



Online detection of process activity executions from IoT sensors using generated event processing services

Ronny Seiger^{ID*}, Aaron F. Kurz^{ID}, Marco Franceschetti^{ID}

Institute of Computer Science, University of St.Gallen, Rosenbergstrasse 30, St.Gallen, 9000, Switzerland

ARTICLE INFO

Keywords:

Internet of Things
Business process management
Activity detection
Event abstraction
Process mining

ABSTRACT

Data streams from Internet of Things (IoT) devices containing sensors and actuators provide new insights into their interactions, context, and process executions in the physical world. These new data sources may open up novel opportunities to apply Business Process Management (BPM) technologies to analyze process and activity executions using established process mining techniques. However, the rather low abstraction level of data emitted from the IoT devices is often not suitable to directly apply process mining, which requires additional steps of event abstraction. Related approaches train expensive supervised machine learning models on historical sensor data to realize this event abstraction enabling only a post-mortem classification of activity executions. In this work we propose a framework to automatically generate activity detection services from IoT data with minimal human involvement to implement the event abstraction. Along with the framework, we present a software architecture focused on a flexible and extensible complex event processing (CEP) platform that achieves high-performance activity detection from IoT data streams at runtime—enabling online process analytics. Evaluations of our proof-of-concept implementation to monitor processes executed in smart manufacturing and smart healthcare show acceptable results when detecting activities that are affected by no to only small variations in the underlying IoT data. We identify several ways to improve the robustness of the activity detections regarding variations in IoT data as starting points for future work.

1. Introduction

The Internet of Things (IoT) emerged as a new technology-based paradigm to foster the interaction of software systems with the physical world through connected devices and objects enabled via sensors and actuators [1]. Sensors act as new sources providing streams of real-time data about the execution of activities and tasks, interactions and states of devices and objects, and their surroundings and context. In Business Process Management (BPM), Process-aware Information Systems (PAIS) are used to execute and monitor process and activity executions in digital systems, creating digital traces in *event logs* that can be used for process analysis tasks such as *process mining* [2,3]. Many IoT environments are also relying on processes to specify the normative sequence of tasks, executed by digital and physical resources such as humans and machines, to produce a specific outcome, which is of value to different stakeholders. However, in IoT settings, there usually is no central PAIS available to create event logs suitable for process mining [3,4]. Instead, we are faced with a heterogeneous set of IoT devices that are controlled by different software applications providing data on different levels of abstraction [5]. Moreover, considering the human as an active process performer requires additional IoT-based monitoring

solutions to create comprehensive digital traces in the absence of PAIS. To extract process and activity related information—*process events*—from IoT data suitable for process mining, existing research focused on the detection of specific types of activities relying on specifically trained machine learning (ML) models based on specific sensors [6]. Other more generic approaches exist, but are only applicable to identify activity executions from IoT data post-mortem [2,7,8]. Being able to detect the execution of process activities at runtime in a generic and sensor-agnostic way, independent of the availability of PAIS for process monitoring, opens up new opportunities for light-weight process analytics to check the correctness of activities executions and provide feedback about conformance and compliance. These analytics, in turn, may then foster the development of IoT-based business applications [9], and provide feedback and insights into optimizing and “repairing” non-compliant activity executions. This could lead, for example, to increased workplace safety by making sure that treatment processes in smart healthcare are executed correctly, or to reduced operational costs and waste by detecting malfunctioning production machines in smart manufacturing in a timely manner [4].

* Corresponding author.

E-mail addresses: ronny.seiger@unisg.ch (R. Seiger), aaron.kurz@unisg.ch (A.F. Kurz), marco.franceschetti@unisg.ch (M. Franceschetti).

<https://doi.org/10.1016/j.future.2025.107987>

Received 13 February 2025; Received in revised form 17 June 2025; Accepted 18 June 2025

Available online 1 July 2025

0167-739X/© 2025 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

In this work we present a framework to flexibly generate services implementing the detection of arbitrary process activities at runtime from IoT data based on Complex Event Processing (CEP) as underlying technology. From an *Activity Signature*, which represents the prototypical sensor readings associated with a process activity as time series, we generate services that are able to detect the occurrence of this activity based on pattern and sequence matching leveraging CEP functionality. The generator component in the software framework supports different pattern matching strategies that can be easily extended. Compared to most machine learning-based approaches, the patterns in the sensor data are formalized and understandable by humans. The generated activity detection services can be modified by domain experts to leverage additional domain knowledge, automatically deployed, and composed on a light-weight, extensible CEP engine to generate high-level activity and process-related events from detected patterns in the IoT event streams. These high-level events serve as basis for process mining, both in online and offline settings. We demonstrate and discuss the framework as a proof-of-concept from automated service generation to deployment and online detection, with human-in-the-loop, using examples from smart manufacturing and smart healthcare. The approach proves to be generalizable and also applicable to less automated, but process-oriented IoT domains for analysis of manual activities, which are tracked through IoT devices. A detailed evaluation of the activity detection services with different detection patterns is provided for two production processes in the domain of discrete smart manufacturing and a blood donation process in smart healthcare. Throughout the paper, we discuss solutions to challenges and concrete implementations of steps regarding the general framework that derives business process-related information from IoT data presented in [10]. The contributions of this paper are:

- A data-driven service generation framework that automates the development of services for process activity detection from IoT data streams with human-in-the-loop;
- A software architecture that demonstrates the use and composition of the generated activity detection services at runtime in IoT scenarios;
- Comprehensive evaluations of the activity detection services in real-world smart manufacturing and smart healthcare settings demonstrating the IoT based monitoring of activity executions.

The paper is structured as follows: Section 2 introduces basic concepts and requirements. Section 3 discusses related work. Section 4 presents the framework for the generation of activity detection services. Section 5 elaborates on the software architecture for activity detection at runtime. Section 6 provides details on the implementation of the service generation prototype. Section 7 presents an evaluation of our approach in smart manufacturing and smart healthcare. Section 8 discusses our approach. Section 9 concludes the paper and presents future work.

This paper is an extended version of the work presented in [11]. It represents a second iteration of the design science cycle [12] and it has been extended as follows: a new use case from the domain of smart healthcare to provide explanations and examples throughout the paper has been added; requirements have been revised and a new requirement (*human-in-the-loop*, R3) has been added; related work (Section 3) has been updated and extended; the explanations of the activity detection framework (Section 4) have been significantly revised and extended, including now also explicit considerations of domain experts in the entire process; the discussion of partial sequence detection in the framework has been added; a new section describing the runtime aspects and software architecture of the system (Section 5) has been added; a new section providing an explanation of the service generator's implementation (Section 6) has been added; a new section with an extensive evaluation of the activity detection for example processes in the two IoT domains smart manufacturing and smart healthcare has been added (Section 7); the discussion of results and requirements (Section 8) has been revised and extended.

```

1 {"id": "f0a42ccb-b7ec-4a02-bdf3-46e0655712c3",
2  "station": "OV_1", "i1_pos_switch": 0,
3  "i2_pos_switch": 1, "i5_light_barrier": 1,
4  "m1_speed": 0, "o7_valve": 0, "o8_compressor": 0,
5  "timestamp": "2023-04-11 10:34:56.03"}

```

Listing 1: Example low-level IoT event from Oven station.

2. Fundamentals and requirements

2.1. Process activities and events

The main objective of this work is to develop an approach to detect activity executions – emitting process-level events for process mining – from IoT data streams. In this context, an *activity* is considered to be an atomic unit of work within the context of a business process, which has a duration derived from its start and end, and which is executed by a human or non-human process performer [13]. These non-human process performers may include digital systems, but also IoT devices [10]. Closely related is the notion of *event* used to capture the occurrence of something of relevance that has happened in the business process execution (e.g., the start of an activity) [14]. We refer to these events as *process-level events*. In contrast, events are also used to capture the state of IoT components and their surroundings *sensed* at a certain point in time [1,15]. We denote these as *low-level IoT events* contained in event streams. Between these extremes, there is a spectrum of events referring to different levels of abstraction and granularity [10,16]. Moving between these levels requires to perform event processing, event aggregation, event transformation, and event correlation [10,17,18]. Note that common business process notations (e.g., BPMN 2.0) also feature the event concept in process models to denote a specific trigger in a business process [14]. We do not include these types of events in our investigations.

2.2. Internet of Things use cases

2.2.1. Smart manufacturing

In this work, we use a small-scale smart factory model as representative of an industrial IoT environment (cf. Fig. 1) [19]. The model allows for simulating the execution of various activities in a production line as part of real-world discrete manufacturing processes. It features different IoT components (production machines) that act as fully automatic performers of process activities, namely a robot, a warehouse, an oven, a sorting line, a milling machine, and a transport unit. Processes composed of (production) activities inherently exist, but a central PAIS to record event logs suitable for process mining is not available [3]. In contrast, 53 sensors and 24 actuators emitting a plethora of low-level IoT events during process executions are built into the 7 production stations.

IoT sensors. Each of the IoT components is equipped with a number of sensors $i_1, \dots, i_n$ as inputs, motors $m_1, \dots, m_m$ as actuators, other output devices $o_1, \dots, o_o$, and machine-specific sensors (e.g., positions of robots) [20]. Real-time access to the values of these sensors and actuators emitted from the factory control system is enabled via an MQTT-based message broker. Messages are published with a customizable frequency in a one-topic-per-component manner, i.e., one message contains the values of all sensors and actuators for the specific IoT component at a specific time [21]. Listing 1 shows an exemplary low-level IoT event from the oven station. Each of these values is considered to be one timestamped low-level IoT event and we assume that sensors and actuators of one production station have dependencies among each other but not with other stations. We view this setup as a typical IoT setting exhibiting characteristics relevant for our work.

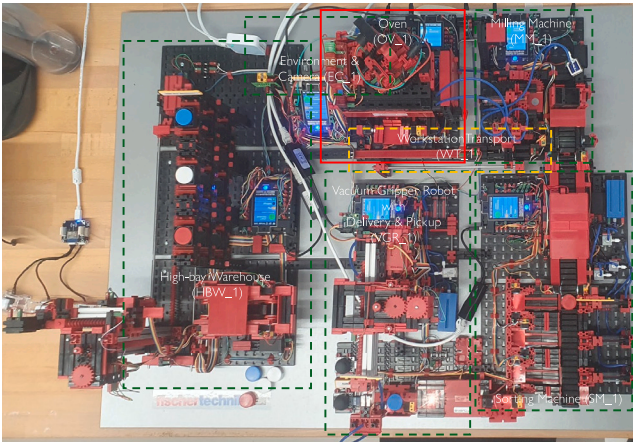


Fig. 1. Components of smart factory model with Oven highlighted.

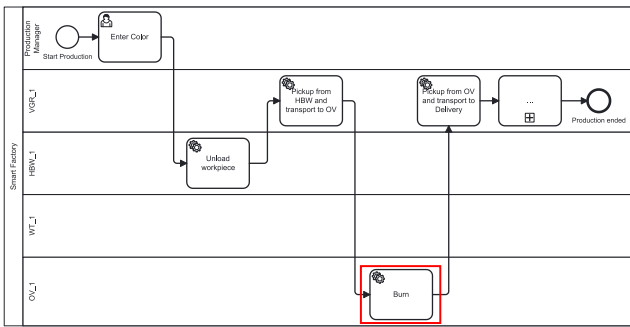


Fig. 2. Parts of the Production process in BPMN 2.0 with highlighted Burn activity.

Example process activities. Fig. 2 depicts a typical production process model (in BPMN 2.0) in the smart factory, including the unloading of a workpiece, transport activities, and the burning of the workpiece as highlighted process activity. These activities are modeled at the level of granularity that comprises production capabilities of individual stations, which is the granularity at which we aim to detect them with our approach from low-level IoT data.

2.2.2. Smart healthcare

In a second scenario, we aim to detect the execution of process activities executed by humans in a smart healthcare setting. In our lab setup (cf. Fig. 3), we simulate the execution of a blood donation process with one healthcare worker and up to two simultaneous blood donors [22].

IoT sensors. In this setup, 20 IoT sensors are deployed to monitor the execution of process activities. We have grouped the sensors into three different stations (cf. Fig. 3). The left and right donor stations both feature a motion sensor, a distance sensor and a brightness sensor to determine if the donor has sat down and the hand has been placed on the table, an NFC reader, an accelerometer mounted to the lid of the blood drawing machine, and three different buttons for user interactions. The central hygiene station features a scale to detect the use of hand sanitizer, an NFC reader, a distance sensor, and an environment sensor. Listing 2 shows an exemplary low-level IoT event from the hygiene station, grouping several sensors in a similar way as the smart factory. In contrast to the smart factory setup, we may also observe dependencies/correlations of sensors across the different stations when an activity is performed by the healthcare worker spanning the central and left or right donor stations.

