The final authenticated version is available online at https://doi.org/10.1007/978-3-030-58666-9_12.

Analyzing Process Concept Drifts Based on Sensor Event Streams During Runtime

Florian Stertz, Stefanie Rinderle-Ma, Juergen Mangler

University of Vienna, Faculty of Computer Science, Vienna, Austria
{firstname.lastname}@univie.ac.at

Abstract. Business processes have to adapt to constantly changing requirements at a large scale due to, e.g., new regulations, and at a smaller scale due to, e.g., deviations in sensor event streams such as warehouse temperature in manufacturing or blood pressure in health care. Deviations in the process behavior during runtime can be detected from process event streams as so called concept drifts. Existing work has focused on concept drift detection so far, but has neglected why the drift occurred. To close this gap, this paper provides online algorithms to analyze the root cause for a concept drift using sensor event streams. These streams are typically gathered externally, i.e., separated from the process execution, and can be understood as time sequences. Supporting domain experts in assessing concept drifts through their root cause facilitates process optimization and evolution. The feasibility of the algorithms is shown based on a prototypical implementation. Moreover, the algorithms are evaluated based on a real-world data set from manufacturing.

Keywords: Online Process Mining, Concept Drift, Sensor Event Stream, Root Cause Analysis, Time Sequence, Dynamic Time Warping

1 Introduction

"World-class organizations leverage business process change as a means to improve performance, reduce costs, and increase profitability" [23]. Companies can react by adapting their business process to the changing requirements at a large scale, e.g., new regulations, and at a smaller scale, e.g., deviations in sensor streams in manufacturing or medicine. In any case, adaptations of the process logic result in a so called *concept drift* [25].

When adapting business processes, a concept drift might be known in case of explicitly defined and applied process changes, but also unknown and "only" recorded in so called *process event logs* that store information on business process execution in an event-based manner. If the process execution events are continuously collected during runtime, we call this a *process event stream*. Existing techniques detect concept drifts from process event logs in an offline manner (ex post) based on process execution logs [4] or in an online way based on *process event streams* [27,16,20], i.e., during runtime as the processes are executed. Online concept drift detection can be crucial to react on process changes in

time. However, approaches to analyze and identify the reason why a concept drift happened, i.e., its *root cause*, are missing although knowing the root cause contributes to, e.g., optimizing future occurrences of similar concept drifts.

Hence the basic question is how to identify and analyze the root cause for concept drifts at runtime. Several examples suggest that data from IoT devices, i.e., external sources such as sensors can influence the execution behavior of a process. Temperature, for example, might cause exceptions in logistics processes [3]. Variations in parameters might indicate the quality of products in manufacturing [13,8]. The data emitted by sensors is called *sensor event streams* and is captured externally, i.e., outside the process execution [17]. Sensor event streams constitute *time sequence data* [13]. Informally, a time sequence holds quantitative, time-stamped data. We opted to analyze time sequence data instead of time series data as the latter requires equidistant time intervals what is not always the case for the real world cases to be considered.

In order to facilitate root cause analysis for concept drifts, this work addresses the following research questions:

RQ1: *How can drifts in sensor event streams associated with process instances be identified?*

RQ2: *How can the analysis of these drifts help domain experts, to assess root causes and thus propose concept drifts / process evolution?*

The approach takes a *process history* [20] as input. The process history holds an ordered sequence of process models that have been mined online and are connected to per-instance sensor event streams. These sensor event streams are time sequences, and the deviations between the streams of different instances of each model are determined using *dynamic time warping (DTW)*. DTW calculates the distance between two time sequences. The challenge is to interpret the drifts in the sensor event streams to identify future concept drifts in the process model (in contrast to [20], which deals with the identification of concept drifts ex-post). The approach was implemented for a real-world IoT application from the manufacturing domain and a data set was gathered that is used to evaluate the approach. Specifically, we show how the results of the analysis support domain experts in understanding why a drift happened (root cause) and to learn what can be done to efficiently deal with it. The objective is therefore to evaluate the applicability of this approach in finding the cause for a concept drift, to evaluate the performance of this approach, i.e., how many reasons can be correctly detected in which time and to give information on how to adapt the current process model to the current situation.

This paper is outlined as follows: In Sect. 2, a running example as well as preliminaries are introduced. Section 3 features two algorithms to determine drifts in event streams from external sources. Section 4 evaluates these algorithms based on a real world IoT application. Related work is reflected in Sect. 6. Sect. 7 summarizes this work and gives a brief outlook of the planned future work.

2 Running Example and Fundamentals

Figure 1(a) shows the process model of a medical round for a patient of a health care facility. This model represents the current care plan for one specific patient. The general health status of a patient is checked, the blood pressure is measured, and drugs are administered. During runtime process instances are created and executed based on the process model. The execution information is stored in a process event log. Assume that a concept drift occurs, which results in the process model depicted in Fig. 1(b), i.e., an additional hydration check is added in parallel to checking the blood pressure. Another drift could be detected in the data elements of a process instance, for example, the task "Blood Pressure" is in (b) done by nurses, while it has been done in (a) by medical doctors. Existing approaches [16,20,27,15] enable drift detection, but do not explain why the drift happened in the first place.

