Detecting and Identifying Data Drifts in Process Event Streams Based on Process Histories

Florian Stertz(✉) and Stefanie Rinderle-Ma

Faculty of Computer Science, University of Vienna, Vienna, Austria
{florian.stertz,stefanie.rinderle-ma}@univie.ac.at

Abstract. Volatile environments force companies to adapt their processes, leading to so called concept drifts during run-time. Concept drifts do not only affect the control flow, but also process data. An example are manufacturing processes where a multitude of machining parameters are necessary to drive the production and might be subject to change due to e.g., machine errors. Detecting such data drifts immediately can help to trigger exception handling in time and to avoid gradual deterioration of the process execution quality. This paper provides online algorithms for concept drift detection in process data employing the concept of process histories. The feasibility of the algorithms is shown based on a prototypical implementation and the analysis of a real-world data set from the manufacturing domain.

Keywords: Process technology · Online process mining · Concept drifts

1 Introduction

Flexibility and change are still among the most pressing challenges for processes [12]. This holds particularly true for data-driven process executions in volatile environments such as manufacturing processes [11]. Manufacturing processes control and are controlled by a multitude of data, e.g., machining parameters and sensor data that constantly monitor the state of the process and the machines. Changes in these parameters are common due to, for example, environmental changes or errors, and can be of tremendous importance for the quality of the process and the product. Similar requirements hold for patient treatments where shifts in vital parameters have to be detected immediately. Hence it is of great importance to be able to detect changes in the data attributes of processes, specifically during run-time, i.e., based on process event streams.

This necessitates making a next step in detecting and evaluating so called concept drifts [6]. So far concept drift refers to changes in the control flow of the process that are discovered based on process execution logs. In [14], we have provided algorithms for detecting and representing concept drifts in control

C. Cappiello and M. Ruiz (Eds.): CAiSE Forum 2019, LNBIP 350, pp. 240–252, 2019.
https://doi.org/10.1007/978-3-030-21297-1_21

flow from *event streams*. This work aims at detecting changes in process data, called data drift in the following, from process event streams at run-time. This is necessary as detecting data drifts from process execution logs ex-post might be too late in order to take necessary actions in many cases.

Generally, data drifts can be categorised following the same guidelines gathered from [6]: data drifts can have recurring effects as well as incremental effects or just reflect sudden changes in the business process logic. As said before, data drifts are also to be detected during run-time and not ex post.

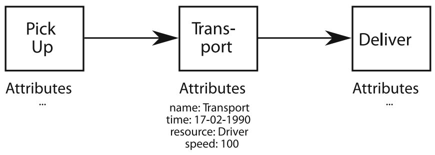

Fig. 1. Process model with data attributes of event `Transportation`

Figure 1 shows a process example from the logistics domain. A product is picked up by a delivery service, transported and delivered to the customer. The data attributes for the event `transportation` are timestamp, name of the event, resource that is executing this event, and average speed. Suddenly this attribute of newer events changes as the driver is now driving significantly slower on average. The reason for this can be manifold, like a construction site on the road, or even a construction site on a different road, which causes the normal route to be jammed. The control flow of this process is not changed, but the data attributes show a drift in the execution of the process, a data drift. Detecting such drifts early helps tremendously in finding errors and bottlenecks that suddenly occur. A data drift could also reflect the natural evolution of a process, e.g, instead of only doctors, nurses administer drugs as well, due to a legislation change. This would be reflected in a new organisational role for this event.

Similarly to control flow drifts [6], data drifts can have different effects, i.e., recurring as well as incremental effects or they just reflect sudden changes in the business process logic. Moreover, data drifts must be detected during run-time and not ex post for many application domains where immediate action is required. Finally, data-intense processes are often emitting a huge amount of events in high frequency. All these challenges will be tackled along the following research questions: **RQ1** How to detect data drifts from process event streams online, i,e., during run-time? and **RQ2** Which types of data drifts can be identified from event streams? How to define and identify them?