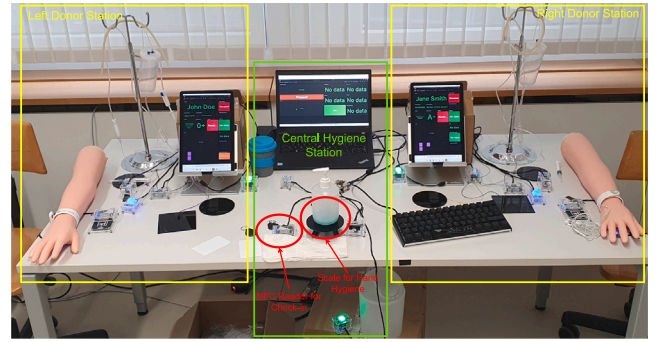


Fig. 3. Setup of the blood donation monitoring system with NFC reader and scale highlighted.

```
1 {"id": "e222a84b-8401-46a4-8824-6849e7f35b7e",
2  "station": "HYGIENE-STATION",
3  "load_cell_weight": 0,
4  "ir_long_distance": 150.0, "button_state": 0,
5  "air_quality": "high", "air_temp": 24.19,
6  "air_humidity": 31.88, "air_pressure": 916.43,
7  "nfc_status": 1, "nfc_tag": "04DAF28AB45780",
8  "timestamp": "2024-01-18 11:35:41.58"}
```

Listing 2: Example low-level IoT event from hygiene station.

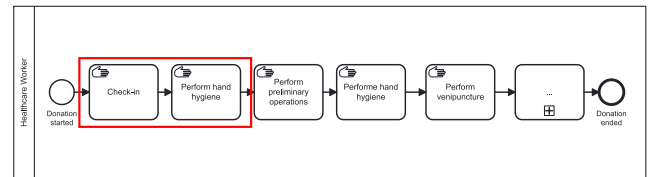


Fig. 4. Parts of the blood donation process with highlighted manual process activities.

Example process activities. Fig. 4 shows parts of the blood donation process including several activities that are performed manually. The highlighted activities refer to the check-in of the healthcare worker and them performing a hand hygiene activity before moving on with the preliminary operations [22].

2.3. Methodology and requirements

The main objective of our work is to develop a framework to detect the execution of manual and automated process activities from low-level, raw sensors streams in typical IoT settings (e.g., smart manufacturing and smart healthcare as illustrated in the previous sections). As our work is closely related to the engineering of information systems, we continue following the well-established design science research methodology (DSRM) for information systems engineering in a second iteration [12]—building upon the results and feedback from the work published in [11]—for developing this activity detection framework.

The following non-exhaustive list of non-functional requirements was derived based on the authors' experience in the field of BPM and IoT and on the characteristics of IoT environments/Cyber-physical Systems (CPS) that fall within the scope of our work under the assumption and motivation that a central PAIS for consistently monitoring process/activity executions is not available [3,23]. The requirements identify several problems associated with the activity detection in IoT scenarios and serve as basis for further defining the objectives of our solution.

R1 Non-invasiveness: Typical IoT settings exhibit a heterogeneous software landscape with different applications controlling the

IoT devices [5,24]. While some of these applications might be open to extension (e.g., to emit process-related data), many control systems are proprietary and limited in their interfaces. Our approach should work non-invasively with existing systems assuming that they use at least one standard communication protocol or provide one open interface to emit data about the IoT components they are controlling. Moreover, it shall not be necessary to introduce a heavy-weight software component (e.g., a PAIS) to the IoT environment.

R2 No-code services: The activity detection shall be added to existing IoT environments in a flexible manner to be used by other software components. Web services have proven to be a suitable technology in IoT that facilitates loose coupling [25]. However, there might be a large number and variety of different activities to be detected in an IoT environment [24]. New types of activities might be added anytime, requiring additional **extensibility** of the activity detection system. A manual implementation of Web services capable of detecting each type of activity seems to be infeasible. Moreover, the IoT data produced by individual sensors is often simple enough to follow a pattern matching approach rather than the complex and expensive training of machine learning models. The activity detection services shall require little to no training or implementation (*no-code*) effort for service providers, enabling also domain experts which are often non-software developers to create and deploy (*no-ops*) detection services for new types of activities [26].

R3 Human-in-the-Loop: Despite the focus on the low-/no-code approach, users (domain experts) should be kept in the loop and be enabled to inspect and make adjustments to the generated services to incorporate their expertise [27]. This is still important because not all the knowledge relevant to detect the occurrence of activity executions from low-level IoT might be available in a machine-readable and processable format [10]. In turn, this requirement makes it necessary to present the activity detection logic and patterns in a way understandable and modifiable by humans [27].

R4 Activity detection at runtime: Systems and entities in IoT and CPS are by definition interacting with the physical world when executing activities. Here, only performing a post-mortem analysis of data is not sufficient anymore as activity detection and decision making *at runtime* is crucial to provide (near) real-time feedback and to react to new and unforeseen situations [28]. Therefore, activity detection must work at runtime, possibly already considering partial occurrences of an activity from its start without having to wait until its completion. In smart manufacturing, this might be relevant for example to detect that the incorrect manufacturing activity has been started on a specific machine and to consequently abort its execution. In the smart healthcare scenario this might be relevant for example to inform the healthcare worker about the missing execution of a required hand hygiene activity when detecting the start of another activity in the blood donation process—which is critical to prevent the risk of cross-contamination. These cases both relate to a violation of process conformance [13].

R5 Robustness: IoT components interact with the physical world during activity execution and thus are subject to varying external (context) factors and emergent behavior that cannot always be anticipated and easily reproduced perfectly for each instance of an activity execution [28]. This influences the activity instances and their manifestations in the IoT data, e.g., additional sensors or actuators may be used depending on process parameters and context, or sensor values and execution times may vary, for the same type of activity. The activity detection shall be robust and flexible to a certain degree regarding these variations, supporting different detection strategies to handle variability.

We use these requirements to design and develop a solution in the form of two artifacts, namely (1) a software framework to generate activity detection services (cf. Section 4), and (2) a software architecture of a service-based activity detection system (cf. Section 5). Along with the conceptual explanations, we demonstrate both artifacts and their usage in these sections, and provide more details on our proof-of-concept implementation in Section 6. The evaluation in Section 7 shows the results of instantiating activity detection services for different processes in the two main application domains we are investigating, namely smart manufacturing and smart healthcare. In line with the DSRM [12], this work communicates the results of the second process iteration of our investigations regarding the IoT-based detection of process activities [11].

3. Related work

Prior works studied the integration of IoT technology with BPM, with the BPM-IoT manifesto [4] at the forefront. The data-driven approach proposed in this paper involves tasks of event extraction, abstraction, and correlation: Diba et al. in [18] give a general overview on existing approaches to these tasks, but do not discuss approaches considering the IoT as a data source.

Event stream processing. Related to the processing of IoT data streams is the work in [29], which proposes an architecture combining CEP with stream processing to consume and manipulate heterogeneous IoT data in real time. The proposed architecture, however, relies on developers implementing the CEP applications. We aim at bridging this gap and alleviating analysts' workload via automation (cf. R2). A similar intent is found in [30], with a visual framework for programming CEP apps processing IoT data; a visual approach for developing IoT-based monitoring systems is also presented in [31]. While the goal of these works is to support developers by abstracting low-level design aspects, the approaches still involve manual interventions in a visual programming environment and manually defining rules associated with the occurrence of events. In contrast, we aim to provide an approach for a highly automated (no-code) generation of activity detection services (cf. R2).

In [32], the authors propose a method to automatically generate CEP queries associated with the lifecycle transitions of process control flow elements for process monitoring. The method is based on the annotation of a process model with *Monitoring Points*, and assumes formalized, structured knowledge of a process in terms of a model. We relax this assumption and allow unstructured process knowledge, requiring only little interactions with the domain expert for manual adjustments and inspections (cf. R3). The approach in [33] allows automatically learning and generating CEP rules for activity detection from historical traces. However, it requires a corpus of data for the training phase, which might not always be available. Instead, our approach aims to be applicable also in cases where only one or few traces are available (cf. R2).

Event abstraction from IoT data. Event abstraction focuses on bridging the gap between the granularity at which the data is recorded and at which it is analyzed [4,18,34]. When considering sensor data, the challenge of mapping fine-grained sensor data to more abstract process activities becomes more pronounced [4,35,36]. In literature [36], various approaches exist to bridge this abstraction gap using machine learning (ML) [37–39]. While these approaches are relatively effective, they do not support the domain experts with inspecting and adjusting the activity detection patterns and models due to their complexity (cf. R3). A notable work in this context to facilitate the understandability of ML-based inference models – although applied in a different domain – aims to create causal relation graphs that illustrate the relations between events explaining *why* certain events happened [40]. This work is highly relevant to our use case as we also aim to make the event relations explicit and even modifiable. The step of bridging

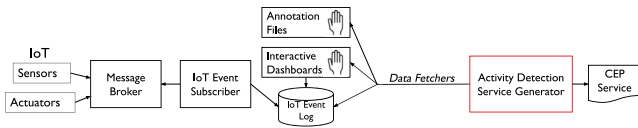


Fig. 5. Framework for generation of activity detection services.

the abstraction gap between low-level IoT events and process events is further known to potentially introduce ambiguity [41,42]. Leveraging additional information such as contextual data might be useful to mitigate such a risk [43].

In [44], Senderovich et al. propose a knowledge-driven approach based on interactions mining to transform historical logs of sensor data into standardized event logs including information about process instances. In [45], Mannhardt et al. propose a supervised method for event abstraction based on activity patterns. These patterns are derived from low-level events using domain knowledge and then used to build an abstracted event log by exploiting existing alignment techniques. Wanner et al. combine expert knowledge and observations with machine learning in the context of a smart factory [46]. All these approaches, again, focus on offline training and classification (cf. R4). To be suitable for the application of process mining techniques, the coarse-granular events then need to be matched to the process activities in a process model or correlated to the process instances through *event-to-activity mappings* [47,48], which is out of scope of our work. We assume that this correlation is known and refer to [17,36,49,50] for broader literature overviews event abstraction and correlation for process mining.

IoT-based activity detection. One of the main benefits of the IoT data sources is the improved recognition of activities and processes from sensor data, e.g., with the help of multi-modal approaches such as the one by Rebmann et al. [51] that combines motion and vision sensors with user feedback for detecting and disambiguating known activity types/tasks in real-time. More specialized approaches to activity detection, often using specialized sensors, have been adopted in the context of smart homes [7,52], smart environments and ambient assisted living [53,54], smart cars [2,55], human activity detection [6, 56], and industrial processes [57]. Most of these approaches rely on the resource intensive, fully or semi-supervised training of models and are only suitable for classification of data in offline settings, where data is analyzed post-mortem and fit to a specific problem. Their robustness to variations in sensors data is usually quite high but only fit to the given problems, scenarios, and sensors (cf. R5). Moreover, none of the discussed approaches demonstrates its applicability for the detection of activities *at runtime* as inference is too costly and time consuming. In contrast, we aim at developing small footprint, extensible detection services (cf. R1, R2) that are also working well in online settings to detect arbitrary activities in near real-time (cf. R4).

4. Software framework to generate activity detection services

The activity detection framework consists of a phase in which the activity detection services are generated from collected low-level IoT events and annotations by the domain expert—described in this section. Section 5 elaborates on the deployment and runtime phases of using the services. Fig. 5 presents an overview of the framework for generating activity detection services. With R2 we emphasize the need for a no-code solution to detect the potentially large variety of activity types in IoT. In combination with R1 assuming that the IoT stations provide interfaces to access data about their sensors, actuators, states and other properties, we propose to follow a *data-driven approach* to generate services for activity detection.

