Flexible Batch Configuration in Business Processes Based on Events

Luise Pufahl, Nico Herzberg, Andreas Meyer, and Mathias Weske

Hasso Plattner Institute at the University of Potsdam
{firstname.lastname}@hpi.uni-potsdam.de

Abstract. Organizations use business process management techniques to manage their core business processes more efficiently. A recent technique is the synchronization of multiple process instances by processing a set of activities as a batch – referred to as batch regions, e.g., the shipment of goods of several order processes at once. During process execution, events occur providing information about state changes of (a) the business process environment and (b) the business process itself. Thus, these events may influence batch processing. In this paper, we investigate how these events influence batch processing to enable flexible and improved batch region execution. Therefore, we introduce the concept of batch adjustments that are defined by rules following the Event-Condition-Action principle. Based on batch adjustment rules, relevant events are correlated at run-time to batch executions that fulfill the defined condition and are adjusted accordingly. We evaluate the concept by a real-world use case.

Keywords: BPM, Batch Processing, Event Processing, Flexible Configuration.

1 Introduction

Companies strive to manage their core business in a process-oriented fashion to be efficient and stay competitive in the market. For this attempt, business processes are documented as process models [25]. These process models can also be used for process automation by a Business Process Management System (BPMS) [18]. Usually, the instances of a process, i.e., the concrete executions, run independently in existing BPMSs, e.g., [3,4,13]. However, efficient process execution may require bundled processing of activities of different process instances. Hereby, efficiency refers to costs savings under the trade-off of increasing average waiting times. For instance, in an hospital, a nurse transports multiple blood samples of patients to the laboratory at once instead of each separately to save transportation costs. To cope with this challenge, batch activities were introduced in business processes, e.g., in [1,14,20,21]. In these works, further application domains as, for instance, logistics and event organization, are discussed.

The recent concept of batch regions [20] enables the synchronization of process instances with similar characteristics for a set of activities. Thereby, several configuration parameters allow the process designer to individually setup the batch execution, e.g., rule-based activation of a batch. However, specifying the rules at design-time does not guarantee optimal process execution, since expected and unexpected events occurring

X. Franch et al. (Eds.): ICSOC 2014, LNCS 8831, pp. 63–78, 2014.

during process execution do influence the execution [10]. Reacting on these events and changing the specified configuration parameters is required for process improvement.

In this paper, we apply event processing techniques to flexibly adapt these configuration parameters at run-time to react in real-time on changes of the business process execution environment and improve the batch execution. The contributions of this paper are (i) to provide an overview about changes on batch configuration parameters triggered by events and (ii) to describe a framework that implements the flexible adaptation of configuration parameters triggered through event occurrence.

The paper is structured as follows. Section 2 introduces the concepts of batch regions and event processing before Section 3 presents a motivating example originating from a real-world scenario in the healthcare domain. It leads to an analysis on how events may influence batch execution and corresponding requirements in Section 4. Section 5 presents the concept of flexible adaptation of batch regions based on event processing techniques. In Section 6, the framework is applied to the healthcare scenario from Section 3 as evaluation. Section 7 is devoted to related work and Section 8 concludes the paper.

2 Foundation

Batch Region. A batch region comprises a connected set of activities. For batch processing configuration, a batch region contains four configuration parameters: (1) a grouping characteristic to cluster process instances to be processed in one batch based on attribute values of utilized data, (2) an activation rule to determine when a batch may be processed while balancing the trade-off between waiting time and cost savings, (3) the maximum batch size indicating the maximum number of entities to be processed, and (4) the execution order of the processed entities [20].

Each single execution of the batch region is represented by a batch cluster collecting – based on the grouping characteristic – a number of process instances for synchronization. Thereby, a batch cluster passes multiple states during its lifetime [21]. It is initialized (state *init*) upon request of a process instance. The batch cluster transitions to state *ready* (enablement), if the activation rule is fulfilled and is then provided to a resource that decides to start execution at some point in time. The execution is indicated by state *running*. If more than one resource is available, several batch cluster can be executed in parallel. After initialization and before execution start, process instances may still be added until the maximum batch size is reached (state *maxloaded*). Termination of all process instances being part of the batch cluster successfully *terminates* it.

Collecting multiple objects, e.g., blood samples, may also be done by utilizing loop or multi-instance structures as specified in the workflow patterns [2]. This requires to merge multiple process instances into one handling the synchronization. However, batch regions do not merge instances to retain the single instances autonomy outside the batch regions. This enables dynamic process instance assignment to batch clusters, e.g., for run-time cluster adjustments as discussed in this paper or for error handling.

Events. Information about changes or exceptions in the business process environment are provided by events. Often those events are not stored at one place, but in several information systems and BPMSs [7]. We refer to events being unstructured and available in an IT system as *raw events*. Event processing techniques help to utilize these

raw events and use them during process execution for process monitoring, adjustment, and control [5, 9, 10]. Structuring raw events according to a certain description referred to as *structured event type*, transforms raw events in a first step into *normalized events*. Normalized events are the basis for further processing by, for instance, combination, aggregation, and enrichment by context data [11]. We distinguish two event types being relevant for flexible batch processing: (a) *business events* and (b) *process events*. Business events base on normalized events enriched by business context information that are relevant for all running process instances. In contrast, a process event is correlated to a specific process instance and thus provides instance-specific information.

