processes (cf. Sect. 3). The relational process type structure (cf. Sect. 3.2) aids in coordinating concurrent process activities by incorporating the relationships and interactions between objects. This hierarchy of objects and their relationships are crucial for managing access to shared physical resources in IoT. The framework allows modeling and managing object lifecycles (cf. Sect. 3.3) and thus enables the coordination of activities that access the same physical resource by enforcing semantic relationships and coordination constraints (cf. Sect. 3.4). One notable benefit of using PHILharmonicFlows is the inclusion of a runtime engine capable of executing the lifecycle models [3].

This work is focused on a specific use case from smart production. While this allowed us to illustrate basic concepts, it limits the generalizability of our findings. Note that the running example constitutes a simplified manufacturing process that handles only a single workpiece at the time. However, in real-world production environments, machines may exist multiple times and be capable of multi-tasking and batch processing to increase efficiency. While PHILharmonicFlows can be also used to model these capabilities, a potential disadvantage could be the quantity and complexity of the models required to represent these objects and coordination processes. This inherently necessitates that users acquire proficiency in the specific modeling language of PHILharmonicFlows.

5 Summary and Outlook

Process activities whose execution involve IoT devices need to be coordinated as the number of available devices and their multi-tasking capabilities might be limited. Traditional BPMS often rely on capabilities of external systems or additional components to handle concurrent activity executions, which are inflexible and incomprehensible for the user. This paper addresses the problem of coordinating concurrent process activities in IoT. We propose an object-centric approach based on the PHILharmonicFlows framework to handle concurrency in IoT-aware processes. PHILharmonicFlows allows coordinating objects which fosters concurrency control for IoT-aware processes as IoT devices can be represented as objects. The solution uses a relational process type structure to capture the relationships between objects in a hierarchical view. Object lifecycles and coordination processes are modeled and managed within the framework, allowing activities that access the same physical resource to be coordinated. We demonstrated the framework in a smart manufacturing scenario. The proposed solution addresses the challenges of coordinating activities that involve constrained physical resources, providing a flexible and comprehensible coordination mechanism.

In future research, we will extend PhilharmonicFlows beyond its current capacity for defining constraints and for handling concurrency towards facilitating dynamic and ad-hoc adaptations of IoT-aware processes. This will include additional capabilities for handling data streams and achieving real-time scheduling and work distribution with the goal of responding to immediate changes, demands, and errors within the IoT landscape.

References

1. van der Aalst, W.M.: Business process management as the “killer app” for petri nets. *Software & Systems Modeling* **14**(2), 685–691 (2015)
2. van der Aalst, W.M., Ter Hofstede, A.H., Kiepuszewski, B., Barros, A.P.: Workflow patterns. *Distributed and parallel databases* **14**, 5–51 (2003)
3. Andrews, K., Steinau, S., Reichert, M.: Enabling runtime flexibility in data-centric and data-driven process execution engines. *Information Systems* **101** (2021)
4. Arnold, L., Breitmayer, M., Reichert, M.: Monitoring object-centric business processes: An empirical study. In: *Research Challenges in Information Science: Information Science and the Connected World*. pp. 327–342. Springer Nature Switzerland, Cham (2023)
5. Bicevskis, J., Karnitis, G., Bicevska, Z., Oditis, I.: Analysis of concurrent processes in internet of things solutions. In: *Information Technology for Management: Business and Social Issues*. pp. 26–41. Springer International Publishing, Cham (2022)
6. Brooks, C., Jerad, C., Kim, H., Lee, E.A., Lohstroh, M., Nouvellet, V., Osyk, B., Weber, M.: A component architecture for the internet of things. *Proceedings of the IEEE* **106**(9), 1527–1542 (2018)
7. Derler, P., Lee, E.A., Sangiovanni Vincentelli, A.: Modeling cyber-physical systems. *Proceedings of the IEEE* **100**(1), 13–28 (2012)
8. Fdhila, W., Gall, M., Rinderle-Ma, S., Mangler, J., Indiono, C.: Classification and formalization of instance-spanning constraints in process-driven applications. In: *BPM’16*. pp. 348–364. Springer (2016)