Note that the problem is two-fold as reflected by the research questions: In RQ1 it is detected that a data drift has just happened. RQ2 and RQ3 aim at identifying the type of the data drift, e.g., recurring. For addressing RQ1, the already established concept of `process histories` [14], is extended to store information on process data attributes and to allow the detection of data drifts. These drifts are identified using outlier detection on the values of a data attribute. The approach can independent of the contol flow of the process if instead of a model, only event attribute pairs are saved. This would yield the disadantage of not seeing the data drifts as the evolution of a process without the process history. RQ2 yields a formal definition for the data drift types. RQ3 is tackled by an algorithm that determines the type of a data drift based on process histories. Summarizing, this paper provides definitions for extended process histories and data drifts as well as two algorithms. One of them synthesizes the extended process history in order to detect the data drift and the other one determines its type. The definitions and algorithms form the conceptual artefacts of this paper. They are evaluated through a prototypical implementation and application to a real-world data set from the manufacturing domain.

The paper is structured into the following sections. Section 2 provides fundamentals. In Sect. 3 the definition of data drift types and two new algorithms are presented. This section is followed by the evaluation in Sect. 4. In Sect. 2, related work is discussed and an outlook and summary are provided in Sect. 5.

2 Fundamentals and Related Work

We recap the fundamentals on process mining and event streams, especially events containing data attributes using related work. Process histories, previously defined in [14], are extended to comprehend viable data attributes into the process history and to detect new types of drift, so called data drifts.

Process mining covers three tasks [3]: process discovery, the mining of a process model based on a process execution log, process conformance checking, which calculates the fitness of a process instance to a process model, and process enhancement, which allows to improve already discovered models. A process execution log consists of a `log` root node. A log may contain an arbitrary number of process instances, so called `traces` and these traces have an arbitrary number of activities, so called `events`. Process execution logs use the XES format [1].

The main contribution of this work focuses on events and their data attributes. Common attributes would be the point of time when an event has been executed, a organisational resource that has executed the event, or other arbitrary data attributes, e.g., the cost of an activity.

Process mining is usually applied ex post. This means that process models are discovered offline after their execution and storage in a process execution log, like the α-miner [10], which transfroms a directly follows translation [15] out of the log into a Petri Net. To negate this disadvantage, so called online process mining algorithms [14,15] have been developed. The main difference is the input data. While the offline algorithms use a process execution log file, online algorithms use an `event stream` as input.

An event stream represents a continuous flow of sequentially processes events and can be used to discover process models [7,15] as well to synthesise **process histories** [14]. A process history contains every viable business process model, that has been detected using an event stream. A viable model is defined, if it fits the currently relevant traces significantly better than the old model. To calculate the viability of process models, conformance checking techniques are applied. So far, only control flow drifts are captured in a process history, in fact data attributes are rarely considered except some exceptions like the decision mining algorithm [13]. In order to enable the detection of data drifts, we define the `data-extended process history` as follows.

Definition 1 (Data-extended Process History). *Let P be a process and ES the corresponding process event stream. A data-extended process history $H_P :=< M_0, M_1, ..., M_{n-1}, M_n, .. >$ is a list of viable process models $M_n, n \in \mathbb{N}$ that have been discovered for P from ES with M_n being the current model for P. $M \in H_P$ is defined as*

$$M :=< E, < (e_0, A_0^c), \ldots, (e_k, A_k^c) >>, e_i \in E \textit{ with}$$

- *$E \subseteq ES$ is the set of all events in M;*
- *$e_i \in E : e_i = (l_{e_i}, A_i)$, i.e., an event stores its label l_{e_i} and the set of data attributes A_i;*
- *For $e_i, A_i^c \subseteq A_i$ denotes the sub set of attributes from A_i that have caused the data drift.*

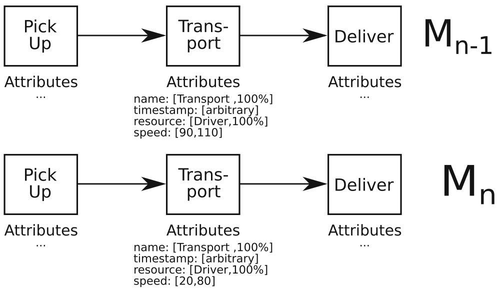

Fig. 2. Process History showing a data drift in the attribute `speed`.

Figure 2 shows the extended process history for the example of Fig. 1. The control flow of the models M_n and M_{n-1} is not changed, but still a new model has been detected because of a data drift in the event `Transport`. As can be seen, the lower bound for the average speed in M_{n-1} equals 90 and the upper bound equals 110. A number of outliers have been detected, e.g, $40, 40, 40, 40, 50, 50, 50, 60, 60, 60, 60, 50, 50$, and 50. This results in the new lower bound 20 and the upper bound 80. The data extension does not interrupt the detection of control flow drifts as presented in [14]. The process history is used in Algorithm 1 in Sect. 3 to detect data drifts and append new process models that show no difference at the control flow, but at the data level.