3 Motivating Example

The following healthcare process, the blood testing process in Fig. 1, is used to illustrate the need for flexible batch execution.

Instantiation of the process takes place, if there is a blood test required for a patient at the ward. First, the blood test order is prepared before a blood sample is taken from the respective patient. Afterwards, a nurse transports both to the laboratory, where the blood sample is first prepared for testing. Then, the actual test is conducted by a blood analysis machine. The laboratory possesses one machine for each type of blood test. As the blood analysis machines have an interface to the central hospital information system, the results are published so that they are accessible by the physicians in the respecting ward. There, they can evaluate the blood test result and can use it for diagnostic.

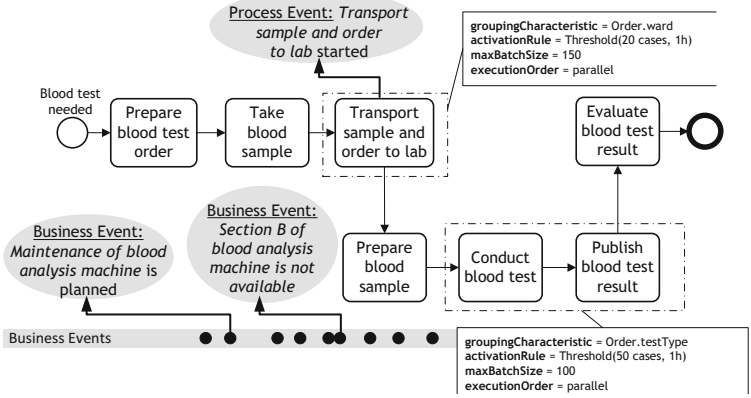

Fig. 1. Blood testing process

Within the given process, two batch regions are specified. As several blood test orders incur at a ward, the nurse would not bring each individually to the laboratory. In fact, a nurse delivers several blood samples together to save transportation cost which is captured by the first one comprising activity *Transport sample and order to lab*. The second batch region comprises activities *Conduct blood test* and *Publish test results* and enables to collect multiple blood samples before a test run on a blood analysis machine is started to save machine costs. So far, the configuration parameters are defined at design-time and can not be adapted at run-time. However, changes and exceptions

within the business process or in its execution environment might require adaptation. Following, we discuss three example events being of relevance for batch regions in the blood testing process:

Planned maintenance of a machine: This business event indicates that a maintenance of a machine is planned. During the maintenance, the machine is not available to conduct tests of the specific type. Blood samples in not yet running batch clusters might expire, because the waiting time of the collected process instances increases by the maintenance time. Thus, in such situations, the blood analysis should be started shortly before the maintenance takes place to avoid expired blood samples.

Partly unavailability of a machine: Assume, a blood analysis machine contains four sections to process blood samples from which one fails. Then, the capacity of the machine is reduced by one quarter. Hence, the maximum number of process instances allowed to be contained by a batch cluster should be reduced accordingly.

Transportation of a set of blood samples of the same type is started: Assume, the timeout is almost reached for a batch cluster while a transportation of blood samples to the laboratory requiring the same test is started. The respective batch cluster may delay its activation until the instances arrive to improve cost savings.

These examples show that there exist various situations requiring a flexible adjustment of predefined batch processing behavior in order to (1) reduce costs, (2) avoid increased waiting time, and (3) ensure correct batch execution, e.g., a reduced capacity of the task performer. Next, we perform an analysis to set the requirements before we present our concept in Section 5.

4 Events and Batch Regions

As discussed above, it is valuable for organizations to design batch processing in a flexible manner. Thus, created batch clusters may be adjusted according to the changes of the process environment as described by business events or process related aspects as described by process events. Adjustments refer to changes of the batch cluster configuration parameters. Table 1 provides an overview how the configuration parameters (1) groupingCharacteristic, (2) activationRule, (3) maxBatchSize, and (4) executionOrder can be adjusted at run-time. More precisely, the table discusses how a parameter can be changed (*type of change*), the influence a change has on a batch cluster and its assigned process instances (*influence*), and the types of events triggering a specific adjustment (*events indicating*) and gives corresponding event *examples*.

In Table 1, all types of adjustments are considered. Each configuration parameter always contains a value that can be also *undefined* for the first three parameters. Usually, the configuration of a batch cluster is adapted as reaction on an event. In the case of changing the grouping characteristic, existing batch clusters have to be canceled and the corresponding process instances need to be reassigned to new ones, because the data view of the existing clusters do not fulfill the new grouping characteristic. For example, grouping characteristic *Order.ward* results in batch clusters with data views *General Surgery* and *Endoscopic Surgery*. If the grouping characteristic is adjusted to *Order.section*, the data views above are not valid anymore. Thus, both batch clusters need to be canceled and their instances reassigned to a cluster with data view *Surgery*.

