Monitoring Batch Regions in Business Processes

Tsun Yin Wong, Susanne Bülow(✉), and Mathias Weske(✉)

Hasso Plattner Institute at the University of Potsdam, Potsdam, Germany
susanne.buelow@student.hpi.uni-potsdam.de, mathias.weske@hpi.de

Abstract. Recently, batch activities have been introduced to improve the execution of business processes by collectively performing batch activities that belong to different process instances. Using traditional techniques to monitor processes with batch activities leads to inadequate representation of process instances, since monitoring is unaware of batch activities. This paper introduces an approach to monitor batch activities, which also takes into account exceptions in batch clusters at different levels of abstraction. The concepts and techniques introduced are evaluated by a prototypical implementation using real-world event data from the logistics domain.

1 Introduction

Many organizations in business and administration represent their working procedures as business processes to improve them and to monitor their execution [1]. Recently, batch activities [2] and batch regions [3] have been proposed to collectively execute activities of different process instances. While methods and techniques for monitoring individual business processes have been proposed, these are inadequate to monitor batch activities. This paper introduces novel concepts and techniques for monitoring batch activities, which also take into account exceptions. The approach is evaluated by a prototypical implementation using real-world event data from the logistics domain.

A batch region [3] of a process model consists of activities that are executed collectively as a batch. We find batch activities in many domains, including health care (many blood samples are analyzed in a batch) and logistics (containers in a vessel are transported together). Since processes are performed non-automatically in these environments, process monitoring uses events that occur while the process is being executed. Events include the arrival of a vessel in a harbor with certain containers or the completion of a blood sample analysis in a hospital.

If traditional techniques for process monitoring were used in these settings, the number of monitoring events would be overwhelming. Using information about batch regions, the number of events to monitor can significantly be reduced. Furthermore, monitoring approaches need to expose the occurrence of irregular

The research leading to these results has received funding from the European Union's Seventh Framework Program (FP7/2007–2013) under grant agreement 318275 (GET Service).

A. Persson and J. Stirna (Eds.): CAiSE 2015 Workshops, LNBIP 215, pp. 317–323, 2015.
DOI: 10.1007/978-3-319-19243-7_30

behaviour, such as exceptions. Therefore, we also provide a classification of different types of exceptions of business processes, involving individual process instances and all process instances in a batch, respectively.

The remainder of this paper is organized as follows: First, the need for batch monitoring is illustrated by a motivating example in Sect. 2. Then, a conceptual approach of batch monitoring is introduced in Sect. 3. In Sect. 4, the prototypical implementation is explained. In Sect. 5, we use the batch monitoring approach for the motivating example. Finally, Sect. 6 concludes this paper.

2 Motivating Example and Requirements

To exemplify the approach, we introduce a real world use case inspired by the GET Service project[1], which is funded by the Seventh Framework Program of the European Union. GET Service aims at supporting efficient transportation planning to reduce both transportation times and empty miles, leading to a reduction of CO_2 emission.

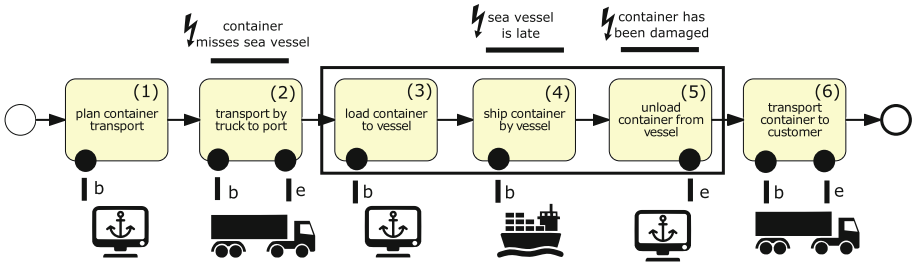

Fig. 1. Process from logistics domain. Events from various sources are related to monitoring points (begin event of activity denoted as 'b', end event denoted as 'e').

The respective process model is shown in Fig. 1; it consists of six sequential activities: At first, the transport planner schedules a container for transport (activity 1). The container is later picked up at the warehouse and transported to the port by truck (2), where the container is loaded on a sea vessel (3). The container is then shipped to another port (4), where it is unloaded (5). Finally, the container is transported to the customer by another truck (6).

As a sea vessel transports multiple containers at a time, activities (3), (4) and (5) form a batch region. Each container is represented by a process instance, whereas a sea vessel is represented by a batch cluster containing all process instances of the containers on the sea vessel. To facilitate process monitoring, we assign monitoring points to each activity, defining its begin and its end event. The set of monitoring points for the given use case is limited to the corresponding real-world events provided by port community systems (1) (3) (5), transport companies (2), (6) and shipping companies (4), resp.