For the algorithm the data structure `trace_map` is used. A trace_map represents key value pairs as a hash-map. Hereby, the trace id, e.g., "Process instance 1" is used as a key. Using such unique identifiers bears advantages regarding the look up time of certain values. The corresponding value would be the trace, that has been detected in the event stream. In addition, the data attributes are now stored as well for each event in the trace_map. In the following algorithms, this map is synthesised using an event stream. This stream has the advantage that every time an event is detected, it is processed immediately, so all the data elements of this event will be saved in the trace_map. To detect the currently relevant traces in an event stream, the sliding window approach is used. This means that only k traces are considered for the detection of drifts. If a new trace is detected and there are already k traces in the trace_map, the oldest trace is removed and the new trace is stored.

Concept/Data Drifts: [5] differentiates four types of concept drifts at control flow level. ① are incremental drifts, which consist of small changes to the business process model, like a new event or a removed event. ② are recurring drifts, which show typical seasonal effects, like in a hotel for example. The business process logic differs from winter to summer and alternates back to winter. ③ are gradual drifts, that represent a change in the business process logic, where process instances of the old logic are still being executed. ④ are sudden drifts, which are the direct opposite to ③, i.e., no process instances of the old logic are still being executed.

A concept drift cannot be a sudden drift and a gradual drift at the same time. All other combinations like a sudden recurring incremental drift, are possible. These concept drifts at the control flow level can also be defined and detected at the data level, which is explained in detail in Sect. 3. Concept drifts on the data level are called data drifts in this work.

The concept of process histories (cf. Definition 1) enables the detection of all four types of data drifts and of the point in time when they occurred in the process history, which is explained in the next section.

3 Detecting and Identifying Data Drifts

In this section, the synthesis of data-extended process histories as basis for detecting and identifying data drifts is elaborated.

3.1 Detecting Data Drifts

Assume a data-extended process history $H_P =< M_0, ..., M_n >$ as defined in Definition 1 with most current process model M_n and the corresponding trace_map. Following [14] the core idea of detecting drifts is to synthesize a new viable process model M_{n+1} in the data-extended process history in case changes to the data attributes have happened. The difference between M_n and M_{n+1} yields the data drift and its type. As basis, for each new event in the stream, the data attributes are checked for changes. In this work, changes in data attributes are detected based on outlier detection in the data attribute values. For this statistical methods will be used. However, it is not feasible to compare every new event to all previous events in all traces as this might be too complex and might lead to misleading results in terms of the drifts. Imagine that a change happened in one event and later the inverse change occurs. Considering all traces this change would not be detected as a drift. Hence, it is feasible to restrict the set of considered events and traces. In [14], the idea of using a sliding window on the traces has been proven promising and hence this concept will be applied for the synthesis of data-extended process histories in the following as well.

Algorithm 1 implements the core ideas of using a sliding window on the traces and outlier detection on the data attributes. As input an event stream, ES, a window limit k and the thresholds ϕ and κ are required. The thresholds are described in detail in the following paragraphs. The algorithm is used while synthesising a process history. A data drift is detected after the detection of a control flow drift; algorithms for detecting control flow drifts are provided in, e.g., [14]. At the beginning of Algorithm 1, process history H_P is an empty list and does not contain any process models. Also the trace_map which is used in the detection of data drifts does not contain any items in the beginning.

The sliding window technique, allows to identify currently significant traces for the detection of new viable models, where k is the maximal number of traces stored in the trace_map. The data extension uses the same window for detecting drifts in the data elements. Since outliers shall be detected, a certain amount of values for a specific data element, respectively a certain number of an events, needs to be detected for statistical analysis. The minimum number of events, κ, is user defined and a value between 0 and k, since it is, except for a loop, impossible to have more events stored, than there are traces in the trace_map.

After the event has been stored in the trace_map, the algorithm tries to detect a data drift. A whole new range of drift types is possible if a concept drift and a data drift occur simultaneously, which require a definition and an algorithm to be detected. This approach is beyond the scope of this paper.