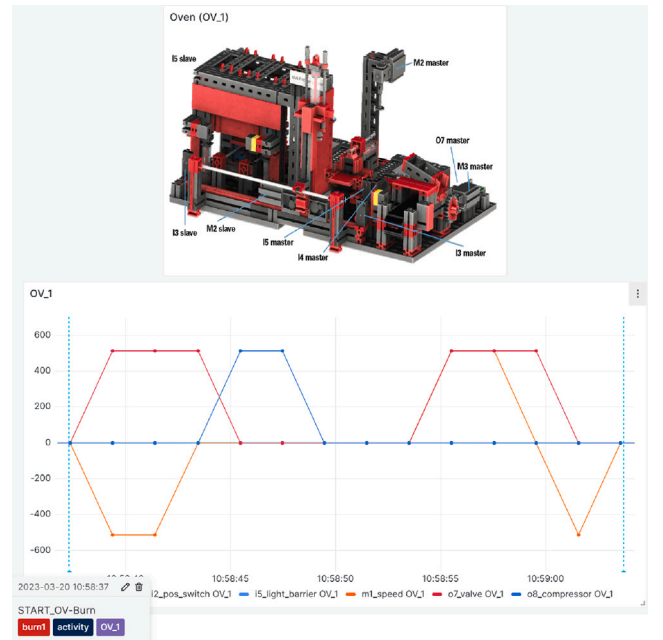


Fig. 6. Low-level IoT data annotated by the domain expert indicating the start and end of the burn activity in the oven station.

4.1. IoT data collection

Low-level events from the IoT stations' sensors and actuators are assumed to be published via a message broker (e.g., an MQTT or Apache Kafka broker) as described in Section 2.2. This data is retrieved by an IoT Event Subscriber and persisted in an IoT Event Log (e.g., a time series database such as InfluxDB). We propose to persist this data on a per-production-station basis, grouping the data readings of all sensors and actuators of a production station at one specific point in time (e.g., in a bucket). To create a basis for the generation of activity detection services, the low-level IoT events related to the execution of at least one representative instance of an activity to be detected must be recorded in the IoT event log.

4.2. IoT data labeling by domain expert

Following the work in [20], we rely on the domain expert applying an interactive method to identify activity executions observed in the IoT data from the IoT Event Log. With R3 we argue that the involvement of domain experts (human-in-the-loop) is a fundamental aspect in many IoT scenarios to incorporate their expertise and domain-specific knowledge, which is otherwise not available.

In a dashboard (e.g., implemented in Grafana) retrieving the data from the IoT event log and showing all the low-level readings of the sensors and actuators belonging to the relevant IoT stations over time, the domain expert annotates the beginning and end of an activity and provides a label (name) for the activity. This requires the domain expert to possess knowledge about the IoT data–process activity associations, especially to identify the patterns indicating an activity's start and end in the IoT data. In line with the work in [20], we assume for the smart manufacturing scenario that the execution of an activity is limited to one production station and that a production station executes only one activity at a time, which is reasonable in many IoT/CPS domains such as discrete manufacturing. In the smart healthcare setting, the sensors of one or more IoT stations might be annotated for one activity as it might unfold across multiple stations; still, we reasonably assume, similar to the manufacturing scenario, that the healthcare worker performs only one activity at a time.

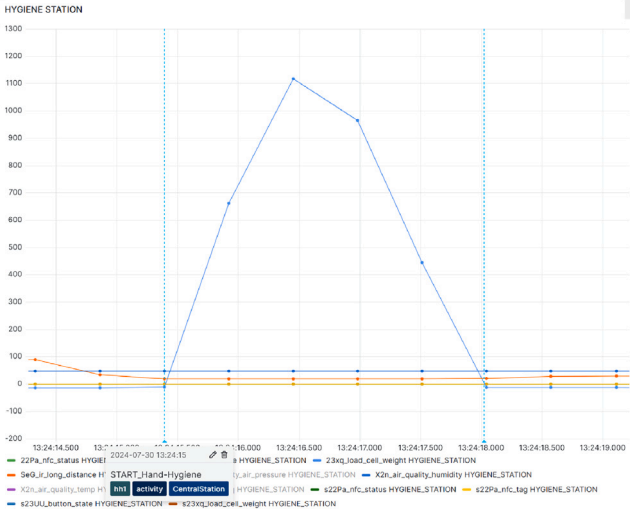


Fig. 7. Low-level IoT data annotated by the domain expert indicating the start and end of a hand hygiene activity.

```

1 sensor,source,target
2 s23qx_load_cell_weight,<=300,unused
3 s23qx_load_cell_weight,>300,in-use

```

Listing 3: Discretization mapping for the weight sensor of the central hand hygiene station.

Fig. 6 shows the annotated low-level IoT data in the smart manufacturing scenario. Following the method described in [20], the domain expert identified the activation of a motor (m1) and a valve (o7) to be indicators for the start of a production activity in the oven station. Similarly, its end is indicated by the last active motor (m1) coming to a halt. Fig. 7 shows the annotation of the hand hygiene activity in the low-level IoT events emitted from the central hand hygiene station in the smart healthcare setup. Here, the main indicator is the change in the values of the scale (load cell), which is used to determine if pressure by the healthcare worker on the hand sanitizer positioned on the scale is applied.

Additionally, the labeling of IoT data by the domain expert includes the marking of sensors to be relevant or irrelevant for the specific activity detection after labeling the start and end of the respective activity [10]. Not all the sensors of a specific station might be relevant for the detection of an activity (e.g., the environment sensor in the central hygiene station continuously emitting data). Furthermore, some sensors may provide highly variable continuous data, where only changes between certain ranges are semantically meaningful. Thus, we allow the domain expert to also provide discretization annotations, which transform the continuous values into discrete categories for the purpose of the detection services. For example, in Fig. 7 we observe that the signature of the hand hygiene activity is mostly based on the cell weight sensor which produces a continuous value related to the pressure/weight applied to it. This value (in milligrams) will most likely not be the same for each execution of the activity. Thus, this sensor needs to be discretized, e.g., to the two states *not-used* and *in-use*, in the context of detecting the usage of the hand sanitizer in the smart healthcare scenario, as can be seen in Listing 3. We expect these discretizations to be optionally provided by the domain expert for the relevant sensors when extracting the activity signatures.

4.3. Extracting activity signatures

In [20] the authors introduce *Activity Signatures* defined as the sequence of low-level IoT events from an IoT station's sensors and

actuators represented as multi-variate time series, with a unique activity label, within the activity boundaries. It “represents the physical manifestation of the execution of the corresponding process activity in the sensors and actuators” [20]. We expect the domain expert to identify *one* specific representative instance for each type of activity to be detected in the IoT data, denoted as *Activity Prototype* from which the signature is extracted by querying the IoT event log. This activity prototype is annotated in the data as described. During the extraction from the IoT event log, the values of all sensors marked as irrelevant are ignored.

Fig. 6 visualizes the activity signature for the activity prototype of *Burn* executed by IoT station *Oven* (cf. Section 2.2). It shows the time series of low-level IoT events for 6 sensors and actuators in total, each represented as a graph in a different color (absolute timestamp on x-axis; value of sensor/actuator on y-axis). Table 1 contains the low-level IoT events for all sampled timestamps $t_0..t_{12}$ at sampling rate of 0.5 Hz. The *Burn* activity starts with opening a door (*o7_valve*) and driving in a sled with a workpiece (*m1_speed*) until a switch is triggered (*i1_pos_switch*). The door then closes (*o7_valve*). The oven waits for a specified amount of time, then opens the door and drives the sled out until another switch is triggered (*i2_pos_switch*). Finally, the door closes (*o7_valve*).

Note that this example shows the activity signature of the burn activity based on absolute values of sensors and actuators at specific points in time. While this configuration is reasonable for the smart factory used in our experiments with exact reproducibility for most sensors/actuators, the values of other sensors may need an additional step of *discretization* to account for variability and make the activity detection more robust (cf. R5) [10], as described before for the hand hygiene activity.

4.4. Generating CEP-based services from signatures

Once the domain expert identified a suitable activity prototype they have to make the relevant annotations (such as start time, end time, label, IoT station/s), as well potentially provide sensors to ignore and the discretization mapping. These annotations are then retrieved from the interactive dashboards and annotation files, and the corresponding low-level IoT data is fetched by the *data fetchers* (cf. Fig. 5). These data fetchers are a flexibly adaptable part of the *Activity Detection Service Generator*, which is one of the core components we are proposing to automatically generate a service capable of detecting executions of the corresponding activity (cf. Section 6).

We have decided to use CEP as the basic technology to implement the activity detection services. CEP is designed for working with high amounts of events in multiple streams in online settings (cf. R4) [58, 59]. One of the main capabilities of CEP is its ability to process one or multiple subsequent events on one or many different streams to detect patterns among a high number of events at runtime, which is perfectly fitting to our requirements and use case of detecting low-level patterns in IoT sensor events to derive higher level events using pattern matching. CEP has proven its feasibility in similar application scenarios involving IoT and BPM considering the abstraction of low-level IoT events to process-level events [17,60]. In contrast to ML-based approaches, CEP is capable of (near) real-time event processing and detecting partial occurrences of activities (cf. R4) [58]. Typical CEP platforms feature a dedicated event processing language (EPL), which simplifies the development and understanding of CEP-based applications—*CEP apps* (cf. R2, R3). We generate a CEP app for detecting an activity based on its signature step-by-step as follows.

4.4.1. Change detection

In this work, we assume that the main indications of starting, ending, and progressing of the activity executions are changes in the corresponding low-level IoT events emitted from sensors and actuators as they signify active work via the actuators or state changes via the

Table 1
Activity signature for activity prototype of *Burn*.

Sensor/Actuator	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
i1_pos_switch	0	0	1	1	1	1	1	1	1	0	0	0	0
i2_pos_switch	0	0	0	0	0	0	0	0	0	0	0	1	0
i5_light_barrier	1	1	1	1	1	1	1	1	1	1	1	1	1
m1_speed	0	-512	0	0	0	0	0	0	0	512	512	0	0
o7_valve	0	512	512	512	0	0	0	0	0	512	512	512	0
o8_compressor	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2
Changes in the activity signature for *Burn*.

	Timestamps (t_i, t_{i+1})	Changes
1	(t_0, t_1)	m1_speed: 0 $\rightarrow$ -512; o7_valve: 0 $\rightarrow$ 512
2	(t_1, t_2)	i1_pos_switch: 0 $\rightarrow$ 1; m1_speed: -512 $\rightarrow$ 0
3	(t_3, t_4)	o7_valve: 512 $\rightarrow$ 0
4	(t_8, t_9)	i1_pos_switch: 1 $\rightarrow$ 0; m1_speed: 0 $\rightarrow$ 512; o7_valve: 0 $\rightarrow$ 512
5	(t_{10}, t_{11})	i2_pos_switch: 0 $\rightarrow$ 1; m1_speed: 512 $\rightarrow$ 0
6	(t_{11}, t_{12})	i2_pos_switch: 1 $\rightarrow$ 0; o7_valve: 512 $\rightarrow$ 0

sensors [10]. For each pair of consecutive timestamps (t_i, t_{i+1}) the changes among the values of all sensors and actuators are derived. Table 2 shows the result for the *Burn* activity signature with relevant timestamps. Note that not all pairs (t_i, t_{i+1}) and not all sensors/actuators lead to entries in the table due to the absence of relevant changes, which automatically discards additional irrelevant sensors. Here it is important that all the sensors not contributing to the activity execution are marked as irrelevant and that the discretization properly handles frequently changing, continuous values where not every change might be relevant. When detecting changes, we might also observe that the sampling frequency of the low-level IoT events can have an effect on the results. In case the frequency is too low, not all changes in the IoT data might be registered.

4.4.2. Change pattern translation

Observed recurring relations between events define event patterns [58]. Among the most important patterns supported by EPLs is the *Event Sequence Pattern*, which considers change patterns in the attributes of consecutive events within one or multiple event streams [58]. As this applies well to IoT time series data, we leverage the pattern by translating the detected changes into *queries* of an EPL to be used for pattern and change detection by a CEP engine. For each pair of consecutive timestamps (t_i, t_{i+1}), each change in the event attributes derived in the previous step is translated into the concrete syntax of the EPL for the event sequence pattern. Depending on the event stream topology, this leads either directly to a conjunction of patterns among the attributes of two subsequent events (e_x, e_y) of the same stream; or to the joining of events from different streams $s_j \dots s_k$ first, to be part of the same stream $s_{join} = s_j \bowtie \dots \bowtie s_k$, followed by the conjunction of patterns among the attributes of events (e_x, e_y). At this point, the values of event attributes emitted from sensors that are explicitly classified as relevant for the activity detection are also added to the specific query by means of a conjunction. Listing 4 shows the exemplary translation of a change in the signature of *burn* for timestamp pair (t_0, t_1) (Table 2, row 1) into a query of *Streaming SQL* used within the WS02 *Siddhi* CEP platform [59].