Table 1. Classification on how batch clusters can be changed and by which events

Configuration parameter	Type of changes	Influence	Events indicating	Examples
groupingCharacteristic	- aggregate - refine - restructure	- cancel existing batch cluster and assign process instances to new clusters	- need for aggregation or division of batch clusters or batch cluster restructuring	- if staff gets ill, a nurse has to organize the transport of two wards
activationRule	- adapt rule parameter - use a new rule	- adapt configuration of batch cluster	- change in availability of task performer/material - the arrival/delay of instances - change of process instance properties	- maintenance of machine - start of the transport of several samples - blood sample expires
maxBatchSize	- increase - decrease	- adapt configuration of batch cluster and, if necessary, remove process instances	- a change in the capacity of task performer, used resource etc.	- section of a machine is not available
executionOrder	- select other type of execution	- adapt configuration of batch cluster	- change of resource or resource type	- usage of a replacement machine acting differently

Reducing the maximum batch size may result in batch clusters exceeding the newly set limit. Then, newest assigned process instances are removed from the corresponding clusters and get assigned to other or new batch clusters accordingly. The concept introduced in the next section covers all changes of Table 1 including these special cases.

As described in Section 2, during a batch cluster's lifetime, it may pass the states *init* - *ready* - *maxloaded* - *running* - *terminated*. When a task performer starts execution of a batch cluster, it transitions to state *running*. From this moment, no adjustments shall be done on the respective batch cluster anymore. Therefore, we assume that batch clusters can only be adjusted in states *init*, *ready*, or *maxloaded*.

Having presented multiple types of changes according to the configuration parameters and their implications, we derive three requirements to implement above observations. First, at design-time, event types relevant for batch cluster adjustment need to be identified (R1). Then, at run-time, occurring events must be correlated to respective batch clusters (R2) and they need to be adjusted accordingly (R3).

5 Flexible Configuration Based on Events

In the following, we describe the basic idea of our approach by referring to the example introduced in Section 3. Afterwards, the newly introduced *batch adjustments* and their *batch adjustment rules* are described, before we explain a method for process instance reassignment and introduce an architecture for realizing the presented approach.

5.1 Basic Idea

We assume that events are observed by an event processing platform. If a relevant event is observed, the corresponding batch cluster gets adjusted accordingly, cf. Fig. 2. Our concept builds on structured events that are a derivation of an event object [16] consisting of an identifier, a timestamp, and some structured event content, e.g., a set of key-value-pairs or a tree-structure expressed in extensible markup language (XML). A structured event type describes a class of structured events that have the same format. Besides attributes specific for an structured event, a structured event type consists of some content description describing the structure of the event content of a structured event, e.g., by defining the attributes (keys) or by an XML schema definition (XSD).

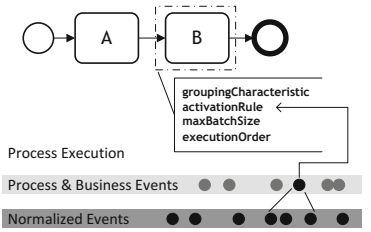

Fig. 2. Events influence the properties of batch clusters during run-time

We propose an approach that enables run-time flexibility of batch clusters by *batch adjustment*s following a *batch adjustment rule*. A *batch adjustment* is triggered by a certain event and may result in the adaptation of some parameters of one batch cluster. The events to react on, the conditions that need to be met, and the adjustments that may need to be applied are defined in the *batch adjustment rule*. The structure of a batch adjustment rule follows the (E)vent-(C)ondition-(A)ction principle originating from the database domain [6]. Events to react on are described by their event type, e.g., an event indicating the maintenance of a machine. The condition information enables the correlation of the event to the corresponding batch cluster, e.g., only the batch clusters containing process instances with blood samples for this machine. The described action specifies the particular adjustment of a batch cluster, e.g., the immediate execution.

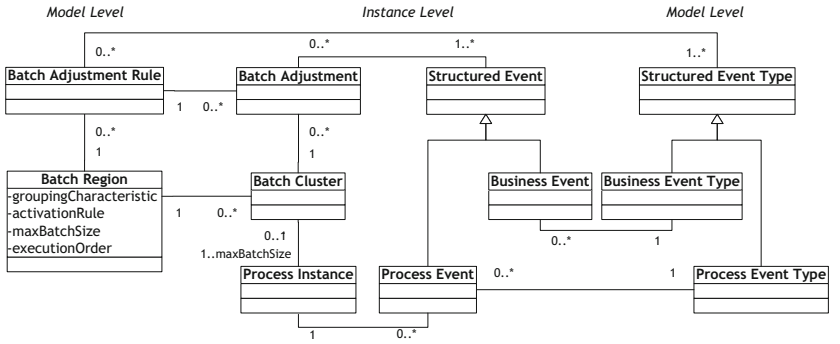

Fig. 3. Class diagram integrating batch region [20] and event processing [11] concepts. The model level shows the design-time concepts and the instance level shows their run-time implementation.

The connection of events and the batch region concept is illustrated in the class diagram of Fig. 3. One batch region can have an arbitrary set of batch adjustment rules which are provided by the process designer. A batch adjustment rule refers to at least one structured event type which can be a business or process event type. The structured

event types describe based on which events a batch adjustment is triggered. If a structured event occurs which is relevant for a set of batch clusters, then for each batch cluster one batch adjustment is created. Thus, a batch adjustment rule can have a arbitrary set of batch adjustments being related to one or several structured events, but each adjustment is assigned to only one batch cluster. During the lifetime of a batch cluster, it can be adapted by an arbitrary set of batch adjustments.