If the process history contains at least one model, a copy of the current model and its events with attributes is created. At the start the `list_of_data_drifts` is an empty list and contains pairs of the drifting attribute and its corresponding event. If the current model of the process history contains the currently processed event, an iteration over the data attributes of this event starts. In this iteration, a denotes a data attribute of currently processed event e. The next expression checks, if a is an outlier to e of the current model.

For the outlier detection following methods are used. If the data attribute a contains continuous data, the data could be transformed into a normal distribution [8] and a range is calculated. We are using the box plot approach, since it is very distribution independent. The whiskers, here used as lower and upper bounds of limits for outlier detection, are placed at 1.5 times of the interquartile range below the first quartile and 1.5 times of the interquartile range above the third quartile. The implementation currently only supports continuous data. If the data attribute a does not contain continuous data, we use the likelihood. If for example, only 3 equally common values have been detected in the last model for this attribute in 50 events, and a new value occurs, its likelihood is lower than all of the known values. On the other hand, if there are 50 different values for one attribute in 50 events, it could be deduced that this attribute is arbitrary. A user input, defining the maximum distance between the new likelihood and the average likehood of choices, is used as threshold, to detect outliers for this. If this attribute is not in the last known model for event e, the outlier function automatically returns true.

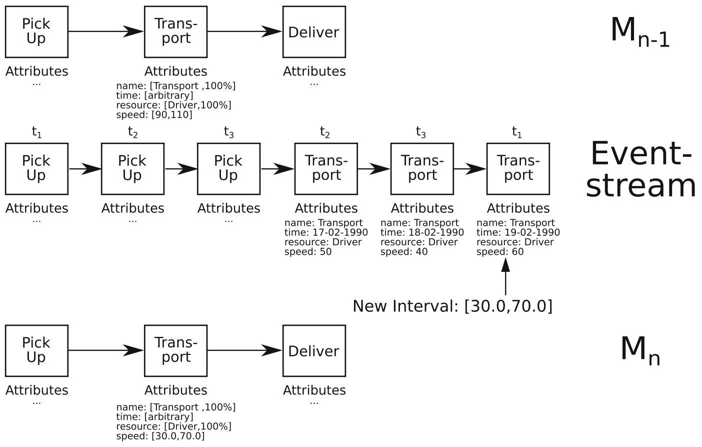

Fig. 3. Synthesising a process history with $\kappa = 1$ and $\phi = 1$

In the next step an empty list `list_a` is created and the variable `as` is initialised with 0. This variable counts how often event e is found in the trace_map containing a. The algorithm searches every trace in the trace_map. If an event is found that equals e and also has the same attribute a as an outlier, this attribute is added to the list.

If the number of occurrences for attribute a in the trace_map (`as`) is smaller than κ, a data drift has been detected. Apparently this data attribute is not used often enough to retrieve significant information and is removed from the new model. The pair e, a is appended to the list_of_data_drifts

Otherwise, the new range or likelihoods will be calculated using only the information of outlying attribute values. It is then counted how often an attribute of the trace_map fits the new properties and is divided by the number of events e. This yields a score value, which represents the percentage of fitting attributes for the new properties. If this score is greater or equal than ϕ, the new properties are added to the new model and the pair e, a is appended to the list_of_data_drifts. The threshold ϕ is in [0, 1], where 0 would be everything and 1 would be only considering scores, where 100% of the attributes match the new properties as a data drift. Afterwards, Algorithm 2 is executed, to detect the type of the data drift.