Note that according to our IoT setup (cf. Section 2.2), all events of one IoT station arrive as messages of the same stream. The low-level IoT events for all sensors and actuators of the IoT station are encoded as attributes of these events, i.e., no join of events from different streams is needed. This join becomes necessary when multiple IoT stations are involved in one activity, e.g., for some cases in the smart healthcare scenario. With R1 we emphasize the need for a non-invasive approach, which means that the event stream topology for low-level IoT events must be aligned with the existing IoT infrastructure and the provided interfaces. This requires additional topology and IoT data knowledge

```

1 @info(name="Detect-LowLevel-Pattern-1-Burn")
2 from every e1 = OV_1Stream, e2 =
   OV_1Stream[(e1.m1_speed==0 and
   e2.m1_speed==512) and (e1.o7_valve==0 and
   e2.o7_valve==512)]
3 select "Burn-1" as name, "Burn" as activity,
   "(t0,t1)" as time
4 insert into LowLevelPatterns;

```

Listing 4: Query for change detection in the event sequences of the *burn* activity for timestamps (t_0, t_1)

```

1 @info(name="Detect-LowLevel-Pattern-1-HandHygiene")
2 from every e1 = HygieneStation_Stream, e2 =
   HygieneStation_Stream[(e1.load_cell=="not-used"
   and e2.load_cell=="in-use")]
3 select "Hand-hygiene-1" as name, "Perform hand
   hygiene" as activity, "(t0,t1)" as time
4 insert into LowLevelPatterns;

```

Listing 5: Query for change detection in the event sequences of the *hand hygiene* activity.

to be available before event stream and query generation, or manual configuration effort later in the CEP app by the domain expert.

The translated change patterns for the Oven-Burn example can be found in Listing 4, line 2. The attributes of a tuple of two subsequent events ($e1, e2$) on the stream related to the Oven (*OV_1Stream*) are analyzed for the occurrence of the change patterns derived before. We denote the patterns that directly relate to the changes within the low-level IoT events between two points in time as *Low-Level patterns*. Once one of these low-level patterns is detected, a new event with attributes *name*, *activity* and *time* is created (line 3) and emitted on a new event stream for the occurrence of low-level patterns (line 4). A similar change pattern for the hand hygiene activity is shown in Listing 5. Here the discretization explained in Section 4.3 has been applied to the values of the load cell sensor (Line 2) to handle variability in sensor readings.

4.4.3. Change patterns sequence detection

The low-level patterns allow for the detection of the changes as defined by the activity signature, i.e., within the IoT events, between two points in time (corresponding to *one* row in Table 2). An activity is represented by the sequence of these patterns from start to end (corresponding to *all* rows in Table 2). This requires keeping track of events that occurred related to the change patterns in the low-level IoT events. Only considering the occurrence of the change patterns in the

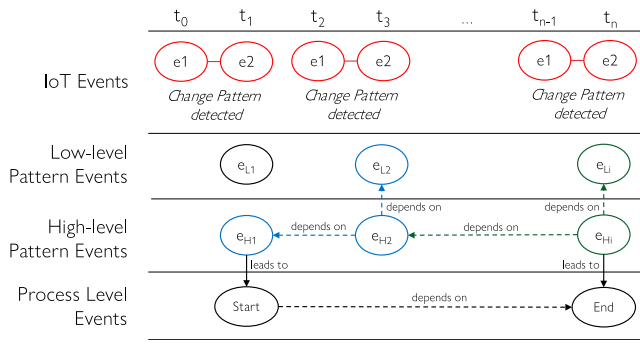


Fig. 8. Event hierarchies and event chains in the activity detection.

low-level IoT events in isolation makes the activity detection prone to errors as the low-level change patterns are likely to occur more often in the IoT event streams, e.g., related to the execution of a different type of activity or not bearing any significance, but still leading to false positives and ambiguities [41].

To make the detection more robust, we introduce a chain and hierarchy of events as special CEP features (cf. Fig. 8): the detection of the first low-level pattern in the IoT events (cf. Table 2, row 1) emits a low-level pattern event e_{L1} . It also emits a high-level pattern event e_{H1} , which leads to a process-level event indicating the start of an activity execution. The detection of the second low-level pattern in the IoT events (cf. Table 2, row 2) leads to the emission of a low-level pattern event e_{L2} , and of a composed high-level pattern event e_{H2} only if e_{H1} occurred before (*depends on*). This is continued by emitting e_{H3} only if e_{H2} occurred before, etc.; until the last change pattern in the IoT events according to the activity signature was detected, and high-level event e_{H6} is emitted only if e_{H5} occurred before. The chain of high-level events e_{H1} to e_{H6} represents the entire activity with e_{H1} and e_{H6} representing the *start* and the *end* of an activity. These process-level events are relevant for process mining. Note that only with detecting the entire chain of high-level pattern events and thus observing the activity execution until the end, we can be sure to also emit the corresponding *Start* event as it has a dependency with the *End* event. Fig. 8 illustrates the different levels of the event hierarchies and dependencies among the events. To implement this sequence detection, we propose to use different event streams containing the IoT events, low-level pattern events, high-level pattern events, and process level events.

Note that in general, this way of activity status tracking only works assuming that there is no concurrent execution of the same type of activity on the same IoT station (cf. Section 4.3). Relaxing this assumption requires introducing an activity identifier with every occurrence of the first (start) low-level pattern of the activity's signature. This identifier then needs to be passed on to the subsequent higher level events. The proposed framework supports this behavior, but CEP queries become slightly more complex and the activity identifier might require additional sensors to distinguish parallel activity executions [61].

Partial sequences detection. To handle variability in the low-level IoT event data (R5) by avoiding overfitting and to better support activity detection at runtime (R4), we decided to also support different strategies of partial instance detections of high-level and low-level event sequences. In contrast to waiting for and expecting the entire sequence of high-level events to occur to inform about events regarding the execution of an activity, we relax the constraints on the activity instance detection in the following ways—trading off completeness, correctness, and latency:

- **Any25LowLevelPatterns:** Any 25% of the low-level patterns have occurred in increasing order in direct succession.

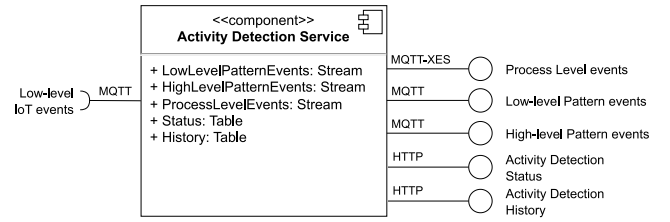


Fig. 9. An activity detection service as one software component.

- **Any50LowLevelPatterns:** Any 50% of the low-level patterns have occurred in increasing order in direct succession.
- **Any75LowLevelPatterns:** Any 75% of the low-level patterns have occurred in increasing order in direct succession.
- **FirstLastLowLevelPattern:** The first and the last low-level pattern have occurred without the appearance of any intermediate out-of-order low-level patterns in between.
- **First50HighLevelPatterns:** The first 50% of high-level patterns have occurred in direct succession.
- **AllHighLevelPatterns:** All high-level patterns have occurred in direct succession (*default*).

4.4.4. Manual adjustments of CEP apps

To further incorporate the knowledge of the domain expert into the activity detection (cf. R3), we leverage the CEP platform's built-in editor (cf. Fig. 11, left) to present the generated CEP app to the expert. Here the low-level patterns in the IoT events can be tweaked, if necessary, and the detection strategy can be chosen. The Streaming SQL language used in the CEP platform is simple to understand [59], especially the patterns in the low-level IoT events are easy to identify and modify, which makes the approach also suitable for non-programmers.

4.4.5. Adding interfaces to CEP apps

The previous paragraphs focused on streams and queries to process events *inside* a CEP app. The CEP app also includes *Event Sources* and *Event Sinks* to non-invasively communicate with outside systems (cf. R1). Here we rely on the capabilities of the underlying CEP platforms that support connectors for standard protocols or means to implement custom connectors. In the proposed framework, the generated CEP app serves as subscriber to receive real-time low-level IoT events using MQTT and as publisher to emit low-level and high-level events on different MQTT topics and process-level events using the MQTT-XES protocol for proper process event semantics [62] to other subscribers according to the publish-subscribe pattern.

Additionally, we extend the generated CEP apps with HTTP-based interfaces acting as service-based event sources and sinks for request-response style communication. This transforms a CEP app into a CEP-based activity detection service with the size of a service corresponding to one type of process activity to be detected as depicted in Fig. 9. The source stream receives service requests from an external client via HTTP, which are treated as new events on a dedicated event stream for outside requests to be processed by dedicated CEP queries. Here we suggest to store and return the current status of the activity execution in and from a state store (table) tracking the activity detection as one service request. Additionally, we propose to introduce a table to store a (history) log of detected activity executions with associated start and end timestamps; and expose this log also via a service-based interface. Fig. 9 shows the interfaces – with exemplary communication protocols in our prototype – of an Activity Detection Service as one component. As these interfaces are static and known at design time, we are able to also generate the corresponding implementations and event streams as event sources and sinks in the CEP app.

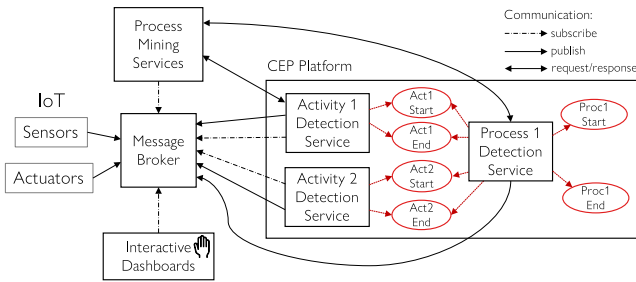


Fig. 10. Service-based software architecture for activity detection.

5. Software architecture of a service-based activity detection system

The generated activity detection services have to be deployed and executed. Fig. 10 shows the software architecture for the online detection of process activities from IoT data streams (cf. R3). This architecture is based on a CEP Platform providing interfaces for automatic service deployment and execution (cf. R2).

5.1. Service deployment & composition on CEP platform

The CEP platform is the runtime environment for the activity detection services. It executes the CEP apps by processing incoming events according to the specified queries and streams to detect relevant event patterns and changes from the IoT sensors. We use the CEP platform *Siddhi* as technological basis of the software architecture [59]. Siddhi features a light-weight runtime engine optimized for high-throughput CEP with a microkernel-like architecture, i.e., it can be extended with CEP apps that use the core functionality of the engine. Siddhi is itself a service providing a Web API for deploying and running new CEP apps at runtime. By using this API we are able to fully automate the generation of the activity detection services and their deployment to the CEP platform (cf. R2).

Although the focus of this work is on the detection of process-level events related to individual activity executions, the activity detection services may also be composed to create *Process Detection Services* (cf. Fig. 10). Assuming the control flow (i.e., sequence of process activities) is known, a new CEP app can be generated that consumes events about the *start* and *end* of activity executions from the individual activity detection services and tracks the processes in a similar way (cf. Section 4), emitting events about process executions. Alternatively, assuming the *Process Signature* (i.e., composition of activity signatures [20]) is known, the process detection service can be generated following the approach described in Section 4. For simplicity, we assume that parallel execution of process instances is not possible and thus, activity-process correlation is straightforward [18].

5.2. Online activity detection and visualization

Upon deployment to the CEP platform, the activity detection services can be activated using the CEP platform's API. When activating a service, the platform establishes the subscriptions to the external IoT event sources and it activates the interfaces to be used by external event and service consumers. Incoming low-level IoT events are processed at runtime based on the CEP queries in the activity detection services. The derived high-level and process-level events related to the activity executions are published via the message-based interfaces, e.g., to be used for process discovery or conformance checking by online process mining services [62,63]. These process mining services may also use the HTTP interfaces of the activity detection services in a request/response manner.

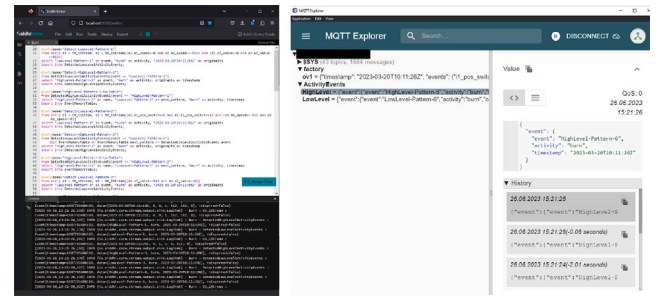


Fig. 11. Demo of activity detection: active activity detection service in editor on the left; low-level IoT and high-level events in MQTT message monitor on the right.