5.2 Batch Adjustment Rule and Batch Adjustment

For connecting batch clusters and events during process execution, we introduce the concepts of *batch adjustments* and *batch adjustment rules*. A batch adjustment rule, following the ECA-principle, describes how and under which conditions a batch cluster needs to be adjusted during run-time.

The events that need to be considered for an adjustment of a batch cluster are described by their event type. For example, a business event type describes the business events that indicate a planned maintenance of the blood analysis machine, cf. Listing 1.1. The event should be provided one machine analysis run before the maintenance starts so that not started batch clusters can be activated and finished before the maintenance start. This information is composed of fine-grained information of normalized events indicating the maintenance need and the schedule of the service technician.

The business event $machineMaintancePlanned_b$ contains information about the name of the corresponding machine. Further, it holds an ID and a timestamp as these are mandatory fields of structured events. The ID of the resulting business event is uniquely generated (getGUID()) and the timestamp is set to the actual time of creation (getTime(now)). The remaining data is collected from two normalized events $machineStatus_n$ and $technicianSchedule_n$ that need to be correlated. This is done by defining constraints in the WHERE-clause of the SELECT statement. In the example, it is checked whether the events target the same machine followed by a check for the maintenance need of the machine and the action of a planned maintenance by the service technician. As mentioned, the event shall be created exactly one machine run before the maintenance

```
1  machineMaintancePlanned_b.extraction =
2  { machineMaintancePlanned_b.id = getGuid();
3    machineMaintancePlanned_b.timeStamp = getTime(now);
4    SELECT
5      machineStatus_n.name,
6    FROM
7      machineStatus_n,
8      technicianSchedule_n
9    INTO
10     machineMaintancePlanned_b.MachineName
11   WHERE
12       machineStatus_n.name =
13       technicianSchedule_n.machineID AND
14       machineStatus_n.status = "MaintenanceNeeded" AND
15       technicianSchedule_n.state = "planned" AND
16       technicianSchedule_n.time - getTime(now) <= machine(name).getRuntime()}
```

Listing 1.1. Definition of the business event type $machineMaintancePlanned_b$ that captures the information about a maintenance in near future. This event results from events of the machine itself (event type $machineStatus_n$) and the technician schedule (event type $technicianSchedule_n$).

takes place. Thus, a time constraint is set to create the corresponding business event, if time until the maintenance is equal or lower to the time needed for a run of the machine (machine(name).getRuntime() returns the duration of a run of machine *name*).

This defined event type can be used as trigger for a batch adjustment rule that adapts the activation rule of batch clusters in case of a maintenance for avoiding expired blood samples. The proposed batch adjustment rule is shown in Listing 1.2, illustrating its basic structure. In the condition part of the batch adjustment rule, we ensure that batch adjustments are only created for batch clusters the event is relevant for. In our example, the events of type $machineMaintancePlanned_b$ are relevant for all batch clusters that are intended to run in time where the maintenance is planned to be conducted. Those should be started before the maintenance takes place to avoid unnecessary waiting times for the blood samples. The relevant clusters are those that have the same blood testing type as the blood analysis machine to be maintained and that are not yet enabled for execution, i.e., in state *init*. The instances of the *blood testing* batch region are grouped based on their blood test type (cf. Fig. 1) with the grouping characteristics = *Order.bloodTestType*. Thus, the batch cluster's data view provides information which blood test type its assigned process instances requires, e.g., *BC1(BloodTestA)*. The data view of the batch cluster can be used for the condition, cf. Listing 1.2 line 2 and 3.

```
1 EVENT {machineMaintancePlanned_b}
2 CONDITION batchCluster.dataView == machineMaintancePlanned_b.name
3 batchCluster.state == "INIT"
4 ACTION batchcluster.activationRule=Threshold (50,0h)
```

Listing 1.2. Definition of a batch adjustment rule to start batch clusters before a maintenance takes place.

Based on this example, we can observe that a specific batch cluster or a set of specific batch clusters for which an event is relevant can be identified based on batch cluster specific characteristics, i.e., (1) data view, (2) current state of the cluster, (3) number of instances contained in a cluster, and (4) type of instances. If no condition is described, a batch adjustment is created for all batch clusters which are in the *init*, *ready*, or *maxloaded* state. Clusters being already accepted by the task performer are not adapted anymore.

The last part of the batch adjustment rule is the definition of actions that need to be performed when an event happened and the conditions are fulfilled. These actions can use information of the underlying events to specify the adjustments of the particular batch cluster. Referring to our example, the action would be to enable the batch execution before maintenance, cf. Listing 1.2 line 4. With this action, the activation rule of the cluster is adjusted so that either 50 blood sample are triggered or the batch cluster waits 0 hours, meaning that the cluster is immediately enabled to be finished before the maintenance starts.

Batch adjustment rules are utilized to create batch adjustments for batch cluster. A batch adjustment holds the ID of the corresponding batch cluster and the action that need to be taken to change certain parameters of the batch cluster. Applying the batch adjustment rule of our example, a batch adjustment as shown in Listing 1.3 will be generated for batch cluster 1234.

```
1 batchCluster.id = 1234
2 batchCluster.activationRule = "Threshold(50,0h)"
```

Listing 1.3. Exemplary batch adjustment created for batch cluster 1234.