Unlike process data such as resource or patient age, typically, the temperature and humidity of the patient's room are constantly monitored by external sensors. The sensors produce event streams which consist of data points representing a single measurement. These measurements are typically not stored in a process execution log, but in a different database, since the tasks are not directly linked to any process data. We investigate whether and how such sensor event streams can be exploited in order to analyze and explain why a concept drift happened.

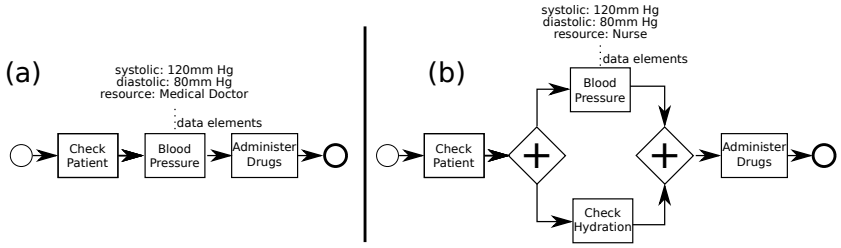


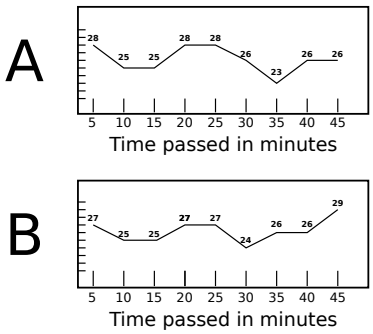
Fig. 1. Concept drift resulting in adapted process model – medical example

This work exploits *process histories* [20] to store drift information reflected by deviations in process models. A process history $HP :=< M_0, M_1, .., M_n, .. >$ contains a list of viable process models $M_i, i = 0, 1, ..$ that reflect the natural evolution of a business process P. While previous work [21] focuses on data that is part of the business logic of a process (e.g., resources as in Fig. 1), this paper addresses high velocity time sequence data that is collected from external sensors, but is otherwise not utilized in the context of processes or sub-processes.

A time sequence is defined as follows in [11, p. 208]: *A sequence of time-stamped data for which the attribute values are the result of measurements of a quantitative real-valued state variable, denoted by* $y \in \mathbb{R}, y = (y(t_1), y(t_2), ..., y(t_n))$.

The challenge is to compare the time sequences in order to detect differences in the associated sensor event streams that can lead to drifts. To compare two

time sequences, an alignment is calculated to determine the distances from one sequence to another. The most common distances measure are the Euclidean Distance (ED) [9] and Dynamic Time Warping (DTW) [2]. While ED has several advantages like linear computing time and being straightforward, it requires time sequences to be of the same length and is deceptive for noise. DTW is also able to globally find the best alignment and can cope with sequences of different length. The complexity is quadratic, since a $m \times n$ matrix has to be constructed, where m and n are the lengths of the time sequence.



Time Series Label	Value at Timestamp
A	28 25 25 28 28 26 23 26 26
B	27 25 25 27 27 24 26 26 29

Fig. 2. Plot of two time sequences and their corresponding values

29	18	12	12	9	9	8	11	8	8
26	17	8	8	9	10	5	8	5	5
26	15	7	7	8	9	5	8	5	5
24	13	6	6	7	7	5	5	7	9
27	9	5	5	3	3	4	8	9	10
27	8	3	3	2	3	4	8	9	10
25	7	1	1	4	7	8	10	11	12
25	4	1	1	4	7	8	10	11	12
27	1	3	5	6	7	8	12	13	14
B/A	28	25	25	28	28	26	23	26	26

Fig. 3. Warp matrix constructed using DTW: orange cell = distance

We use DTW as we are dealing with sequences of different lengths. Figure 2 shows time sequences A and B, together with a table containing the exact values at every timestamp. The $m \times n$ matrix `D` for the alignment between A and B is constructed by starting in the bottom row and filling every value from left to right, as can be seen in Fig. 3. The distance as absolute difference between the actual values is calculated, so in the bottom left corner it is

$| 28 - 27 | = 1$. Afterwards the cheapest cost from one of the cells before is used, so $D(n, m_1)$,$D(n_1, m_1)$ and $D(n_1, m)$ is added to the distance and assigned to $D(n, m)$. The definition follows [19]: $D(i, j) = Dist(i, j) + min[D(i-1, j), D(i, j-1), D(i-1, j-1)]$

The distance is found in the top right corner of Fig. 3. The alignment can be found using back-tracing starting in the top right corner and following the path back to the start cell in the bottom left corner, i.e., the green cells.

This work employs the DTW Barycenter Averaging (DBA) [18] algorithm. DBA uses DTW as distance measure and calculates the average time sequence for a set of time sequences. It starts by an arbitrary average sequence and adapts it iteratively by trying to minimize the sum of squared DTW distances from the average sequence to the set of sequences. The computation time of this technique is again quadratic, since a DTW matrix has to be created for each iteration.

3 Time Sequence Assignment and Root Cause Detection

This section details the main contribution of this work, i.e., how to utilize time sequence data from sensor event streams to flag process instances for closer inspection when performing a root cause analysis for concept drifts. Note that an analysis for both cases is possible, i.e., finding reasons for concept drifts that have already been detected (ex post) and – particularly during runtime – detecting and analyzing deviations in the sensor event streams that might lead to a future concept drift, i.e., a process evolution.