Figure 3, shows how an outlier is detected for the running example Fig. 1 and how and when a new model is appended. The range from 90 to 110 has been

```
Input: Event Stream ES (a series of events)
       k (Limit for number of trace_map items)
       $\kappa$ (Threshold for number of an attribute for consideration, [0,k])
       $\phi$ (Threshold for distinction of a new viable data range [0,1])
Result: Process History $H_P$ (contains all viable process models in chronological order.)
$H_P$ = [ ], trace_map<trace_id,trace> = 0
for e in ES do
    if trace_map contains_key e.trace_id then
        trace_map['e.trace_id'].append(e)
    else
        if trace_map.size $\geq$ k then
            trace_map.delete_oldest
        trace_map.insert(e.trace_id,e)
    detect_concept_drifts_based_on_workflow_drifts();
    if $|H_P| \neq 0$ then
        New_Model = $H_P$.last, list_of_data_drifts = [ ]
        if $H_P$.last.contains(e) then
            for a in e do
                if outlier($H_P$.last[e],a) then
                    list_a = [ ], as = 0
                    for t in trace_map.values do
                        for ev in t do
                            if ev == e and ev contains a then
                                as+=1;
                            if ev == e and outlier($H_P$.last[e],ev.a) then
                                list_a.append(ev.a);
                    if as < $\kappa$ then
                        if New_Model[e] contains a then
                            New_Model[e].remove(a);
                            list_of_data_drifts.append({e,a});
                        break;
                    else
                        e_size = 0; fitting_e = 0; properties = calc_properties(list_a);
                        for t in trace_map.values do
                            for ev in t do
                                if ev==e then
                                    e_size+=1;
                                    if !outlier(properties,ev.a) then
                                        fitting_e+=1;
                        score = fitting_e / e_size;
                        if score $\geq \phi$ then
                            New_Model[e].a.properties = properties;
                            list_of_data_drifts.append({e,a});
            if |list_of_data_drifts| >0 then
                $H_P$.append(New_Model)
```

Algorithm 1: Algorithm to synthesise a process history

detected earlier. In the event stream three new traces are occuring, each of them having an outlier in the event `Transport`. With a sliding window size of 3, only outliers are in consideration for new models. Each time an outlier is detected, a new range is calculated if there are more or equal κ outlier in the sliding window. When the third outlier is detected, this requirement is met and the new range from 30 to 70 is calculated. Each of the currently viewed speed values are fitting this range. A new model is appended to the process history.

3.2 Data Drift Identification

Algorithm 1 detects data drifts in an event stream and creates new models for the process history. Every time a new process model is appended to the process history a data drift is detected. The four types of data drifts, in relation to a process history, can be defined formally as follows:

Definition 2 (Data Drift Types). *Let U be a given set of unfinished traces. Moreover let H be a process history for a process P containing only data drifts, which can be easily filtered by checking if the list of data drifts in a Model M is $\neq \emptyset$. Let $H_{dd} \subseteq H$ be the models of the process history containing data drifts and for $M_n =< E, < (e_0, A_0^c), ..., (e_k, A_k^c) >> \in H_{dd}$ let $M_n.drifts$:= $< (e_0, A_0^c), ..., (e_k, A_k^c) >$ yield the list of event attribute pairs, containing the attributes which have shown the data drift. Let $\phi \in [0, 1]$, $\sigma \in [0, 1]$ be thresholds, the function outlier, defined for a model and a data attribute, yielding true or false and the function similarity, defined for two attributes of an event, ranging from 0 to 1. The following drift types are defined as follows:*

- *Incremental Drift if $|H| \geq 2 \ \wedge \ \exists \ (e, A) \in M_n.drifts, (A \not\subset M_{n-1}[e] \ \wedge A \subset M_n[e]) \ \vee \ (A \not\subset M_n[e] \ \wedge A \subset M_{n-1})$*
- *Recurring Drift if $|H| \ \geq \ 3 \wedge \ \ \exists \ m \in \mathbb{N}, 2 \leq m \leq n, M_{n-m} \forall \ (\{e, A\} \ \in M_n.drifts,$*
  $similarity(M_n[e].A, \ M_{n-m}[e].A) \ \geq \sigma$
- *Gradual Drift if $|H| \ \geq \ 2 \wedge \exists \ t \ \in \ U \ , \{e, A\} \in M_n.drifts, \ \neg outlier \ (M_{n-1}, t[e].A))$*
- *Sudden Drift if $\neg GradualDrift$*

As a fitness function the same technique as in [14] using conformance checking with only considering moves in the log [2] is used. The similarity function checks if the statistical properties are alike. For example, if the intervals have a tremendous overlap or the distribution of likelihoods is similar.

It should be noted, that in this definition of data drift types, only the sudden and the gradual drift are distinct. It is possible for a data drift to be an incremental drift and recurring drift at the same time, e.g., a new data attribute has been detected in comparison to the last model, but this data attribute is also available and similar to an even older model. In the following, Algorithm 2, is explained in detail and shows how to answer RQ3.

As input parameters a list of unfinished traces U for M_0, M_{n-1}, a process history H_P and σ are required. σ describes the threshold for determining if two statistical properties are alike and ranges from 0 to 1, where 0 determines any 2 properties as equal and 1 determines only exactly equal properties to be similar.