The batch adjustment mentioned above will replace the activation rule *Threshold (50,1h)* of batch cluster 1234 by *Threshold (50, 0h)*. With regards to the generation of batch adjustments, if an event is received, it is immediately checked whether this event is relevant for any available batch cluster. For each relevant cluster, a batch adjustment is created. In case that the event is valid for a certain time period, the event is stored. For each further initialized cluster, it is checked whether this event applies. Upon invalidation of the event, it is removed from the event storage. After presenting the structure of batch adjustment rules and the generation of batch adjustments, the next section discusses the special case where a batch cluster is not only adapted, but a reassignment of process instances is necessary.

5.3 Reassignment of Process Instances

A batch adjustment usually results in the adaptation of the configuration of one batch cluster. Sometimes, it also triggers (a) the reduction of instances contained by the batch cluster in case of a decreased *maxBatchSize* or (b) the cancellation of a batch cluster in case of a changing *groupingCharacteristic*. The extended lifecycle of batch clusters with the *canceled* state is shown in Fig. 4; a cancellation is only possible from states *init*, *ready*, and *maxloaded*. In both cases, process instances have to be reassigned to other or new batch clusters.

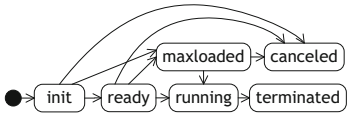

Fig. 4. Lifecycle of batch cluster extended by *canceled* state

In general, process instances that arrive at a batch region, i.e., the enablement of the entry activity into the region, are temporarily deactivated and assigned to a queue of the so-called *batch cluster manager* in the order of their arrival time (first-in-first-out). The batch cluster manager organizes the assignment of process instances to batch clusters and, if necessary, initializes new batch clusters.

If a process instance, in case of an adjustment, is reassigned, it should be handled prioritized, because it already experiences a longer waiting time than newly arriving instances at the batch region. Thus, the to-be reassigned process instance is placed in the front of the queue based on its arrival time at the batch region. Then, it is assigned to an existing or new batch cluster. In the example of Fig. 5, the number of instances of the batch cluster BC1 have to be reduced because an event indicated that a section of machine A is not

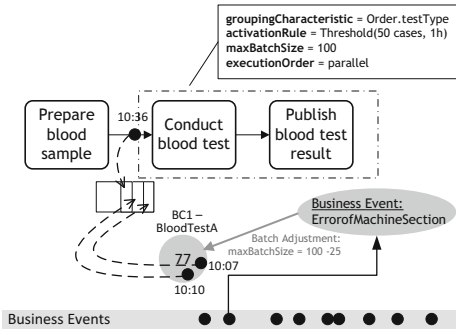

Fig. 5. Reassignment of process instances in case of a reduced *maxBatchSize*

working currently. Then, the newest assigned instances are removed from the size-reduced cluster. The process instance with the arrival time 10:07 is placed at the beginning of the queue, then the instance with 10:10 is added followed by the newly arrived instance at 10:36.

Often batch regions have an activation rule with a time constraint which describes the maximum waiting time for a process instance in a batch cluster. In the example process of Fig. 5, the threshold rule states that either 50 instances have to be available or the waiting time of 1h is exceeded to activate the batch cluster. For assuring the maximum waiting time also for reassigned process instances, we propose the usage of the batch adjustment concept here. If an instance is added to a batch cluster which was arrived at the batch region earlier than the batch cluster was created (or one of its instances), an event is created. This event triggers a batch adjustment which reduces the time constraint of the batch cluster by the difference between the batch cluster's creation time and the reassigned instance arrival time at the batch region.

5.4 Architecture

Next, we present an architecture showing details about a technical implementation to flexibly adapt batch cluster configurations. Fig. 6 presents the main components and their interactions as FMC block diagram [12]. The architecture is structured into three parts: event producer, *event processing platform*, and process control. The *process engine*, which controls process execution and batch handling, is an event producer and consumes event provided by the event processing platform at the same time. Besides the process engine, several event producers (*event sources*) can be connected via an appropriate *event adapter* to the event processing platform. These can be information systems as well as databases. The event processing platform normalizes the received raw events and creates business and process events based on defined rules. Event consumers are connected by an *event consumer interface*.

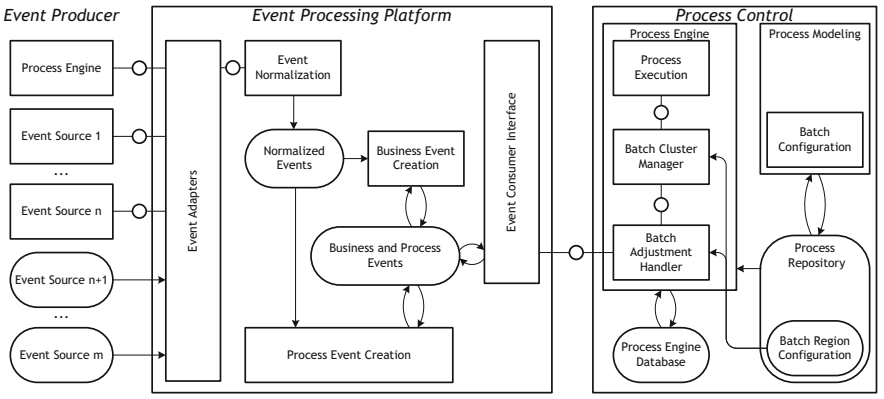

Fig. 6. Architecture to realize batch adjustments during process execution based on an event processing platform