We start with the architecture of the solution presented in this work (cf. Fig. 4) as foundation for the subsequent considerations. Note that the components *Time Sequence Module* and *Drift Decision Detection* (both red) realize the contribution of this paper.

For detecting drifts, sensor event streams are taken as input. They can be fed into the system by any process execution engine. In the manufacturing scenario presented throughout this paper, the Cloud Process Execution Engine CPEE[1] is utilized. The sensor components provide data streams collected through tasks in (5) (Fig. 7). The process history is therefore extended to include all the data from the sensors. Further implementation details will be described in Sect. 4.

3.1 Time Sequence Module

This component enriches the process history by adding the average time sequence of every sensor to each new viable process model M_n. To relate a time sequence of an event stream produced by a sensor to a specific process instance, the timestamps of the first and currently last event of the stream are taken into account and the corresponding time sequence is cut out for the process instance. In the example (cf. Fig. 1), time sequences between the start time of "Check patient" and end time "Administer Drugs" are mapped to a process instance for both sensors, temperature and humidity.

[1] `http://cpee.org/`

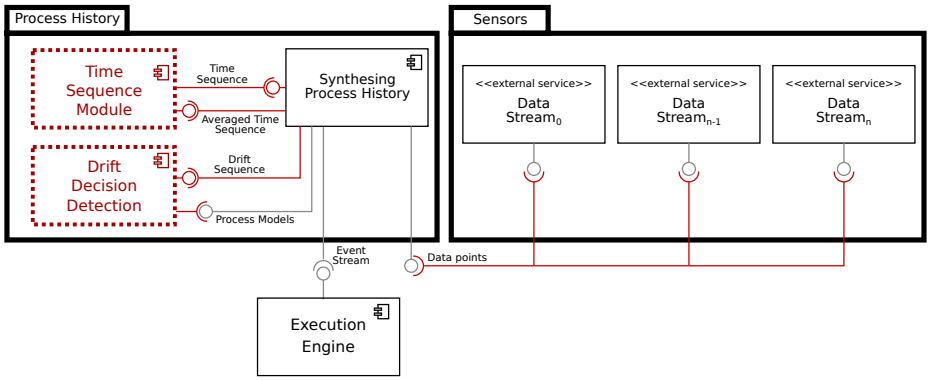


Fig. 4. Proposed architecture: red parts denote the contribution of this paper

Algorithm 1 shows the pseudo code for the *Time Sequence Module*. The set of unfitting traces T is provided by the process history, i.e., those traces that do not conform to the current model M_n. The time sequences are provided by external sensors through the process history. In line 1, the return value is initialized as an empty dictionary. A dictionary here reflects a hash table [6] data structure with a key and a related value to it. Line 2 starts the iteration over time sequences of each sensor. At first, the time sequence for each trace out of T is collected starting in line 6. We map a time sequence from a sensor to a trace by beginning from the first time stamp of this trace to the last known time stamp of this trace, as can be seen in line 9. Since we are working in an online environment, it is possible that traces just started and contain only one event, which results in no time sequence for this specific trace.

Another important aspect of the online setting is, that each trace could have greatly varying execution times, since we do not know how long a complete trace is going to take. To diminish the impact of outliers and faulty or aborted instances, we exclude sequences with a duration shorter than the first quartile minus 1.5 times the IQR (Interquartile Range) or with a duration greater than the third quartile plus 1.5 times the IQR, similar to boxplots. The IQR is calculated here between third and first quartile. Other methods for detecting outliers can be applied here or even working with every trace.

We calculate the quartile at (lines 11 and the IQR at line 12). Afterwards the outliers of the collected time sequences are removed. Otherwise the time sequence will be taken into account (lines 14- 17). In the last step (line 18), the averaged time sequence is put into the dictionary ATS with its corresponding sensor id as key. The dictionary of averaged time sequences is then sent back to the process history. The current viable process model in the process history is thereby extended by this dictionary, which is then used by the `Drift Decision Detection` component.

Figure 5 shows an example of Alg. 1 where the following 5 sequences for the room temperature have been collected: $(27, 29)$, $(27, 27, 28, 27, 29, 27, 29, 28]$,

Input: ST: dictionary of a time sequence for each sensor ID
Result: ATS: dictionary of an averaged time sequence for each sensor ID

```
ATS = dict()
for w,ts in ST do
    // w is id of sensor, ts its corresponding time sequence
    temp_ts_list = list()
    stats = list()
    for t in ts do
        if |t| < 2 then
            next
        temp_ts_list.append(time_sequence(t.first_event.timestamp,t.last_event.timestamp))

        stats.append(temp_ts_list.last.length)
    first,second,third,fourth = quartile(stats)
    x = iqr(stats)
    ts_list = list()
    for t in temp_ts_list do
        if |t| < first − x || |t| > third+x then
            next
        ts_list.append(t)
    ATS[w] = dba(ts_list)
return ATS
```

Algorithm 1: Find relevant time sequences and compute avg. time sequence

$[30, 30, 30, 30, 29, 27)$, $(27, 30, 29, 30, 29, 27)$, $(30, 27, 29, 29, 28, 28, 30, 29, 30, 29, 30, 29, 28, 29)$. The third quartile for the lengths of these sequences would be 8, the first quartile is 6. Therefore the IQR equals 2. This excludes sequences which are shorter than 3 or longer than 11. The first sequence $(27, 29)$ and the last sequence $(30, 27, 29, 29, 28, 28, 30, 29, 30, 29, 30, 29, 28, 29)$ are not taken into account for the calculation and are printed in a green dashed line. The red time sequence shows the calculated average time sequence with DBA.