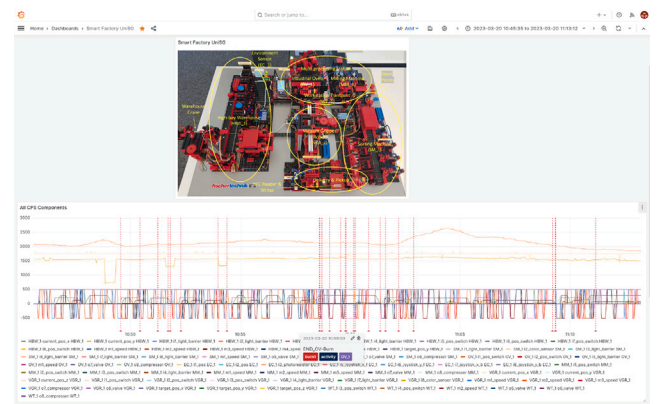


Fig. 12. Interactive dashboard showing detected activity executions in the IoT data.

Fig. 11 shows two screenshots of our prototype: on the left is the web-based editor for CEP apps with the running activity detection service for *Burn*; on the right is the messaging client that monitors the low-level IoT events and the high-level events emitted by the activity detection service. Moreover, we provide live feedback to the user by visualizing the detected process-level events along with the low-level IoT event streams in the interactive dashboards (cf. Fig. 12), which are subscribed to the activity detection topics via the message broker.

6. Implementation

The generator of the CEP-based activity detection services is implemented as a Python package, publicly available under an open-source license [64]. The package is designed to be flexible and extensible, allowing for the selection of included detection methods and multiple possible data sources. The package also allows for developers supplying their own detection methods or data fetchers.

6.1. Usage

The package facilitates the generation of CEP-based services for activity detection, given IoT data and activity annotations as previously described. In addition, signature and change files are generated for the activities. An example of using the package to configure and generate detection services can be seen in Listing 6. The main features of the package include:

- Annotations and IoT data can be provided from various *input sources*; users can provide custom fetchers for required inputs or use included fetchers (cf. Listing 6, line 10 et seq.).
- Generated detection services apps and supporting files are written to a user-specified *output directory* in the local file system (cf. Listing 6, line 7).

```

1 from cepact import (SiddhiConfig, DAGConfig,
2   DetServiceGenerator, LocalIgnoreSensorFetcher,
3   InfluxFetcher, GrafanaFetcher,
4   LocalDiscretizationFetcher,
5   AllHighLevelPattern, First50HighLevelPattern)
6
7 siddhi_conf = SiddhiConfig(mqtt_url = "url",
8   topic_prefix = "SOMEPREFIX",
9   map_sensor_name_data_to_mqtt = {})
10
11 dag_conf = DAGConfig(
12   det_methods = [AllHighLevelPattern(),
13     First50HighLevelPattern()],
14   out_dir = "out", sampling_freq = 1,
15   siddhi_config = siddhi_conf,
16   signature_fetcher = InfluxFetcher(url = "url",
17     auth = "authkey", org = "org",
18     influx_station_bucket_map = {}),
19   annotation_param_fetcher = GrafanaFetcher(url =
20     "url", auth = "authkey"),
21   ignore_sensor_fetcher =
22     LocalIgnoreSensorFetcher("in"),
23   discretization_fetcher =
24     LocalDiscretizationFetcher("in"))
25
26 generator = DetServiceGenerator(dag_conf)
27 generator.run()

```

Listing 6: Package usage example. IoT data obtained from Influx, annotation parameters from Grafana, sensors to ignore, discretization annotations fetched from local file.

- Users can customize the detection service generation using a *configuration* object (cf. Listing 6, line 5).
- *Selection* of included *detection methods* are supported, and *custom* implementations can be added by implementing provided interfaces (cf. Listing 6, line 6).
- The package generates output according to the provided *Siddhi configuration*, ensuring compatibility with downstream systems (cf. Listing 6, line 3).

6.2. Structure

The package is divided into four parts: the *top-level* modules, the *input_processor* package, the *representations* package and the *output_producer* package (cf. Fig. 13).

Top-level modules. The top-level modules are responsible for handling the control flow, delegation of work, and forwarding of objects. The primary class is the *DetAppGenerator*, which follows the mediator pattern, controlling and coordinating between the packages [65]. Furthermore, the *DAGConf* class represents a parameter object class for configuration of the detection app generation process.

Input processor. The input processor package is responsible for retrieving data from the sources and returning the relevant data in the form of objects and object structures from the representations package. Inside this package, the *InputProcessor* class acts as the mediator, while we use the strategy pattern to allow flexible choice of data sources [65]. The package expects objects of four different abstract data fetcher classes during the configuration. A range of concrete implementations of those classes is provided with the package, but users are free to provide their own, thus enabling varied data sources and extensibility.

Representations. The classes in this package represent core data structures for the service generation and associated functionality. The *Activity* class itself acts mainly as a dataclass, encapsulating annotations, signatures, and sensor exclusions for a specific activity. *AnnotationParams* represents parameters associated with annotations, including identifiers, activity names, start and end times, and related stations.

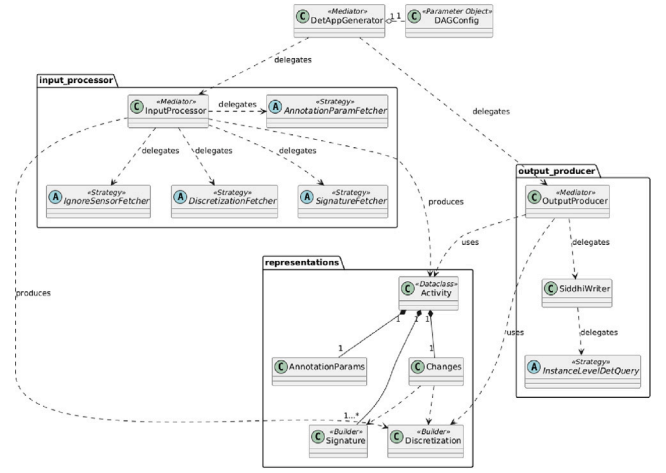


Fig. 13. Internal package structure as a UML class diagram.

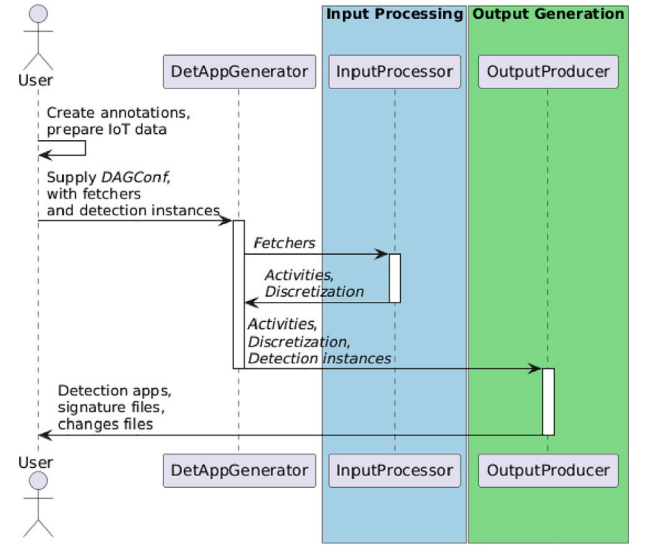


Fig. 14. Interactions in the service generation process.

The *Discretization* and *Signature* classes follow the builder design pattern [65], facilitating the structured construction of their instances. This is especially important when users might want to implement their own *DiscretizationFetcher* or *SignatureFetcher*, thus having to construct these objects. *Changes* represents the aforementioned changes happening in the discretized IoT data.

Output producer. The output producer receives both a set of activities and discretizations. For each activity, it creates an activity detection service using the *SiddhiWriter*, where we employ the strategy pattern to allow for flexible selection of detection methods and custom detection schemes. For each of the services, users may specify any number of partial detection methods to be included and also provide custom implementations as subclasses of the abstract class *InstanceLevelDetQuery*. We support each of the instance level detection query methods described in Section 4.4 for partial sequence detection.

6.3. Behavior

The communication during service generation is divided into three phases (cf. Fig. 14). (1) The user creates the annotations and points to the source of IoT data used for the service generation as specified

by the data fetchers. The user can set additional parameters for the through the *DAGConf* object. After this, the *DetAppGenerator* coordinates between the input processor and the output producer. (2) The *InputProcessor* delegates work to the different concrete data fetchers using the strategy pattern to compose the *Activity* objects. Finally, the activity annotation and IoT data are returned to the *DetAppGenerator*. (3) The *DetAppGenerator* forwards the data obtained from the *InputProcessor* to the *OutputProducer*. Here, the creation of the signature and changes files is delegated to the *Signature* and *Changes* objects of the activities. Based on the *Changes* object of each activity, the *SiddhiWriter* is then instructed to create the services.

7. Evaluation

In this section, we evaluate the generated activity detection services. We describe the overall experiments and datasets, which are based on the IoT scenarios described in Section 2.2. We present the results for each scenario, starting with an overview of all metrics and then focusing on specific metrics and partial sequence detections.

7.1. Setup

For each of these scenarios, we recorded datasets during the execution of the processes described in Section 2.2. All of the datasets contain the low-level IoT time series data and corresponding ground truth annotations, i.e., files describing when which activity has actually been executed. For the smart manufacturing scenario, the ground truth was created automatically by the BPM system used to orchestrate the smart factory [21]. For the smart healthcare scenario, the ground truth was created manually by monitoring and tracking activities during the execution of blood donation process instances. The datasets for the service generation and evaluation and the scripts used to generate and evaluate the services are publicly available [64].

In one of the datasets for each process, domain experts have annotated the IoT data with the activity prototypes. We use the package described in Section 6 to create the activity detection services. We execute the services and replay the data from the datasets as event streams via MQTT using the original timestamps contained in the events. The resulting detections are compared with the ground truths, using the tool and metrics presented in [66]. An important distinction to note is between frame-based (also called *2SET*) and event-based (also called *event analysis/EA*) metrics. *2SET* metrics evaluate the correctness based on the sampling frames of the IoT data and whether each frame has been correctly classified by the detection method, while the *EA* metrics categorize entire activity executions as being correctly detected, thus being one level of abstraction above the IoT sampling frames [67].

7.2. Smart manufacturing scenario

7.2.1. Dataset

The input IoT event log was recorded from our small-scale smart factory (cf. Fig. 1) executing the storage and production processes over 27.63 min. 13 distinct activities have been annotated from six production stations; according to our assumptions (cf. Section 2.2), none of the activities span more than one station. The expert annotations include discretizations of 14 originally continuous sensors, and 6 sensors have been marked to be ignored. The IoT event logs that have been replayed to test the detection quality span 178.44 min; 36 process instances were executed, which contained 395 activity instances. All IoT event logs were sampled at a frequency of 0.5 Hz.

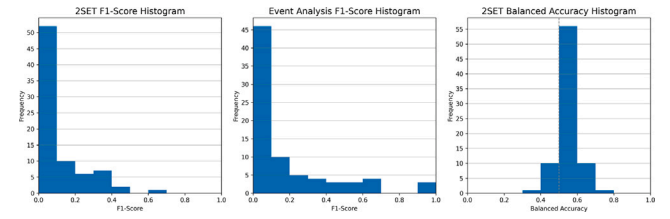


Fig. 15. Histograms for some metrics of all detection method/activity combinations in the factory scenario; vertical dotted line marks 0.5 balanced accuracy.

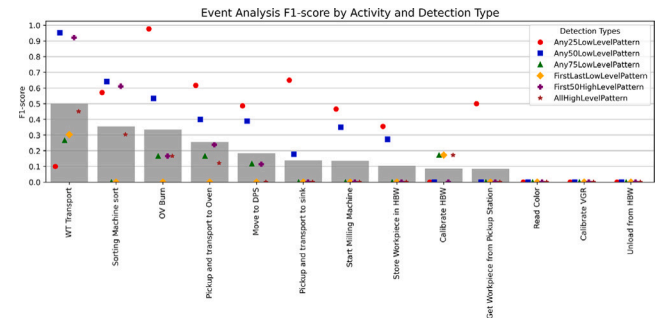


Fig. 16. F1 scores for the annotated activities in the factory scenario, averaged over the results from the different detection methods, value for each detection method shown as separate markers.