9. Guabtni, A., Charoy, F., Godart, C.: Concurrency management in transactional web services coordination. In: Bressan, S., Küng, J., Wagner, R. (eds.) *Database and Expert Systems Applications*. pp. 592–601. Springer Berlin Heidelberg (2006)
10. Janiesch, C., et al.: The internet of things meets business process management: A manifesto. *IEEE Systems, Man, and Cybernetics Magazine* **6**(4), 34–44 (2020)
11. Kirikkayis, Y., Gallik, F., Seiger, R., Reichert, M.: Integrating iot-driven events into business processes. In: *CAiSE Forum*, Zaragoza, Spain, 2023. Springer (2023)
12. Kirikkayis, Y., Gallik, F., Winter, M., Reichert, M.: BPMNE4IoT: A framework for modeling, executing and monitoring iot-driven processes. *Future Internet* **15**(3) (2023)
13. Künzle, V., Reichert, M.: Philharmonicflows: towards a framework for object-aware process management. *Journal of Software Maintenance and Evolution: Research and Practice* **23**(4), 205–244 (2011)
14. Malburg, L., Klein, P., Bergmann, R.: Using semantic web services for ai-based research in industry 4.0. *CoRR* **abs/2007.03580** (2020)
15. Mangler, J., Rinderle-Ma, S.: Rule-based synchronization of process activities. In: *IEEE 13th Conf. on Commerce and Enterprise Computing*. pp. 121–128 (2011)
16. Mangler, J., et al.: Datastream xes extension: Embedding iot sensor data into extensible event stream logs. *Future Internet* **15**(3) (2023)
17. Marrella, A., Mecella, M., Sardina, S.: Supporting adaptiveness of cyber-physical processes through action-based formalisms. *AI Communications* **31**, 47–74 (2018)
18. Pufahl, L., Weske, M.: Batch activity: enhancing business process modeling and enactment with batch processing. *Computing* **101**(12), 1909–1933 (2019)
19. Schulte, S., Janiesch, C., Venugopal, S., Weber, I., Hoenisch, P.: Elastic business process management: State of the art and open challenges for bpm in the cloud. *Future Generation Computer Systems* **46**, 36–50 (2015)
20. Seiger, R., Herrmann, S., Aßmann, U.: Self-healing for distributed workflows in the internet of things. In: *2017 IEEE ICSAW'17*. pp. 72–79 (2017)
21. Seiger, R., Malburg, L., Weber, B., Bergmann, R.: Integrating process management and event processing in smart factories: A systems architecture and use cases. *Journal of Manufacturing Systems* **63**, 575–592 (2022)
22. Steinau, S., Andrews, K., Reichert, M.: The relational process structure. In: *Advanced Information Systems Engineering: 30th Inter. Conf., CAiSE 2018, Tallinn, Estonia, June 11–15, 2018, Proceedings 30*. pp. 53–67. Springer (2018)
23. Steinau, S., Andrews, K., Reichert, M.: Enacting coordination processes. *arXiv preprint arXiv:2012.08409* (2020)
24. Traganos, K., Spijkers, D., Grefen, P., Vanderfeesten, I.: Dynamic process synchronization using bpmn 2.0 to support buffering and (un)bundling in manufacturing. In: *Business Process Management Forum*. pp. 18–34. Springer, Cham (2020)
25. Valckenaers, P., Van Brussel, H., Verstraete, P., Saint Germain, B., Hadeli: Schedule execution in autonomic manufacturing execution systems. *Journal of Manufacturing Systems* **26**(2), 75–84 (2007)
26. Vanderfeesten, I., et al.: Developing process execution support for high-tech manufacturing processes. Empirical studies on the development of executable business processes pp. 113–142 (2019)
27. Vasilecas, O., Smaizys, A., Rima, A.: Business process modelling and simulation: hybrid method for concurrency aspect modelling. *Baltic Journal of Modern Computing* **1**(3–4), 228–243 (2013)
28. Wu, X., Tian, S., Zhang, L.: The internet of things enabled shop floor scheduling and process control method based on petri nets. *IEEE Access* **7**, 432–442 (2019)