3.2 Drift Decision Detection

The second component of the solution proposed in this work is the `Drift Decision Detection` component.

Algorithm 2 shows the detection of the most likely external sensors that can have caused the drift in the process model. The process history sends two process models to this component in order to receive a set of external sensors which caused a drift from the first model to the second model. Each of these process models contains its average time sequence for each external sensor. The dictionary TH is user defined and contains for every external sensor, a related threshold for the distance between the two average time sequences. A different threshold for each external sensor is needed, because the dynamic warp distance is calculated using the differences in the data points. Assume that for the running example (cf. Fig. 1), the ideal temperature ranges between 27° and 29°C. Thus

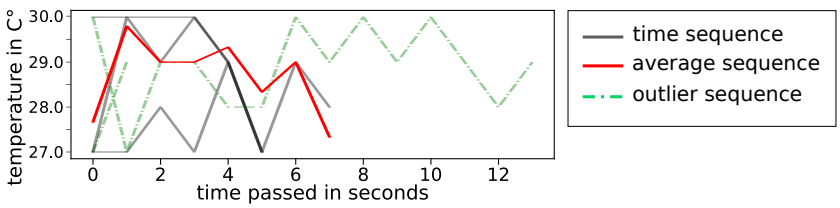


Fig. 5. Exemplary Result of Alg. 1. The red line is the average sequence calculated using DBA. The green dashed lines represent outliers, potentially due to a faulty process instance.

Input: M: A list of process models with averaged time sequence dictionary
TH: Dictionary of thresholds for similarity
Result: D: Set of sensor IDs that are likely to have caused a drift

```
1 D = set()
2 for ws_id in M.second.ATS.keys do
3 |   if ws_id not in M.first.ATS.key then
4 |   |   next
5 |   ts_a = M.first.ATS[ws_id]
6 |   ts_b = M.last.ATS[ws_id]
7 |   if dtw(ts_a,ts_b) > TH[ws_id] then
8 |   |   D.add(ws_id)
9 return D
```

Algorithm 2: Detecting a set of sensor data streams which caused a drift

similar time sequences have a low absolute cost depending on the length of the alignment. A sensor keeping track of parts with higher tolerances can therefore have a higher warping distance for similar sequences. These thresholds can be approximated using a test set for the classification, where an expert has to define sensitivity and specificity for the sensor data. At the start, the return value D is initialized as an empty set in line 1. The loop iterates over every key that is in the dictionary of averaged time sequences (ATS) in the second model in line 2. It is to be noted that the models could have different events attached to them, but the external sensors should be the same. If a specific sensor is not present in both models, we cannot take it into account, see lines 3. The average time sequence for one sensor is retrieved for both models in line 5 and 6. If the cost of the alignment, which is reflected in the top right cell of the warping matrix, see Fig. 3, is greater than the corresponding threshold to the sensor, this sensor is added to the return value D in line 8. This set of external sensors is then returned to the process history, where it is stored.

3.3 Performance Optimizations

One problem with DTW is the computation time, since a $m \times n$ matrix has to be constructed, where m is the length of one time sequence and n the length of the other time sequence.

One approach that can be used to optimize the performance of Alg. 1 and Alg. 2 is FastDTW [19]. FastDTW aims at performing DTW in linear time with 3 steps. First the time sequence is shrunk into smaller time sequences that reflect the same curve approximately. Then the minimum distance warp path is computed for the smaller time sequence. Afterwards this warp path is adjusted to the original time sequence. For a length of 10000 data points the computation time can be reduced from 57.45 seconds to $8,42$ seconds. The error rate for this approximation is below 1%.

Another way to speed up time warping is early abandoning [24,12]. In this strategy, if the warping distance is above a certain threshold while creating the warping matrix, the algorithm can stop the execution and label it as an outlier.

Both methods are suitable optimizations for Alg. 2, since it uses user defined thresholds for each external sensor, but not for Alg. 1. This is because the average time sequence is computed using the complete DTW distance score, thus not exact methods like FastDTW and early abandoning cannot be used. In Sect. 4, Alg. 2 is evaluated using DTW and FastDTW.

4 Evaluation

Fig. 6. GV12 part

The algorithms and components presented in Sect. 3 are prototypically implemented and tested based on a real-world IoT application from the manufacturing domain in order to prove the effectiveness and feasibility of the approach: The Austrian Center for Digital Production[2] produces parts called GV12 for a gas-turbine (see Fig. 6) as a prototypical solution for a customer. The requirements for the part include high precision manufacturing (low tolerances, i.e. some aspects allow for deviations of only 0.02 mm), and strict quality assurance for each part, including (a) detailed tracking of manufacturing data for each part and (b) measuring the adherence to tolerances for more than 12 features with automated precision measurement equipment.