7.2.2. Results

The overall results, separated by detection method, for the smart manufacturing processes can be seen in Table 3. In general, the detection results are relatively low with the F1 score on the event level being at 0.3686. When taking a closer look at the distribution of some metrics over all activities and all detection methods (cf. Fig. 15), it becomes clear that this is because the detection works well in some cases, moderately well in a few cases, and rather poorly in many cases. To better understand why the detection works well for some activities/detection methods but not others, we take a closer look at the EA F1 scores for the activities we try to detect, as shown in Fig. 16. We chose the EA F1 score due to the importance of correct detections on the event level: we care mostly whether the right activity instances are detected roughly at the correct time, and to some extent implied by that, in the correct order. The specific correctness of duration is also important, but secondary to the mere detection of an activity (e.g., for conformance checking only analyzing the correct order of activity executions w.r.t. a normative model). The partial sequence detection method performing best for the majority of the metrics is *Any25LowLevelPatterns*, indicating that there is a high variability among the occurrences of the activity instances, i.e., matching more of the low-level patterns decreases the detection performance. One problem that affects the activities with poor detection performance is that the activity signatures and thus change patterns for different executions are indeed highly variable, e.g., in the *Unload from HBW* activity retrieving storage containers from the different slots in the warehouse. Furthermore, there are activities where the change patterns are highly similar to the change patterns of different activities due to missing sensors, leading to incorrect detections (e.g., *Pickup and transport to Oven*, *Pickup and transport to sink*). The current detection services have no way of resolving this *ambiguity*, leading to low precision.

7.3. Smart healthcare scenario

7.3.1. Dataset

The IoT log used as an input for the service generation has a total duration of 5.82 min. The expert annotation consist of 18 activities

Table 3

Detection results in smart factory scenario, by detection method (Abbr.: *Det Meth*: Detection Methods, *Prec*: Precision, *Bal Acc*: Balanced Accuracy, *NDLD*: Normalized Damerau-Levenshtein-Distance, *CC*: Cross-Correlation; detection methods in Section 4.4).

Det Meth	Frame (2SET) metrics				Event metrics			Other metrics	
	Prec	Recall	F1	Bal Acc	Prec	Recall	F1	NDLD	CC
Any25LL	0.1677	0.1969	0.181	0.5677	0.3825	0.359	0.3686	0.664	0.5446
Any50LL	0.2832	0.1553	0.1878	0.5607	0.771	0.2005	0.3129	0.7748	0.4544
Any75LL	0.3785	0.0369	0.0657	0.5166	0.6746	0.0464	0.0845	0.9439	0.1846
FirstLastLL	0.2163	0.0453	0.0663	0.5068	0.2985	0.0303	0.0549	0.9524	0.1049
First50HL	0.3901	0.0786	0.1146	0.5294	0.8207	0.1087	0.1907	0.8835	0.2361
AllHL	0.1088	0.03	0.042	0.3409	0.4889	0.0243	0.0461	0.6363	0.0742

Table 4

Detection results in healthcare scenario, by detection method (Abbr.: *Det Meth*: Detection Methods, *Prec*: Precision, *Bal Acc*: Balanced Accuracy, *NDLD*: Normalized Damerau-Levenshtein-Distance, *CC*: Cross-Correlation; detection methods in Section 4.4).

Det Meth	Frame (2SET) metrics				Event metrics			Other metrics	
	Prec	Recall	F1	Bal Acc	Prec	Recall	F1	NDLD	CC
Any25LL	0.1289	0.1826	0.1483	0.5619	0.0763	0.1875	0.1065	0.8723	0.2849
Any50LL	0.1286	0.1836	0.1448	0.5597	0.2081	0.3993	0.27	0.7682	0.4748
Any75LL	0.0710	0.0539	0.0585	0.5085	0.2886	0.2118	0.2401	0.8094	0.3179
FirstLastLL	0.1019	0.1456	0.1165	0.5435	0.2189	0.3889	0.274	0.7783	0.5987
First50HL	0.1412	0.1266	0.1272	0.542	0.2831	0.2708	0.2719	0.7523	0.3338
AllHL	0.049	0.0325	0.0373	0.5011	0.2783	0.1528	0.1942	0.8472	0.2408

from the blood donation process. Three of these activities are cross-station, meaning they happen across and their detection relies on more than one station. Seven of the sensors have discretization annotations, and three sensors are taken out of consideration in general, with some other sensors being ignored only for specific activities. The data for replay and evaluation consists of IoT logs with a total duration of 65.31 min. All IoT event logs were sampled at a frequency of 2 Hz. The ground truth data contains 188 activity instances from a total of 16 process instances. These processes have been executed in our IoT lab (cf. Fig. 3) by 17 students of the Medical Master Programme at our university.

7.3.2. Results

An overview of the results can be seen in Table 4, showing the main metrics for the various detection types. In the healthcare scenario, the partial sequence detection methods relying only on certain percentage of the low level patterns or the first and last low level pattern show the best values for most metrics. A closer look at the distribution of values for the metrics frame-based (2SET), EA F1 scores and the 2SET accuracy is provided in Fig. 17. These distributions reveal that the detection works rather poorly for many, but relatively well for some of the activities. Fig. 18 shows the EA F1 score for all activities, averaged over all detection methods but also separately indicated for each detection method. The activities which have higher F1 scores are mostly among those with clearer IoT semantics: the starting and stopping of the blood drawing machine and the check-ins/check-outs are clearly identifiable by discrete button-presses and NFC scans. A notable activity is the hand hygiene. Here the discretization provides a clear signal for when the hand hygiene is executed: the weight measurements of the scale with the disinfectant increases sharply when the healthcare worker presses on the bottle (cf. Fig. 7). One reason for the low detection performance can again be identified as *ambiguity* [41]: the activities *Monitor patient*, *Apply tourniquet*, *Remove tourniquet*, and *Insert needle* all rely on the same set of sensors and their change patterns are almost indistinguishable from each other (same for other sets of activities). Another reason for the low detection performance is a high degree of variability among executions: activities such as *Take out samples* or *Monitor patient* have been executed in different ways by students.

8. Discussion

With the evaluation results we have shown the feasibility of our approach based on a proof-of-concept implementation that is able to

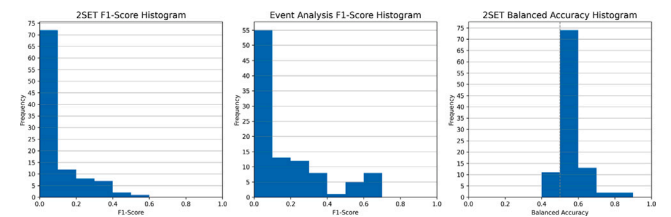


Fig. 17. Histograms for some metrics of all detection method/activity combinations in the healthcare scenario; vertical dotted line marks 0.5 balanced accuracy.

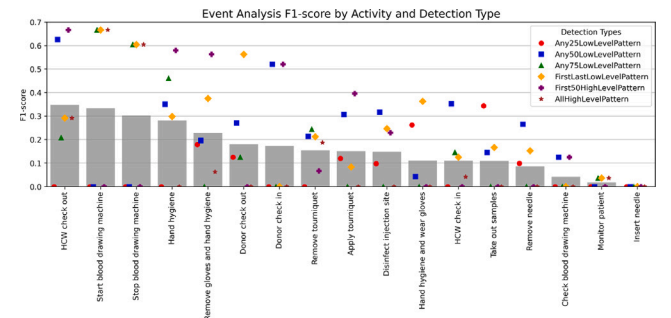


Fig. 18. F1 scores for the annotated activities in the healthcare scenario, averaged over the results from the different detection methods, value for each detection method shown as separate markers.

detect activities from IoT data streams at runtime (cf. R4). We were also able to demonstrate its generalizability by applying it to IoT data streams from two different application domains. In this section we discuss the fulfillment of the requirements R1–R5 considering the results of the evaluation, potential improvements and limitations of the framework.

8.1. Requirement R1: Non-invasiveness

The non-invasive integration with existing IoT environments is achieved using the capabilities of modern CEP platforms that provide connectors to external systems either relying on standard protocols or on simplified custom connectors. The chosen *Siddhi* platform is a light-weight Java-based service that integrates seamlessly with other services in the software system [59]. The detection service generation

package can be used in a variety of environments, through the possibility of choosing from a range of data sources or even supplying user-defined data fetchers. Nevertheless, with **R1** we depend on the interfaces and data the specific IoT devices provide. The framework relies on the quality of the low-level IoT data, i.e., activity detection might not work reliably or even lead to ambiguities if the provided data is of poor quality or sensors are missing [41]. Here, additional sensors can help to increase data quality; and semantic knowledge about IoT components can be used to introduce enriched events and more advanced CEP patterns into the activity detection services. The pattern-based detection in the CEP engine may not work with more complex data (e.g., from cameras as sensors [61]) and require a pre-processing step (e.g., using ML) executed by a dedicated software component to derive more abstract events to be integrated into the CEP queries. The proposed framework and software architecture are very flexible and extensible as they rely on services and standardized interfaces and protocols to communicate with external systems. The activity detection components are not required to be CEP-based or deployed on the CEP platform. As long as the MQTT interface for subscribing to IoT event messages, MQTT-XES/XES interfaces for publishing and persisting process-level events, and HTTP interfaces for retrieving process-level events are implemented and provided by the activity detection component (cf. Fig. 9), it can be easily plugged in as a standalone (micro-)service (e.g., applying ML for activity detection [6]) into the software architecture.

8.2. Requirement R2: No-code services

The activity detection services are generated from the activity signatures and automatically deployed to the CEP platform (no-code and no-ops). The domain expert only has to identify the activity prototype in the low-level IoT events and annotate sensors regarding their relevance and discretizations as first steps in the interactive dashboards [20]. The rest is automated via the service generation pipeline and the CEP platform's API. Extensibility is supported through the CEP platform, i.e., detection services for new types of activities can be added and composed at runtime via its API. The activity detection services generated in our evaluation scenarios were in the order of a few hundred lines of code in Streaming SQL. The Java-based CEP platform was able to handle multiple instances of these services consuming IoT events and producing process-level events well. However, with increasing complexity of an activity (e.g., regarding the number of changes in the IoT data) the CEP apps also grow in complexity, which needs to be further investigated for larger scale manufacturing settings and more complex production machines. Moreover, we currently follow a one-service-per-activity approach, which may also influence the performance of CEP platform negatively when a large number of different types of activities should be detected, i.e., a high number of services are running concurrently on the CEP platform.

8.3. Requirement R3: Human-in-the-loop

Although the major focus of the framework and software architecture is on automation, the domain expert is kept in the loop at several important points. Besides annotating the IoT data and providing sensor configurations, the domain expert is able to inspect and adjust the parameters in the generated CEP patterns before deployment (cf. Fig. 11, left), which goes beyond the pure explainability of event correlations as shown in [40]. In contrast to classical ML models, the CEP apps show the patterns applied in the detection of individual activities, which fosters *explainability*. However, with increasing complexity of an activity (e.g., regarding changes in the IoT data) the CEP apps also grow in complexity and difficulty to understand the number of change patterns and individual patterns. The same holds for the visualizations of detected activities in the interactive dashboards. Here, the domain

expert receives instant feedback about detections via special annotations, contextualized in the IoT event streams. These can be used as basis to provide feedback about process conformance and compliance, e.g., in the form of overlays combined with the normative process model [22,68]. However, with increasing complexity of IoT data and parallel instances of activity detection services, the visualizations might also become more difficult to follow. The used dashboard tool Grafana provides many means for overview, filtering, zooming in, details on demand, and customization of dashboard panels to counteract complexity. In general, we expect the approach to be usable in IoT domains with similar or less sensors than in the presented examples. A comprehensive user study is required to confirm this assumption. Finally, the output of the activity detections is provided as process-level events conforming to the XES standard, which facilitates the subsequent loading and (visual) expert analysis of the event data using common process mining tools.

8.4. Requirement R4: Activity detection at runtime

CEP platforms are designed to work with high volumes of event streams at runtime [58,59]. By deploying and executing the generated activity detection services on the light-weight CEP platform Siddhi, we are able to process event streams from IoT sensors close to the edge in near real-time and emit higher levels events for further online process analysis. The experiments replaying the IoT datasets as described in Section 7 and processing the events in near real-time on the CEP platform have shown a high degree of performance, which demonstrates that CEP technology is very well suited for activity detection in the scenarios and IoT setups involving tens to hundreds of sensors emitting new events several times per second. We did not experience any considerable load on a state-of-the-art personal computer, even when running up to 16 detection services at once, consuming and processing up to 1200 IoT events per minute. More extensive load and performance tests with higher amounts of sensor data from real-world production machines remain subject to future work.