If there is only one process model in the new process history no data drift had happened. The first distinction is made between a gradual drift and a sudden drift. It has to be either of them, so if it is a gradual drift, it cannot be a sudden drift and vice versa. For this, the algorithm iterates over the list_of_data_drifts of the current model. If there is a trace out of U for which its attributes and events match an entry in the list and is not an outlier, if compared to the second to last model, M_{n-1}, a gradual drift is detected, because there are still unfinished traces, that corresponds to the older model. The outlier function is the same, like in Algorithm 1. The third position in the return vector is set to 1, which signals a

```
Input: Traces u (list of unfinished traces), H (Process History),
       $\epsilon$ (maximum error between similar statistical properties.)
Result: type_vector[0,0,0,0] (Positions represent Drifts [Inc,Rec,Grad,Sudden], 1
        represents this type of drift occurred. )
res = [0,0,0,0];
if $|H| \leq 1$ then
    return "Error: No drift"
// $M$ is an abstraction to directly access the models of $H$
for $e,a$ in $M.list\_of\_data\_drifts$ do
    for $t$ in $U$ do
        if $!outlier(M_{n-1},t[e].a)$ then
            res[2] = 1 //Gradual Drift
            break;
if $res[2] \neq 1$ then
    res[3]=1; // Sudden Drift
for $e,a$ in $M_n.list\_of\_drift\_events$ do
    if $(!(M_{n-1}[e].contains\ a)$ then
        res[0]=1; // Incremental Drift
    if $(!(M_n[e].contains\ a)$ then
        res[0]=1; // Incremental Drift
if $|H| \geq 3$ then
    for $m$ in $(M_0,M_{n-2})$ do
        bool found = false;
        for $e,a$ in $M_n.list\_of\_drift\_events$ do
            if $similarity(M_n[e].a,m[e].a) < \epsilon$ then
                found = true;
                break;
        if found then
            res[1]=1; // Recurring drift;
return res;
```

Algorithm 2: Algorithm to identify data drift.

detected gradual drift. Likewise it can be determined if it is not a gradual drift, a sudden drift is detected.

In the next step, the list_of_drift_events is again iterated. If an attribute is not found in the older model M_{n-1} or if an attribute is not found in the current model M_n, it can be deduced that the attribute has been added or removed respectively. This indicates an incremental drift, represented by a change the value of the first element to 1 of the return vector.

If there are at least 3 process models in the history, a recurring drift can be detected. If there is at least one model from M_0 to M_{n-2} where all attributes of the list_of_data_drifts are similar, a recurring drift is detected. The resulting vector is returned at the end containing the information on which data drift could been detected. In the next section, the two algorithms are evaluated on a real life log, using a log from the manufacturing domain.

4 Evaluation

For evaluating the approach, a prototypical implementation in Ruby [9] is used and applied on a real world process execution log from the manufacturing domain. The underlying process executes the manufacturing of small metal parts for different machines.

Algorithms 1 and 2, are integrated into the algorithms presented in [14], however the work at hand is completely independent of the existing work. The steps creating the trace_map and using a sliding window have been merged from the algorithm from [14] into Algorithms 1 and 2 to save computation time.

The log files from the real world example are stored in XES format and consist of 10 process instances containing 40436 events in total, but instead of being serialised in XML, the log files are serialised in YAML [4]. There is no information lost while transforming XML into YAML and vice versa. The process execution log has been transformed into an event stream. The models in the process history have been discovered, using the approach in [14].

For this evaluation we pick the event "AXIS/Z/aaTorque" and look at the data attribute "value". This event appears 4415 times in the log files in total and is numeric. The only available non-numeric data attributes in this log file, reflect either an enumeration, where only specific values are allowed or an arbitrary value, which lets us only detect the moment this attribute has been detected often enough to be in the process history, κ times.

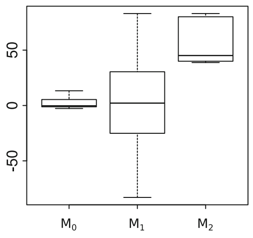

Fig. 4. Results reflecting the range of the torque value

The event "AXIS/Z/aaTorque" describes the positioning of the machine part in the z axis. With the sliding window k set to 200 and κ to 100, the first boxplot, seen in Fig. 4 (M_0), has been detected. For the outlier detection, the length of the whiskers, the interval $[-11.05, 15.28]$ has been calculated. Using 0.9 for the threshold ϕ, 2 data drifts have been detected, at the 123rd and 3187th time the event appeared in the event stream, shifting the boxplot to Fig. 4 (M_1) and (M_2) with the new intervals $[-105.10, 38.75]$ and $[-20.28, 89.99]$, two significant changes in the business process logic. This could be caused by a different part being produced in the machine using different values, or the replacement of a part of the machine where the new part is using new parameters.