The entire production is carried out automatically by implementing the interaction between the involved machines through industrial robots and transport systems. We focus on the manufacturing and quality control as shown in Fig. 7. Currently more than 20 processes and sub-processes are involved, and orchestrated during production of up to 40 parts per batch. Figure 7 illustrates the basic manufacturing logic:

- (1) Batches of up to 40 pieces are ordered, the manufacturing is scheduled.
- (2) The interaction between all machines and robots is orchestrated, while enforcing industrial safety principles[3].
- (3) Individual parts are produced by using the following three steps:

[2] `https://www.acdp.at/`
[3] `https://www.iso.org/standard/51330.html`

  - Machining of a part from hardened steel, which takes about 4 minutes per part.
  - Measuring of the part by a high-speed optical micrometer[4], while the next part is machined. This takes about 12 seconds per part.
  - Measuring of a part with automated precision measurement equipment[5], which takes about 8 minutes per part, and is also done in parallel to the machining.
- (4) A generic machine monitoring process determines when to start data collection for both, machining and measuring.
- (5) A generic data collection process produces a continuous stream of values when the laser of the high-speed optical micrometer is scanning the surface of the part.

The "Measure with Keyence" task is done automatically by a Keyence measuring machine at no additional cost in parallel to the production of the next part. As the Keyence machine is very compact, fast, and operates without touching the part, this step is done after the robot extracted the part from the production machine, and before it puts it on the palett. On the palett it is transported to the MicroVu measuring machine, which is rather big and has to be operated in a location with low vibrations and special light and temperature conditions. The task "Measure with MicroVu", as opposed to the task "Measure with Keyence", is required by the customer, because it basically creates an objective report about the quality of a part.

After some time, deviations in the process event stream collected by (5) can be observed. These deviations can happen based on

- physical effects due to deteriorating machining tools, or temperature fluctuations.
- problems stemming from accumulating debris that affects the production quality as well as measurement quality.

Up to this point only the extreme values of the time sequence (i.e. min, max) from the Keyence machine were used, which are sufficient for detecting (a) if the part has been dropped by the robot (no part), or (b) the part appears to be too big (i.e., it is engulfed by chips). However, the extreme values proved not to be effective for early detection of parts which do not comply to the quality requirements. If an early reliable estimation of the quality was available, it could be used to skip the 'Measure with MicroVu" altogether, which would save valuable resources.

Hence, our approach for this evaluation is instead of taking only the extreme values of the measurement into account, to analyze the complete time sequence of the measurement. Every time the process history detects a drift in the data elements, i.e, "Measure with MicroVu" detects only faulty instances, a drift

[4] `https://www.keyence.com/products/measure/micrometer/ls-9000/index.jsp`
[5] `https://www.microvu.com/products/vertex.html`

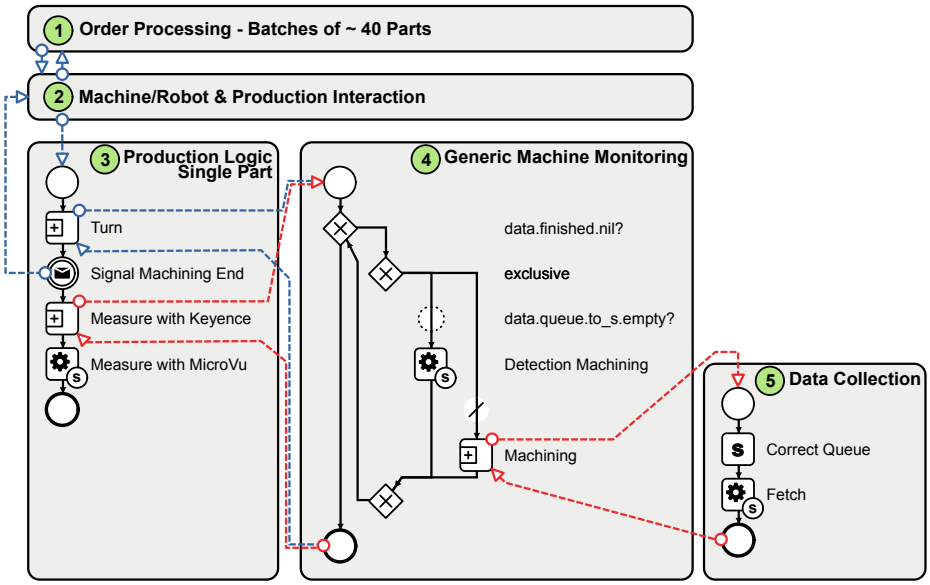


Fig. 7. Batches of GV12 parts

has been detected in the data model. Algorithm 1 calculates then the average sequence, e.g., Fig. 8. The threshold for Alg. 2 is here calculated ex-post, with the results of "Measure with MicroVu".

4.1 Prototypical Implementation – RQ1

The orchestration of the BPMN 2.x based process models on the factory floor (cf. Fig. 7) is driven by the process engine CPEE[1]. The process history component subscribes to the CPEE in order to receive information about every executing event. The external sensor represented by activity "Measure with MicroVu" is a high-speed optical micrometer[6].

The data set[7] contains 1026 traces in the XES[8] format for 37 parts and is available at the figshare repository [10]. The traces are produced by 13 different process models. The sensor values amount for 6.2 MiB out of 1 GiB of total data. A time sequence for this event contains on average about 776 data points. Measurements from a time sequence range from 4.09 up to 37.87 millimeters.

The process history creates the process models as described in [20]. The process history uses a sliding window approach to deal with infinite amount of data that is being captured by listening to streams, i.e., only a specified number of traces are used for detecting a new process model in the history.