Process control comprises the process engine and some modeling environment to create the process model to be executed within the process engine. After creation, a

process model is stored in the process repository. While modeling a process, batch regions can be designed. Thereby, the process designer can define batch adjustment rules used at run-time to adapt the batch regions. Those are saved together with the process model in the process model repository. During process execution, the process engine retrieves the process model and the adjustment rules from the repository. For each designed batch region, the *batch cluster manager* assigns the process instances to batch clusters. The *batch adjustment handler* registers for events that are specified in the batch adjustment rules of a batch region at the event consumer interface. If the handler receives a registered event from the event processing platform, then the event is evaluated and the according action is triggered for the appropriate batch clusters. The batch adjustment handler has an internal list of all batch clusters which are in state *init*, *ready*, or, *maxloaded* as these are the only ones that might be affected by events.

6 Evaluation

The approach is evaluated by showing its applicability to a real world use case: the blood testing scenario introduced in Section 3 with a simulation. As described, the laboratory uses a batch region to synchronize several blood samples for the blood analysis to save machine costs. The blood analysis machine needs to be maintained regularly respectively on request. Based on an event informing about the maintenance some time before it actually starts, the configuration of a running batch cluster can be adjusted. With the adjustment, the cluster is started in-time to decrease the number of expired blood samples due to unavailability of the machine. A blood sample expires after a certain time frame, often 90 to 120 minutes, because the blood structure changes. Then, the blood sample is not useful for medical analysis. Each expired blood sample causes costs of taking a new one.

Simulation Setup. For the evaluation, a simulation is used to compare the number of expired blood samples in case of normal batch execution, i.e., without run-time adaptations, to flexible batch execution as presented in this paper. Therefore, the laboratory part of the blood testing process was implemented as simulation[1]

with DESMO-J [8], a Java-based framework for discrete event simulation. The simulation starts with the arrival of process instances, i.e., blood samples, at the laboratory. Each process instance is terminated after finishing the blood test. At average, using an exponential distribution, every 12 minutes, a nurse brings 20 ± 5 blood samples (normally distributed) to the laboratory. For this simulation, we assumed that only one blood analysis machine exists. One run of the machine for analyzing blood samples takes 25 minutes. At maximum, the machine can handle 100 blood samples in one analysis.

For the simulation, the laboratory selected *ThresholdRule(50 instances, 1h)* as activation rule requiring 50 instances or a waiting time of one hour to enable a batch cluster (cf. Fig. 1). If a batch cluster fulfills this rule, it queues for being processed by the machine. The machine is already in use for a longer time period. Thus, twice a week, every 3.5 days with a deviation of 1 day, a maintenance is required. For the flexible batch handling, some time before the technician arrives, an event regarding the maintenance is

[1] The simulation source code and the reports of the different simulation runs are available at `http://bpt.hpi.uni-potsdam.de/Public/FlexibleBatchConfig`

provided. When the technician arrives, he is prioritized, but a current analysis on the machine is not interrupted.

Results. We conducted several simulation runs for two scenarios to compare the impact of flexible batch adjustments. The scenarios differ in the expiration time for blood samples: 120 minutes and 90 minutes. Fig. 7 and 8 summarize the results of the simulation runs over a period of two years, one diagram for each scenario. In both diagrams, we compare the results for maintenance times of 45 minutes and 60 minutes (intercept 2 and 3) with the result where no maintenance takes place (intercept 1). The black bars provide the numbers of expired blood samples, if (1) no adjustments are made at run-time. The different gray bars (2)-(4) show the results for event triggered batch adjustments, if the event is sent 1, 1.5, or 2 times the analysis run, i.e., 25, 37.5, or 50 minutes respectively, before the technician arrives.

If no maintenance would be conducted, 1,738 samples in scenario 1 and 19,913 samples in scenario 2 would expire due to exponential arrival of these blood samples and resulting waiting times for the machine. If the maintenance is conducted at average twice a week as indicated above, the number of expired blood samples increases by 14% respectively 29% for 45 and 60 minutes maintenance duration in scenario 1 (cf. black bars in Fig. 7) and they increase by 13% respectively 41% in scenario 2 (cf. black bars in Fig. 8).

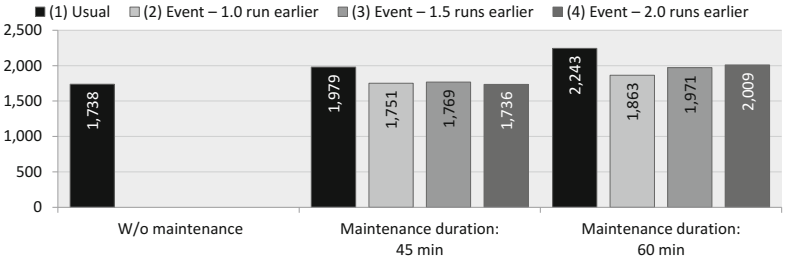

Fig. 7. Scenario 1 – 120 min expiration time: Number of expired blood samples in two years for different simulations

Applying flexible batch adjustments aims at reducing the number of expired blood samples. The recognition of the event indicating the maintenance directly activates all initialized batch clusters by changing the activation rule accordingly (cf. line 4 in Listing 1.2 in Section 5.2). The impact of the batch adjustment rule with respect to the point in time the event is sent is shown by the different gray bars (2)-(4). In 9 of 12 cases, we observe measurable improvements. The highest improvements for the different settings are mostly observed for the light gray bar ((2) Event 1.0 run earlier). It indicates that it is most beneficial for reducing the number of expired blood samples to inform about the maintenance one analysis run before the start of the maintenance. The improvement is at 13% respectively 20% in scenario 1 for 45 respectively 60 minutes maintenance time and at over 3% in scenario 2 (60 minutes maintenance). With these numbers, for scenario 1, we almost compensate for the maintenance.