In our experiments, we evaluate the CEP platform's capabilities of processing incoming events and deriving higher level events at runtime. Note that this event abstraction is based on the specified patterns in the respective CEP apps. These patterns are dependent on the initial IoT data associated with an activity signature that is used for generating the activity detection services. Here, we observe an influence regarding the sampling frequency of this data when evaluating the runtime detection as the changes to detect a process-level event have to be encoded in the CEP app, i.e., the sampling frequency for the IoT data in the activity signature should be high enough to register all relevant changes. In turn, the sampling frequency of the unknown IoT data streams to be classified at runtime has to match or be higher than the frequency of the activity signature to also show the relevant changes that need to be detected. The detection of the high level patterns in *near real-time* strongly depends on the publishing frequency of new events from the IoT sensors and their ingestion into the CEP platform. Since in IoT we are naturally faced with a distributed system in line with the software architecture proposed in Section 5, we expect a certain latency here due to network communication and message processing by the message broker. A more comprehensive profiling of the proposed system's performance is needed to evaluate its applicability in environments with harder real-time constraints. This also involves evaluating the processing of the IoT events *inside* the CEP platform to derive the higher level events according to the specified patterns [69], which adds to the overall end-to-end latency to emit a process-level event from the raw IoT events and thus influences real-time guarantees and related KPIs.

With supporting different strategies of detecting activities based on the partial occurrence of change pattern sequences, we are able to detect executions of activities before their completion (e.g. considering only the start pattern or 25% of the first low-level patterns), which allows us to provide more immediate feedback without having to wait until the end of an activity. Here we trade off completeness and

accuracy – different types of activities may have, for example, the same start pattern in the sensor data – for latency, which is also more difficult to achieve with pure ML-based approaches that do not support partial detections or activity detection at runtime due to long durations of inference as discussed in Section 3. In addition to partial activity detections, considering the appearance of sequences/chains of low-level event patterns of an activity and abstracting these to composite high-level events allows us to vary the granularity of detected activities [34]. By introducing more levels into the event hierarchy, we are able to make the detection more fine grained (e.g., considering also sub-activities) following the principles described in Section 4.4. The *Burn* activity executed by the Oven station as depicted in Fig. 6 for example consists of the sub-activities: open door, insert sled, close door, bake, open door, eject sled, close door. All these can also be detected from the IoT data based on the low-level change pattern and their combinations. Moreover, these event hierarchies and compositions also allow us to implement a more distributed event processing approach achieving high levels of scalability with low-level event processing on different edge nodes and the ingestion and processing of the emitted higher level events on more centralized nodes. These investigations remain subject to future work.

8.5. Requirement R5: Robustness

The evaluation results presented in Section 7 show that, in principle, our proof-of-concept implementation works well in IoT scenarios with low variations in sensor data for the same type of activity. However, there might be variations due to external factors or different execution parameters that may have a negative influence on the quality of the detections. Detecting the sequence of change patterns as described in Section 4 is robust regarding *different execution durations* of an activity as time is not factored in, similar to *Dynamic Time Warping* [70]. Note that as stated above, sampling rates of the low-level IoT data matter when deriving the changes between two consecutive timestamps, i.e., changes might not be registered if the sampling rate is too low. In turn, the sampling rate of the low-level IoT events used for activity detection at runtime also has to reflect the changes encoded in the CEP apps. Introducing sensor discretizations and sensor value ranges before deriving the change patterns, and marking sensors as *irrelevant*, increases the robustness of the pattern detection against variations in the values of specific sensors. By applying these specific means and configurations as well as directly modifying the generated CEP apps, the domain expert is able to select and influence the phenomena against which the activity detection should be robust. While related ML-based approaches might show better detection qualities and tolerance against variations through specially trained models (cf. Section 3), our proposed pattern-based approach is rather generic and light-weight, and it can be applied to arbitrary IoT data streams. In our experiments, we did not experience any major distribution shifts regarding sensor values and detection qualities with repeated executions of the same processes and activities. In this context, apart from environmental sensors that are subject to stronger variations, but that are usually irrelevant for the execution of an activity, long-term variations and drifts in the sensor data underlying an activity execution [71] are usually indications of hardware-related problems or changes in how the activity is performed. This would require selecting a new prototypical activity execution and generating a new version of the corresponding activity detection service.

Activities of the same type showing *different change pattern sequences* might lead to incorrect activity detections. An example is the activity *Unload* for a workpiece in the warehouse (cf. Section 2.2). Here the motor movements and involved sensors depend on the location of the workpiece resulting in different change pattern sequences as the position from which to unload a workpiece is determined at runtime based on its type and availability. This issue could be mitigated, e.g., by making the *Unload* activity more fine grained, also specifying the

position in the warehouse and treating each position as a distinct activity (e.g., *Unload from Slot 1*). Another possibility of counteracting these types of variations would be to follow an underfitting approach, i.e., to let the domain expert select a suitable activity prototype that involves most of the change pattern common for all executions of one activity type and to only adapt a partial sequence detection strategy as described in Section 4.4.3. Another potential improvement to deal with variability is to consider multiple signatures per activity, which could enable a more nuanced creation of the changes, e.g., by only considering changes that happen in at least a certain percentage of the annotations. Integrating more advanced patterns for CEP (e.g., related to time-based or count-based windows and value ranges) would also increase robustness, but requires more domain knowledge and is subject to future work.

Apart from variations in sensor data, we experienced *ambiguity* as another source influencing detection quality. The change patterns of an activity might not uniquely identify one activity, or changes in IoT data when no activity is executed might be confused to be an activity execution. One possible mitigation is to put focus on changes within an activity signature that differentiate it from changes that happen outside of that signature. This could be done automatically, for example by identifying the most relevant sensors and change patterns for certain activities. A semi-automated approach could also be considered here, where derived differentiations could be modified or rejected by an expert. Another possibility is to rely more on the temporal interplay of detected activities for the disambiguation in real-time. This would require further domain knowledge, which could come from normative process models or additional expert inputs.

Another issue that arose is that of lacking *data quality*: due to the reliance on only one activity signature and one set of activity annotations, data quality issues in either of these categories are exacerbated and lead to subpar detection results. A high enough sampling frequency can help avoid missing sensor values, while a potential inclusion of shift detections (e.g., an increase of a certain value by more than 50% between samples) in the change patterns can help lessen the reliance on normalized data. Further work on the methodology of obtaining expert annotations (visualizations, tools, user studies) could help to improve the quality of the annotations. Finally, we also determined some time-related discrepancies when aligning the detected activity executions with the ground truth recorded by the BPM system orchestrating the processes in the smart factory. Here, each automated activity is implemented as an invocation of a Web service controlling the individual IoT hardware [21]. While the BPM system will log the invocation of the Web service as start of an activity, the actual manifestation of its execution in the IoT sensors and actuators as recognized by our detection services might be delayed for several reasons (e.g., network communication, processing of data, other software operations). These potential delays must be taken into account when modeling and implementing the actual activities that should be detected, e.g., by considering *IoT activities* as special types of activities that start and end only with changes in their respective sensors and actuators [20].

9. Conclusion and future work

Bringing the worlds of BPM and IoT closer together promises to be mutually beneficial for both fields [4]. The novel data sources – sensors and actuators – introduced by the IoT may open new opportunities to foster process analysis and decision making at runtime, even in the absence of a BPM system. However, the data streams emitted from these IoT devices might be too fine-grained and low-level to be used for process mining right away [16], requiring additional steps of event abstraction. We propose a data-driven framework to generate services that are capable of detecting process activity executions from IoT data streams at runtime. Having a focus on non-invasiveness and automation, the framework relies on the domain expert to only provide basic data regarding the IoT setup and to identify a representative activity

execution manifested in the available IoT data: the *Activity Signature*. From this signature, we generate a detection service for the activity based on the sequence of change patterns translated into a CEP app. We support partial activity detections and event hierarchies to enable fast decision making and distributed event processing at runtime. Along with the framework, we propose a software architecture based on a flexible, extensible and light-weight CEP platform for automatic deployment, composition and execution of the activity detection services. Compared to ML-based approaches, the CEP apps make the specific detection patterns explicit, explainable and modifiable, facilitating the human-in-the-loop. Evaluations of the framework and architecture in the domains of smart manufacturing and smart healthcare have shown the feasibility of our proof-of-concept implementation and acceptable detection results when there is little to no variation in the IoT data associated with the execution of the same type of activity. In our discussions, we identified several ways to potentially improve these results by making the activity detections more robust against variations in the IoT data.

In future work we will investigate the use of semantic knowledge about the IoT components and activities to extend the CEP patterns for increasing the robustness of the activity detection and for automating the manual step of identifying activity prototypes. Furthermore, we will extend the concept of activity signatures to incorporate more than one instance and a certain degree of variability to make the detection more robust. We will apply the framework in larger scale smart manufacturing settings to investigate its scalability and conduct a more comprehensive profiling of the architecture to evaluate the activity detection at runtime with respect to potential real-time behavior. Finally, we will integrate online process mining tools (e.g., Beamline [63]) into the architecture to enable process analytics, especially process discovery and conformance checking, at runtime.

CRedit authorship contribution statement

Ronny Seiger: Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Resources, Methodology, Investigation, Data curation, Conceptualization. **Aaron F. Kurz:** Writing – review & editing, Validation, Software, Resources, Data curation. **Marco Franceschetti:** Writing – review & editing, Writing – original draft, Validation, Methodology, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Marco Franceschetti and Aaron F. Kurz report financial support was provided by Swiss National Science Foundation. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has received funding from the Swiss National Science Foundation under Grant No. IZSTZ0_208497 (*ProAmbition* project). This work has been supported by the Internet of Processes and Things (IoPT) community¹ through discussions, datasets, and feedback.

Data availability

The dataset and prototype implementation are publicly available in an open repository, which is cited in the paper.