[6] `https://www.keyence.com/products/measure/micrometer/ls-9000/index.jsp`
[7] `http://gruppe.wst.univie.ac.at/data/timesequence.zip`
[8] `http://xes-standard.org/`

The `Time Series Module` component is implemented in Python as we are using the `tslearn` package [22] because it provides functions for computing an alignment using DTW as well as DBA for finding the average time series. The results of this component are retrievable via a RESTful web service as well. The `Drift Decision Detection` component also uses these libraries. The result of Alg. 1 on the data set is depicted in Fig. 8. Each grey sequence relates to one specific trace and shows the measurement data points of one part. As it can be seen, one time sequence only lasts for about 8 seconds, while the other ones last about 12 to 14 seconds. The average sequence, calculated using DBA is depicted in red. This sequence is stored as additional information in the process history for the current process model. Since this log only provides one sensor, i.e., "Keyence", only one sequence has to be calculated using DBA. To determine the feasibility of the implementation, we furthermore looked at the following questions:

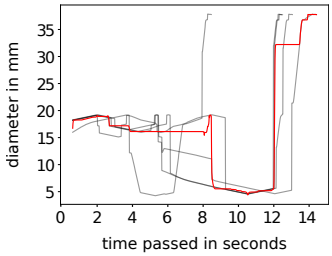


Fig. 8. Result of implementation. The red line represents the average sequence

- How are parameters such as the duration of a process instance or the number of traces in the process history sliding window, affecting the algorithms?
- What is the performance of these algorithms?

The performance of Alg. 2 is only depending on the length of the average time sequence, which is hard to tweak with parameters of the process history and the number of sensors. It can be adapted by changing the amount of data points that are to be stored per time unit. The performance of Alg. 1 on the other hand is highly dependent of the parameters set for the process history.

In order to rate the effectiveness of the approach it is possible to rely on the data provided by "Measure with MicroVu". Out of 37 parts, 18 parts were faulty. With this knowledge we first varied the threshold in order to achieve 0% false negative detection of parts. In other words: no parts that are faulty should be delivered to the customer, on the other hand it is acceptable that some parts that are actually good are detected as faulty. The optimal threshold proved to be 22.

Fig. 9. Chips on GV12 - wrong measurement

When varying the window size, i.e., the number of traces to be analyzed during the drift detection, the following results emerge: As can be seen in Tab. 1, a window size of 5 and a threshold of 22 is sufficient for our scenario. With these values 100% of the faulty parts can be identified, without relying on the time intensive "Measure with MicroVu" task. This means that for a rate of 18 faulty tasks, almost 50% of the production time can be saved, based on calculation of drift for sensor event streams.

Table 1. Results of both Algorithms

Window Size	False Positives	False Negatives	Runtime
1	45%	0%	1.4s
5	0%	0%	10.3s
10	0%	0%	20.7s

4.2 Concept Drift Prediction / Process Evolution - RQ2

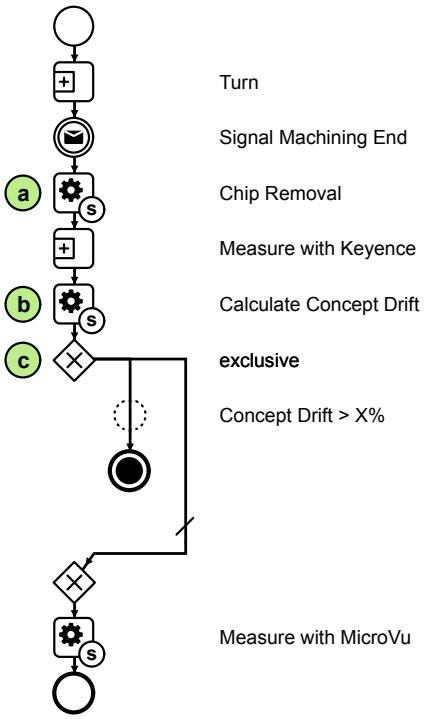


Fig. 10. GV12 Prototype Part

For the task "Measure with Keyence" in Fig. 7, as collected by process (5), the deviations for the measurements between parts have some serious repercussions that can lead to multiple possible process evolutions.

The results were discussed with three domain experts involved in the production of the GV12 parts. When discussing the results from the drift analysis, the domain experts came up with the following discussion points.

As can be seen in Fig. 9, the machining produced long chips, which entangled the part. Furthermore, the comparison of the drift for "Measure with Keyence" with the quality data from "Measure with MicroVu" (see Fig. 7) was deemed sufficient for predicting the quality of a part, thus allowing for immediate removal of faulty parts from production which decreases the overall time per batch greatly: the less "Measure with MicroVu" the better. This led to the proposal of the concept drifts / process evolutions shown in Fig. 10. Overall, the concept drifts can be classified as follows:

- Static Evolution (a): an extra activity "Chip Removal" was proposed to be inserted, based on the observed drifts. A robot blows compressed air on the part, to remove debris and chips, which allows for more accurate measuring. This will allow for lower possible thresholds in future / similar scenarios.
- Dynamic Evolution (b)+(c): The drift is to be actively calculated at runtime, based on previous process instances, and made available to the current instance. A decision (c) is proposed to be inserted, that allows for terminating single parts without "Measure with MicroVu".