For scenario 2, shown in Fig. 8, only slight improvements as well as two cases of no improvements are observed. This may be explained as follows: The arriving event enables a batch cluster which is then started for the blood analysis. During the analysis,

Fig. 8. Scenario 2 – 90 min expiration time: Number of expired blood samples in two years for different simulations

multiple new samples might arrive, but they are not processed before the maintenance as the technician is prioritized. Due to the small expiration time of 90 minutes, there is a good chance that those samples expire. For a maintenance time of 45 minutes, all samples which arrive 5 minutes after the start of the flexibly enabled cluster expire, because they have at least 20 minutes waiting time before the maintenance plus 45 minutes maintenance time plus another 25 minutes analysis time summing up to at least 90 minutes. For 60 minutes maintenance, all samples arriving at least 20 minutes after the start of the flexibly enabled cluster will expire. Thus, if – due to the arrival distribution of the blood samples – many samples arrive within these time frames, also negative results can be observed.

Summarizing above observations, it is important to check the relation between expiration time as well as waiting and maintenance times to decide whether to apply batch adjustments or not. In case, the relations are appropriate as, for instance, in scenario 1, applying batch adjustments provides reasonable and measurable improvements.

The simulation results indicate that the waiting time for the technician slightly increases, in average less than a minute. Due to limited space, the reader is referred to our simulation reports (see footnote 1). If, we take scenario 1, the cost savings due to reductions in expired blood sample will be higher than the technician costs due to small increases in the waiting time.

In most cases, we can observe that the number of zero-waitings increases, because starting an analysis run shortly before the technician arrives, increases the chance that the run is terminated just upon arrival. However, sometimes a run may only be started shortly before the technician's arrival as some other analysis run was still busy. Then, the technician must wait longer resulting in a higher distribution of waiting times and a higher total average waiting time.

7 Related Work

In the business process research domain, few works exists to synchronize the execution of multiple instances. For example in [1,14,23], the integration of batch processing into process models is discussed. These works provide limited parameters to configure the batch execution at design-time, often only the maximum capacity. This also limits opportunities to conduct adjustments at run-time. [23] provides some means for flexible run-time batch control by introducing batch activation by user invocation. Extending

the options for batch configuration in business processes, [21] introduces batch activities with three configuration parameters: capacity as the ones above as well, rule-based activation generalizing the user invocation based on rules, and execution order. One step forward, [20] extends the parameters by the grouping characteristic to distinguish process instances. However, all these works focus on specifications at design-time and do not support automatic adjustments of the batch configuration at run-time, for instance, due to changes in the process environment or within the process itself. In this paper, we extend the concepts presented in [20, 21] to allow run-time flexibility in terms of configuration adaptation to improve batch processing in business processes. We utilize events as trigger for taking adjustment actions. These extensions can also be applied to other works for adapting the configuration parameters offered there.

Batch processing flexibility has also been discussed in other domains as, for example, the manufacturing domain [17]. Here, batch scheduling is used to schedule a number of available jobs on a single or on multiple machines for saving set-up costs. Changes of market factors, e.g., a canceled order, or on the operational level, e.g., breakdowns, require a rescheduling functionality. In [17], an overview of suitable algorithms is presented and the need for a framework which combines possibly occurring events with some reschedule action is discussed. The contributions of this paper can be adjusted to offer a first approach in this direction: instead of configuration parameter adjustments, rescheduling action can be used in the batch adjustment rule.

Adoption of process instances during run-time is a widely discovered field. [22] discusses manual ad-hoc changes of single instances, e.g., to insert, delete, or shift activities according a given process model. This provides flexibility for single process executions but this does not provide possibilities to pool several process instances and to work on them as a batch. The CEVICHE framework [9] allows to change process instances automatically during run-time. Similar to this paper, it uses Complex Event Processing (CEP) to detect changes and exceptions which then trigger dynamic adaptation of the BPEL processes. In the same vein, [5] discusses means to integrate CEP with BPMSs on architectural level and shows how to do this for a BPEL engine. [24] introduces an approach to discover deviations of process executions and the underlying process model by using CEP techniques.

In this paper, we use CEP techniques as, for example, described in [7, 15], to create the necessary business and process events. [7] lists definitions for CEP-related terms, e.g., event type, that are used in this paper. Based on these works, a framework for CEP for business processes was introduced [10, 11]. We utilize this framework to allow dynamic batch activation and configuration rule adaptations as presented in Section 5. In this paper, we deal with comparably simple rules to correlate events to each other, to process instances, and to batch clusters. Applying common correlation techniques extends the correlation capability of the presented approach. One of these techniques, the determination of correlation sets based on event attributes, is introduced in [19].