Using a less strict drift detection threshold ϕ with 0.8, 8 drifts have been detected. The ranges of the intervals differ greatly, where only the fourth drift, when the event appeared for the 1795th time and the last drift, when the event appeared for the 2800th time could be suggested as a recurring drift. The intervals $[-105.10, 38.75]$ and $[-117.42, 91.08]$ overlap about 68%. Since there is always the same number of data attributes in the event, caused be the process execution engine which saved these logs, the only incremental drift is always the first on at the κth time the event occurs, since in the previous model the data attribute is absent. All of these drifts are gradual drifts, because the drift never occurred in the last appearance of an event of a process instance.

This evaluation was carried out with a proof of concept implementation to visualise and present data drifts in a data attribute of an event and the determination of its type. This procedure can be reproduced with any number of attributes of events, yielding a new model with adjusted statistical properties for the drifting attributes.

5 Summary and Outlook

This work introduces an extension to process histories to include data attributes and to detect and identify data drifts from event streams. Data drifts are part of the evolution of business process, therefore a data drift can be categorised into the four categories of concept drifts., i.e., incremental, recurring, gradual, and sudden. All four types can be detected and are formally defined. Two new algorithms have been presented. The first one synthesises a process history with data attributes. The other one allows to determine the type of data drift. The evaluation shows promising results. Based on a prototypical implementation and a real-world data set from the manufacturing domain it is possible to detect data drifts. The future work includes a more user friendly implementation of the algorithms and testing the algorithms on more data sets.

Acknowledgment. This work has been partly funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-072 and by the Austrian Research Promotion Agency (FFG) via the "Austrian Competence Center for Digital Production" (CDP) under the contract number 854187.

References

1. IEEE standard for extensible event stream (XES) for achieving interoperability in event logs and event streams. IEEE Std 1849–2016, pp. 1–50, November 2016
2. Van der Aalst, W., Adriansyah, A., van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **2**(2), 182–192 (2012)
3. van der Aalst, W.M.P.: Process Mining - Data Science in Action, 2nd edn. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
4. Ben-Kiki, O., Evans, C., Ingerson, B.: Yaml ain't markup language (yaml^{TM}) version 1.1. yaml.org, Technical Report, p. 23 (2005)
5. Bose, R.P.J.C., van der Aalst, W.M.P., Žliobaitė, I., Pechenizkiy, M.: Handling concept drift in process mining. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 391–405. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21640-4_30
6. Bose, R.J.C., Van Der Aalst, W.M., Zliobaite, I., Pechenizkiy, M.: Dealing with concept drifts in process mining. IEEE Trans. Neural Netw. Learn. Syst. **25**(1), 154–171 (2014)
7. Burattin, A., Sperduti, A., van der Aalst, W.M.: Heuristics miners for streaming event data. arXiv preprint arXiv:1212.6383 (2012)

8. Chen, S.S., Gopinath, R.A.: Gaussianization. In: Advances in Neural Information Processing Systems, pp. 423–429 (2001)
9. Matsumoto, Y., Ishituka, K.: Ruby programming language (2002)
10. Alves de Medeiros, A., Van Dongen, B., Van Der Aalst, W., Weijters, A.: Process mining: Extending the alpha-algorithm to mine short loops. Technical report, BETA Working Paper Series (2004)
11. Pauker, F., Mangler, J., Rinderle-Ma, S., Pollak, C.: centurio.work - modular secure manufacturing orchestration. In: BPM Industry Track, pp. 164–171 (2018)
12. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30409-5
13. Rozinat, A., Aalst, W.M.P.: Decision mining in business processes. Beta, Research School for Operations Management and Logistics (2006)
14. Stertz, F., Rinderle-Ma, S.: Process histories-detecting and representing concept drifts based on event streams. In: CoopIS, pp. 318–335 (2018)
15. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.: Event stream-based process discovery using abstract representations. Knowl. Inf. Syst. **54**(2), 407–435 (2018)