Table 2. Runtime of Alg.1.

Datapoints/ Sequences	10	100	1000
5	1.17s	1.22s	3.53s
10	1.18s	1.31s	7.01s
50	1.29s	4,43s	75.83s
100	1.22s	4.63s	133.00s

Table 3. Runtime of Alg.2.

Datapoints	DTW	FastDTW
10	0.88s	0.0002s
100	0.87s	0.0003s
1000	0.91s	0.001s
10000	4.70s	0.01s

Performance evaluation: Table 2 shows the runtime of Alg. 1 to analyze the applicability of this solution. We generated random time sequences with 10, 100, and 1000 data points on average. Algorithm 1 is then applied on a set consisting of 5, 10, 50, and 100 time sequences. The results of 10 data points on average show, that the execution time of Alg. 1 for 100 sequences is even lower as the one for 50 sequences. This happens, because the time for the calculation is so small, that other currently running tasks of the operating system may interfere with the execution.

With 100 data points, Alg. 1 affects the total execution time to a greater extent, especially with more sequences: 50 sequences result in a more than 3 times longer execution time than 10 sequences. With 1000 data points, the execution time with more than 50 sequences is increased by more than 10 times the execution time with 10 sequences.

Table 3 shows the comparison between DTW and FastDTW (cf. Sect. 3.3) in terms of speed. As expected, FastDTW is the faster technique as it works in linear time. Unfortunately the results differ greatly for DTW when compared to FastDTW. While, for example, the distance between 2 sequences with random values between 90 and 110 and 10000 data points was 343.2 when using DTW, the distance equals to 45299 when using FastDTW. Since both algorithms are highly depending on the global maximum of the alignment of sequences, FastDTW is not a suitable option.

Assessment by domain experts: We presented the results to a machine operator, a mechanical engineer, and a measurement engineer. All three experts were overall satisfied with the results. They highlighted that to the best of their knowledge in order to achieve similar results, additional – hard to configure – software would be necessary.

5 Discussion

Possible **limitations** in the context of the presented approach include:
• *Performance:* An important aspect for the performance of this approach is the number of data points in a time sequence. As can be seen in Tab. 2, even

with 1000 data points and 10 time sequences the implementation took about 7 seconds. This of course increases linearly with number of sensors. Other techniques like FastDTW instead of DTW, reduce the runtime drastically, but the alignment using FastDTW varies greatly from the globally best alignment using DTW, which leads to worse results.

• *Sensors selection:* While in general IoT devices such as external sensors provide a valuable source for detecting the cause of a concept drift, choosing the "right" IoT device may be hard in some cases. The reason is that in many real-world scenarios there is a plethora of devices creating data streams and therefore time sequences. Taking an external sensor into account, that has no relation to the process model, for example, can produce wrong results, since the time sequences of this sensor may vary to a great extent and hence be incorrectly identified as the source of a drift. In addition, the runtime is heavily depending on the number of sensors, hence not significant sensors should be excluded. Therefore it is recommended that an expert additionally validates the results. If no sensors can be excluded by experts, a parallel optimization is advised of Alg. 1 where each sensor can be calculated separately. This reduces the execution time of the algorithm to the execution time of the sensor with the most data points.

• *Thresholds:* Another important aspect is finding the threshold for Alg. 2 automatically. If there is a training set, the threshold can be calculated until a specified sensitivity and specificity are met. Otherwise, an expert sets the threshold.

Also, the following **threats to validity** have to be considered: The data set of the evaluation comprises the data of one sensor. Hence, the selection of the sensors cannot be evaluated. While the increase of the runtime is predictable, the quality of the results can differ greatly, if not related sensors are taken into account. The real-world case comes from the manufacturing domain, where the selection of the sensors may be easier, since the conditions of the events are often in a controlled environment, like a factory. In other domains, like the medical domain or logistic domain where numerous external data stream sources can affect the execution of a process, the selection may be more difficult. In future work, experiments in different domains are planned.

6 Related Work

Several algorithms for offline process discovery exist [1]. Existing work shows that a selection of these algorithms can be used for online process discovery as well. This includes the heuristics miner [5], which takes the frequency of events into account and the inductive miner [14], which tries to find a certain block structure to find splits for the process model. Concept drift detection can also conducted in an offline [4] and online manner [16,20,27,15]. However, the mentioned online mining techniques neither consider external data nor analyze the root cause for concept drifts. [26] enables the visual exploration of the concept drift type. This work, by contrast, analyzes sensor event streams as time sequence data. Time series data in process mining domain have been analyzed for finding decision points by [7] in an offline manner. Other approaches exploit sensor data for

outcome predictions for process instances [3] and manufacturing systems [13], but do not address concept drifts.

7 Conclusion

This paper elaborates a novel approach to predict the root cause of a concept drift in a business process based on external sensor streams. Two algorithms are introduced to compare the time sequences associated with the sensor event streams in combination with the process event stream. In the evaluation, it is shown that the algorithms are capable of detecting the drifts in the external sensor event stream with high accuracy, given a certain amount of traces for a specific setting. A big factor for this approach, is the type of available sensors. A domain expert has to distinguish which sensors are important, and is used best to verify a drift in certain aspects of a produced part. Otherwise the computation time is increased with no benefit, as some external sensors are not able to contribute to a drift in the process. Furthermore, three domain experts, based on the highlighted drifts, verified root causes, and proposed multiple concept drifts / process evolutions, thus showing the validity of the solution.