8 Conclusion

In this paper, we showed the necessity to synchronize multiple cases in batch clusters and the requirement of their flexible adjustments during run-time. Therefore, a concept is introduced to apply event processing to batch execution allowing to flexibly

adjust batch configuration parameters and batch activation based on run-time changes represented by events. Based on the principle of Event-Condition-Action rules, relevant events are identified and then compared to defined conditions. If the conditions are fulfilled, the configured actions are executed as a batch adjustment for the corresponding batch cluster. Further, an architecture is presented showing details about a technical implementation and the components that are necessary to apply the concept within a process engine. We showed applicability of the introduced concept of batch adjustments during run-time with a real-world use case of a blood analysis in a hospital's laboratory. We simulated two years of work in the laboratory and showed that the application of the presented concept compensates for maintenance interruptions decreasing the blood expiration rate by at most 7%. With integrating more information about the process environment, e.g., the availability of resources, the presented concept can be extended. Further, techniques to ensure that batch adjustments do not lead to inconsistencies should be developed. We will investigate this topic in the future.

References

1. van der Aalst, W., Barthelmess, P., Ellis, C., Wainer, J.: Proclets: A Framework for Lightweight Interacting Workflow Processes. IJCIS 10(4), 443–481 (2001)
2. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
3. Activiti: Activiti BPM Platform, `https://www.activiti.org/`
4. Bonitasoft: Bonita Process Engine, `https://www.bonitasoft.com/`
5. Daum, M., Götz, M., Domaschka, J.: Integrating CEP and BPM: How CEP Realizes Functional Requirements of BPM Applications (Industry Article). In: DEBS, pp. 157–166. ACM (2012)
6. Dayal, U.: Active Database Management Systems. In: JCDKB, pp. 150–169 (1988)
7. Etzion, O., Niblett, P.: Event Processing in Action. Manning Publications Co. (2010)
8. University of Hamburg, D.o.C.S.: DesmoJ - A Framework for Discrete-Event Modeling and Simulation, `http://desmoj.sourceforge.net/`
9. Hermosillo, G., Seinturier, L., Duchien, L.: Using Complex Event Processing for Dynamic Business Process Adaptation. In: SCC, pp. 466–473. IEEE (2010)
10. Herzberg, N., Meyer, A., Weske, M.: An Event Processing Platform for Business Process Management. In: EDOC, pp. 107–116. IEEE (2013)
11. Herzberg, N., Weske, M.: Enriching Raw Events to Enable Process Intelligence - Research Challenges. Tech. Rep. 73, HPI at the University of Potsdam (2013)
12. Knöpfel, A., Gröne, B., Tabeling, P.: Fundamental Modeling Concepts: Effective Communication of IT Systems. Wiley (2005)
13. Lanz, A., Reichert, M., Dadam, P.: Robust and flexible error handling in the aristaFlow BPM suite. In: Soffer, P., Proper, E. (eds.) CAiSE Forum 2010. LNBIP, vol. 72, pp. 174–189. Springer, Heidelberg (2011)
14. Liu, J., Hu, J.: Dynamic Batch Processing in Workflows: Model and Implementation. Future Generation Computer Systems 23(3), 338–347 (2007)
15. Luckham, D.: The Power of Events. Addison-Wesley (2002)
16. Luckham, D., Schulte, R.: Event Processing Glossary - Version 2.0 (July 2011), `http://www.complexevents.com/wp-content/uploads/2011/08/EPTS_Event_Processing_Glossary_v2.pdf`

17. Méndez, C.A., Cerdá, J., Grossmann, I.E., Harjunkoski, I., Fahl, M.: State-of-the-art review of optimization methods for short-term scheduling of batch processes. Computers & Chemical Engineering 30(6), 913–946 (2006)
18. Meyer, A., Pufahl, L., Fahland, D., Weske, M.: Modeling and Enacting Complex Data Dependencies in Business Processes. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 171–186. Springer, Heidelberg (2013)
19. Motahari-Nezhad, H.R., Saint-Paul, R., Casati, F., Benatallah, B.: Event Correlation for Process Discovery from Web Service Interaction Logs. VLDB Journal 20(3), 417–444 (2011)
20. Pufahl, L., Meyer, A., Weske, M.: Batch Regions: Process Instance Synchronization based on Data. In: EDOC. IEEE (2014) (accepted for publication)
21. Pufahl, L., Weske, M.: Batch Activities in Process Modeling and Execution. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 283–297. Springer, Heidelberg (2013)
22. Reichert, M., Dadam, P.: Enabling Adaptive Process-aware Information Systems with ADEPT2. In: Handbook of Research on Business Process Modeling, pp. 173–203. Information Science Reference (2009)
23. Sadiq, S., Orlowska, M., Sadiq, W., Schulz, K.: When Workflows Will Not Deliver: The Case of Contradicting Work Practice. BIS 1, 69–84 (2005)
24. Weidlich, M., Ziekow, H., Mendling, J., Günther, O., Weske, M., Desai, N.: Event-based monitoring of process execution violations. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 182–198. Springer, Heidelberg (2011)
25. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Second Edition, 2nd edn. Springer (2012)