References

- [1] M. Bauer, N. Bui, J. De Loof, C. Magerkurth, et al., IoT reference model, in: *Enabling Things to Talk: Designing IoT Solution with the IoT Architectural Reference Model*, Springer Berlin Heidelberg, 2013, pp. 113–162.
- [2] H.H. Beyel, O. Makke, M. Pourbafrani, O. Gusikhin, W.M. van der Aalst, Analyzing data streams from cyber-physical-systems: A case study, *SN Comput. Sci.* 5 (6) (2024) 706.
- [3] B. Weber, A. Abbad-Andaloussi, M. Franceschetti, R. Seiger, et al., Leveraging digital trace data to investigate and support human-centered work processes, in: *Int. Conf. on Evaluation of Novel Approaches To Software Engineering*, Springer, 2023, pp. 1–23.
- [4] C. Janiesch, A. Koschmider, M. Mecella, B. Weber, A. Burattin, C. Di Ciccio, et al., The internet of things meets business process management: a manifesto, *IEEE Syst. Man Cybern. Mag.* 6 (4) (2020) 34–44.
- [5] J. Cooper, A. James, Challenges for database management in the internet of things, *IET Tech. Rev.* 26 (5) (2009) 320–329.
- [6] E. Garcia-Ceja, R.F. Brena, J.C. Carrasco-Jiménez, L. Garrido, Long-term activity recognition from wristwatch accelerometer data, *Sens.* 14 (12) (2014) 22500–22524.
- [7] D. Janssen, F. Mannhardt, A. Koschmider, S.J. van Zelst, Process model discovery from sensor event data, in: *ICPM 2020*, Springer, 2020, pp. 69–81.
- [8] G. Di Federico, A. Burattin, Vamos: event abstraction via motifs search, in: *Int. Conf. on Business Process Management*, Springer, 2022, pp. 101–112.
- [9] A.P. Dake, Smart environment integrating various business applications based on Internet of Things (IoT), *J. Netw. Technol.* 11 (1) (2020) 1–8.
- [10] J. Mangler, R. Seiger, J.-V. Benzin, J. Grüger, Y. Kirikkayis, et al., From Internet of Things data to business processes: Challenges and a framework, 2024, arXiv: 2405.08528.
- [11] R. Seiger, M. Franceschetti, B. Weber, Data-driven generation of services for IoT-based online activity detection, in: *International Conference on Service-Oriented Computing*, Springer, 2023, pp. 186–194.
- [12] K. Peffers, T. Tuunanen, M.A. Rothenberger, S. Chatterjee, A design science research methodology for information systems research, *J. Manage. Inf. Syst.* 24 (3) (2007) 45–77.
- [13] W. Van Der Aalst, A. Adriansyah, A.K.A. De Medeiros, F. Arcieri, T. Baier, T. Blickle, et al., Process mining manifesto, in: *BPM 2011 International Workshops*, Part I, Vol. 9, Springer, 2012, pp. 169–194.
- [14] M. Dumas, M. La Rosa, J. Mendling, H.A. Reijers, *Fundamentals of Business Process Management*, Springer, 2018.
- [15] Y. Bertrand, J. De Weerd, E. Serral, An expert-validated bridging model for IoT process mining, *Bus. Inf. Syst. Eng.* (2024) 1–20.
- [16] I. Beerepoot, C. Di Ciccio, H.A. Reijers, S. Rinderle-Ma, W. Bandara, et al., The biggest business process management problems to solve before we die, *Comput. Ind.* 146 (2023) 103837.
- [17] P. Soffer, A. Hinze, A. Koschmider, H. Ziekow, C. Di Ciccio, et al., From event streams to process models and back: Challenges and opportunities, *Inf. Syst.* 81 (2019) 181–200.
- [18] K. Diba, K. Batoulis, M. Weidlich, M. Weske, Extraction, correlation, and abstraction of event data for process mining, *Data Min. Knowl. Discov.* 10 (3) (2020).
- [19] L. Malburg, R. Seiger, R. Bergmann, B. Weber, Using physical factory simulation models for business process management research, in: *BPM Workshops*, Springer, 2020, pp. 95–107.
- [20] R. Seiger, M. Franceschetti, B. Weber, An interactive method for detection of process activity executions from IoT data, *Futur. Internet* 15 (2) (2023).
- [21] R. Seiger, L. Malburg, B. Weber, R. Bergmann, Integrating process management and event processing in smart factories: A systems architecture and use cases, *J. Manuf. Syst.* 63 (2022) 575–592.
- [22] M. Franceschetti, R. Seiger, M.J.G. González, E. Garcia-Ceja, L.A.R. Flores, et al., ProAmbition: Online process conformance checking with ambiguities driven by the Internet of Things, in: *CAISE Research Projects Exhibition*, 2023, pp. 52–59.
- [23] R. Seiger, F. Zerbato, A. Burattin, L. García-Bañuelos, B. Weber, Towards IoT-driven process event log generation for conformance checking in smart factories, in: *24th EDOC Workshop, EDOCW*, 2020, pp. 20–26.
- [24] C. Chang, S.N. Srirama, R. Buyya, Mobile cloud business process management system for the internet of things: a survey, *ACM Comput. Surv.* 49 (4) (2016) 1–42.
- [25] D. Raggett, The web of things: Challenges and opportunities, *Comput.* 48 (5) (2015) 26–32.
- [26] H.A.A. Chaudhary, I. Guevara, J. John, A. Singh, T. Margaria, D. Pesch, Low-code Internet of Things application development for edge analytics, in: *Internet of Things. IoT Through a Multi-Disciplinary Perspective*, Springer, 2022, pp. 293–312.
- [27] X. Wu, L. Xiao, Y. Sun, J. Zhang, T. Ma, L. He, A survey of human-in-the-loop for machine learning, *Future Gener. Comput. Syst.* 135 (2022) 364–381.
- [28] E.A. Lee, Cyber physical systems: Design challenges, in: *11th Int. Symposium on Object and Component-Oriented Real-Time Distributed Computing*, IEEE, 2008, pp. 363–369.

¹ <https://zenodo.org/communities/iopot/about>.

- [29] D. Corral-Plaza, I. Medina-Bulo, G. Ortiz, J. Boubeta-Puig, A stream processing architecture for heterogeneous data sources in the Internet of Things, *Comput. Stand. Interfaces* 70 (2020) 103426.
- [30] M.O. Gökalp, A. Kocyiğit, P.E. Eren, A visual programming framework for distributed internet of things centric complex event processing, *Comput. Electr. Eng.* 74 (2019) 581–604.
- [31] B.R. Barricelli, S. Valtolina, A visual language and interactive system for end-user development of internet of things ecosystems, *J. Vis. Lang. Comput.* 40 (2017) 1–19.
- [32] M. Backmann, A. Baumgrass, N. Herzberg, A. Meyer, M. Weske, Model-driven event query generation for business process monitoring, in: *ICSOC 2013 Workshops*, in: LNCS, vol. 8377, Springer, 2013, pp. 406–418.
- [33] R. Mousheimish, Y. Taher, K. Zeitouni, Autocep: automatic learning of predictive rules for complex event processing, in: *ICSOC 2016*, Springer, pp. 586–593.
- [34] F. Zerbatto, R. Seiger, G. Di Federico, A. Burattin, et al., Granularity in process mining: Can we fix it? in: *Int. Workshop on BPM Problems To Solve before We Die*, 2021, pp. 40–44.
- [35] G. Di Federico, A. Burattin, CvAMoS—event abstraction using contextual information, *Futur. Internet* 15 (3) (2023).
- [36] S.J. van Zelst, F. Mannhardt, M. de Leoni, A. Koschmider, Event abstraction in process mining: literature review and taxonomy, *Granul. Comput.* 6 (3) (2021) 719–736.
- [37] F. Folino, M. Guarascio, L. Pontieri, Mining predictive process models out of low-level multidimensional logs, in: *Int. Conf. on Advanced Information Systems Engineering*, Springer, 2014, pp. 533–547.
- [38] N. Tax, N. Sidorova, R. Haakma, W.M. van der Aalst, Event abstraction for process mining using supervised learning techniques, in: *SAI Intelligent Systems Conf.*, Springer, 2016, pp. 251–269.
- [39] D. Janssen, F. Mannhardt, A. Koschmider, S.J. van Zelst, Process model discovery from sensor event data, in: *Process Mining Workshops*, 2021, pp. 69–81.
- [40] B. Sedlak, V.C. Pujol, P.K. Donta, S. Dustdar, Equilibrium in the computing continuum through active inference, *Future Gener. Comput. Syst.* 160 (2024) 92–108.
- [41] M. Franceschetti, R. Seiger, H.A. López, A. Burattin, L. García-Bañuelos, et al., A characterisation of ambiguity in BPM, in: *International Conference on Conceptual Modeling*, Springer, 2023, pp. 277–295.
- [42] A. Koschmider, D. Janssen, F. Mannhardt, Framework for process discovery from sensor data, in: *EMISA*, 2020, pp. 32–38.
- [43] G. Di Federico, A. Burattin, CvAMoS—event abstraction using contextual information, *Futur. Internet* 15 (3) (2023) 113.
- [44] A. Senderovich, A. Rogge-Solti, A. Gal, J. Mendling, A. Mandelbaum, The ROAD from sensor data to process instances via interaction mining, in: *International Conference on Advanced Information Systems Engineering*, CAISE, in: LNCS, vol. 9097, Springer, 2016, pp. 257–273.
- [45] F. Mannhardt, M. de Leoni, H.A. Reijers, W.M.P. van der Aalst, P.J. Toussaint, From low-level events to activities - A pattern-based approach, in: *International Conference on Business Process Management*, BPM, in: LNCS, vol. 9850, Springer, Cham, 2016, pp. 125–141.
- [46] J. Wanner, L.-V. Herm, C. Janiesch, Countering the fear of black-boxed AI in maintenance: Towards a smart colleague, in: *Proceedings of the 2019 Pre-ICIS SIGDSA Symposium*, 2019.
- [47] T. Baier, J. Mendling, M. Weske, Bridging abstraction layers in process mining, *Inf. Syst.* 46 (2014) 123–139.
- [48] H. Van Der Aa, H. Leopold, H.A. Reijers, Efficient process conformance checking on the basis of uncertain event-to-activity mappings, *IEEE Trans. Knowl. Data Eng.* 32 (5) (2019) 927–940.
- [49] F. De Luzzi, F. Leotta, A. Marrella, M. Mecella, On the interplay between business process management and internet-of-things: A systematic literature review, *Bus. Inf. Syst. Eng.* (2024) 1–24.
- [50] K. Akhramovich, E. Serral, C. Cetina, A systematic literature review on the application of process mining to industry 4.0, *Knowl. Inf. Syst.* (2024) 1–48.
- [51] A. Rebmann, A. Emrich, P. Fettke, Enabling the discovery of manual processes using a multi-modal activity recognition approach, in: *BPM 2019 International Workshops*, Revised Selected Papers 17, Springer, 2019, pp. 130–141.
- [52] L. Esposito, F. Leotta, M. Mecella, S. Veneruso, Unsupervised segmentation of smart home logs for human habit discovery, in: *2022 18th International Conference on Intelligent Environments*, IE, IEEE, 2022, pp. 1–8.
- [53] G. Di Federico, E.R. Nikolajsen, M. Azam, A. Burattin, Linac: A smart environment simulator of human activities, in: *International Conference on Process Mining*, Springer, 2021, pp. 60–72.
- [54] M. Shirali, M.F. Sani, Z. Ahmadi, E. Serral, LLM-based event abstraction and integration for IoT-sourced logs, 2024, arXiv preprint arXiv:2409.03478.
- [55] H.H. Beyel, O. Makke, O. Gusikhin, W.M. van der Aalst, Analyzing behavior in cyber-physical systems in connected vehicles: A case study, in: *International Conference on Business Process Management*, Springer, 2023, pp. 92–104.
- [56] M. Cornacchia, K. Ozcan, Y. Zheng, S. Velipasalar, A survey on activity detection and classification using wearable sensors, *IEEE Sens.* 17 (2) (2016) 386–403.
- [57] F. Mannhardt, R. Bovo, M.F. Oliveira, S. Julier, A taxonomy for combining activity recognition and process discovery in industrial environments, in: *Int. Conf. on Intelligent Data Engineering and Automated Learning*, Springer, 2018, pp. 84–93.
- [58] O. Etzion, P. Niblett, *Event Processing in Action*, Manning Publications, 2010.
- [59] M. Dayarathna, S. Perera, Recent advancements in event processing, *ACM Comput. Surv.* 51 (2) (2018).
- [60] A. Baumgräß, M. Botezatu, C. Di Ciccio, R. Dijkman, P. Grefen, et al., Towards a methodology for the engineering of event-driven process applications, in: *BPM 2015, 13th International Workshops*, Springer, 2016, pp. 501–514.
- [61] L. Malburg, M.-P. Rieder, R. Seiger, P. Klein, R. Bergmann, Object detection for smart factory processes by machine learning, *Procedia Comput. Sci.* 184 (2021) 581–588.
- [62] A. Burattin, M. Eigenmann, R. Seiger, B. Weber, MQTT-XES: Real-time telemetry for process event data, in: *BPM 2020 Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Track*, 2020, pp. 97–101.
- [63] A. Burattin, Beamline: A comprehensive toolkit for research and development of streaming process mining, *Softw. Impacts* 17 (2023) 100551.
- [64] A.F. Kurz, R. Seiger, M. Franceschetti, Prototype of an IoT-based activity detection service generator with evaluation datasets, 2025, <http://dx.doi.org/10.5281/zenodo.14860395>.
- [65] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Boston, MA, USA, 1994.
- [66] A.F. Kurz, R. Seiger, M. Franceschetti, B. Weber, Activity and sequence detection evaluation metrics: A comprehensive tool for event log comparison, in: *BPM 2024 Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Forum*, *ceur-ws.org*, 2024.
- [67] J.A. Ward, P. Lukowicz, H.W. Gellersen, Performance metrics for activity recognition, *ACM Trans. Intell. Syst. Technol.* 2 (1) (2011).
- [68] L. García-Bañuelos, N.R. Van Beest, M. Dumas, M. La Rosa, W. Mertens, Complete and interpretable conformance checking of business processes, *IEEE Trans. Softw. Eng.* 44 (3) (2017) 262–290.
- [69] S. Suhothayan, K. Gajasinghe, I. Loku Narangoda, S. Chaturanga, S. Perera, V. Nanayakkara, Siddhi: A second look at complex event processing architectures, in: *Proceedings of the 2011 ACM Workshop on Gateway Computing Environments*, 2011, pp. 43–50.
- [70] D.J. Berndt, J. Clifford, Using dynamic time warping to find patterns in time series, in: *KDD Workshop*, Vol. 10, Seattle, WA, 1994, pp. 359–370.
- [71] M. Ehrendorfer, J. Hebstreit, J. Mangler, S. Rinderle-Ma, Interactive drift visualization in sensor data streams for explainable process outcome prediction, in: *International Conference on Business Process Management*, Springer, 2024, pp. 162–178.