Future work aims at algorithms for predicting and explaining future drifts.

Acknowledgment

This work has been partly funded by the Austrian Research Promotion Agency (FFG) via the "Austrian Competence Center for Digital Production" (CDP) under the contract number 854187. This work has been supported by the Pilot Factory Industry 4.0, Seestadtstrasse 27, Vienna, Austria.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
2. Berndt, D.J., Clifford, J.: Using dynamic time warping to find patterns in time series. In: KDD workshop. vol. 10, pp. 359–370. Seattle, WA (1994)
3. Borkowski, M., Fdhila, W., Nardelli, M., Rinderle-Ma, S., Schulte, S.: Event-based failure prediction in distributed business processes. Inf. Syst. 81, 220–235 (2019)
4. Bose, R.J.C., Van Der Aalst, W.M., Zliobaite, I., Pechenizkiy, M.: Dealing with concept drifts in process mining. IEEE Trans. Neural Netw. Learning Syst. 25(1), 154–171 (2014)
5. Burattin, A., Sperduti, A., van der Aalst, W.M.: Control-flow discovery from event streams. In: 2014 IEEE Congress on Evolutionary Computation (CEC). pp. 2420–2427. IEEE (2014)
6. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. MIT press (2009)
7. Dunkl, R., Rinderle-Ma, S., Grossmann, W., Fröschl, K.A.: A method for analyzing time series data in process mining: application and extension of decision point analysis. In: CAiSE Forum. pp. 68–84 (2014)

8. Ehrendorfer, M., Fassmann, J., Mangler, J., Rinderle-Ma, S.: Conformance checking and classification of manufacturing log data. In: Business Informatics. pp. 569–577 (2019)
9. Faloutsos, C., Ranganathan, M., Manolopoulos, Y.: Fast subsequence matching in time-series databases, vol. 23. ACM (1994)
10. Grossmann, W., Rinderle-Ma, S.: Fundamentals of Business intelligence. Springer (2015)
11. Junkui, L., Yuanzhen, W.: Early abandon to accelerate exact dynamic time warping. International Arab Journal of Information Technology (IAJIT) 6(2) (2009)
12. Kammerer, K., Hoppenstedt, B., Pryss, R., Stökler, S., Allgaier, J., Reichert, M.: Anomaly detections for manufacturing systems based on sensor data - insights into two challenging real-world production settings. Sensors 19(24), 5370 (2019)
13. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Application and Theory of Petri Nets and Concurrency (2013)
14. Maaradji, A., Dumas, M., Rosa, M.L., Ostovar, A.: Detecting sudden and gradual drifts in business processes from execution traces. IEEE Trans. Knowl. Data Eng. 29(10), 2140–2154 (2017)
15. Maggi, F.M., Burattin, A., Cimitile, M., Sperduti, A.: Online process discovery to detect concept drifts in ltl-based declarative process models. In: On the Move to Meaningful Internet Systems. pp. 94–111 (2013)
16. Mottola, L., Picco, G.P., Oppermann, F.J., Eriksson, J., Finne, N., Fuchs, H., Gaglione, A., Karnouskos, S., Montero, P.M., Oertel, N., Römer, K., Spieß, P., Tranquillini, S., Voigt, T.: makesense: Simplifying the integration of wireless sensor networks into business processes. IEEE Trans. Software Eng. 45(6), 576–596 (2019)
17. Petitjean, F., Ketterlin, A., Gançarski, P.: A global averaging method for dynamic time warping, with applications to clustering. Pattern Recognition 44(3), 678–693 (2011)
18. Salvador, S., Chan, P.: Toward accurate dynamic time warping in linear time and space. Intelligent Data Analysis 11(5), 561–580 (2007)
19. Stertz, F., Rinderle-Ma, S.: Process histories – detecting and representing concept drifts based on event streams. In: Cooperative Information Systems. pp. 318–335 (2018)
20. Stertz, F., Rinderle-Ma, S.: Detecting and identifying data drifts in process event streams based on process histories. In: CAiSE Forum. pp. 240–252 (2019)
21. Stertz, F., Rinderle-Ma, S., Mangler, J.: Data set containing process execution log data with time sequence information for conference proceeding 2020 paper: Analyzing process concept drifts based on sensor event streams during runtime. `http://dx.doi.org/10.6084/m9.figshare.12472634`
22. Tavenard, R.: tslearn documentation (2018)
23. The Hackett Group: Enabling business process change (2019), `https://www.thehackettgroup.com/business-process-change/`
24. Wei, L., Keogh, E., Van Herle, H., Mafra-Neto, A.: Atomic wedgie: efficient query filtering for streaming time series. In: Fifth IEEE International Conference on Data Mining (ICDM'05). pp. 8–pp. IEEE (2005)
25. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine learning 23(1), 69–101 (1996)
26. Yeshchenko, A., Ciccio, C.D., Mendling, J., Polyvyanyy, A.: Comprehensive process drift detection with visual analytics. In: Conceptual Modeling. pp. 119–135 (2019)
27. van Zelst, S., van Dongen, B., van der Aalst, W.: Event stream-based process discovery using abstract representations. Knowl. and Inf. Syst. 54(2), 407–435 (2018)