[1] http://getservice-project.eu.

In the use case, three exceptions may occur:

- *Container misses sea vessel*: A container arrives with excessive delay at the port of origin and cannot be transported on the sea vessel for which it was scheduled.
- *Sea vessel is late*: The calculated arrival of the ship is after the planned arrival.
- *Container has been damaged*: During the unloading of the containers, customs notice that the container is unsealed and therefore needs further inspection.

From our scenario, we can identify two main requirements for batch monitoring:

R1. In the traditional process monitoring approach, events indicate information about single process instances. In our use case, each container transport would represent one process instance and events about each container would be monitored individually. However, as soon as the container is loaded onto the sea vessel, it would be sufficient to be updated about the progress of the vessel, instead of the progress of the hundreds of containers on the vessel. To enable monitoring of the vessel as a batch cluster containing several process instances, the events arriving for each container must be aggregated. To enable monitoring of a batch cluster, we therefore need a *batch aggregation strategy* for the events on the process instance level.

R2. Exceptions occurring in batch regions need to be handled differently than exceptions during normal process executions. For example, the exception "Ship is late" would normally result in one exception for each container on the ship. In batch monitoring, it would be sufficient, to mark the sea vessel as having an exception. On the other hand, the exception "Container has been damaged" detected for a container should not result in an exception of the whole vessel, but only in an exception for the affected container. Thus, a handling for different *batch exceptions* has to be examined.

3 Batch Monitoring Approach

In this section, an approach for batch monitoring is introduced. Processes are monitored using technical representations of real world happenings, so called *events*. Each event has an event type that defines its structure [4]. A *monitoring point* is a binding of an event type to an activity of a process model. Monitoring points are used to measure the progress of process instances based on events [5].

A *batch region* is a coherent part of a process model with a single entry, in which several process instances with similar characteristics are executed together as *batch clusters*. The assignment of an instance to a cluster is defined by the grouping characteristic of the batch region [3]. *Exceptions* indicate an erroneous execution of a process instance. Several exception types on different levels of a process can be distinguished [6].

Based on this preliminary work, the novel approach for batch monitoring is described, covering requirements R1 and R2 from Sect. 2. To allow monitoring of batch clusters (R1), we introduce two *batch aggregation strategies* for process instance events:

- *Complete Event Set Strategy*: Only if events for all process instances of a batch cluster have been observed, the cluster progress will be recognized. This is a cautious approach that needs additional exception handling in case of missing events.
- *Single Event Strategy*: The first event connected to one process instance within a batch cluster determines the cluster progress. We here assume that the correct execution of one instance directly implies the correct execution of the whole cluster. For our implementation, we chose this approach.

As far as *batch exceptions* (R2) are concerned, those have to be differentiated in exceptions outside of a batch region, which would be normal *process exceptions* and exceptions within a batch region. Moreover, we consider the following two levels for exceptions in a batch region:

- *Batch-level Exceptions*: If the exception affects the whole batch cluster, namely all contained process instances, it is an exception on batch level.
- *Instance-level Exceptions*: If the exception affects only one process instance, this instance is then in exception and cannot be further executed together with the remaining, correct process instances in the batch cluster. It is therefore removed from the cluster and has to be handled separately. This is an exception on instance level.

4 Batch Monitoring Tool

In this section, we present the prototypical implementation of the batch monitoring approach. An overview of the system architecture is presented in Fig. 2. It contains three main components. The *ProcessConfiguration* is accessed by the *Frontend* to create the monitoring points and batch regions as part of a process model. *Monitoring* includes the monitoring of process instances, batch clusters (as described in R1 of Sect. 2) and exceptions (as described in R2) using monitoring points and batch regions specified in the *Frontend*. It communicates with the Event Processing Platform (UNICORN[2]) introduced in [7] that consumes events provided by process engines executing the process model. Information about process instances and batch clusters are propagated to the *Frontend*. The *Frontend* offers an intuitive visualization of the progress of process instances as well as batch clusters and visualizes occurring exceptions using the information propagated from the *Monitoring*. The visualization is limited to sequential processes as the one introduced in Sect. 2. However, the monitoring concept is applicable to all well-formed process models.

The tool is implemented in Java 7, using the Apache Wicket Framework for the frontend. For process import of BPMN-alike signavio.xml-files, we use the libraries jBPT[3] and promniCAT[4]. The ability of this tool for batch and exception

[2] http://bpt.hpi.uni-potsdam.de/UNICORN.
[3] https://code.google.com/p/jbpt/.
[4] https://code.google.com/p/promnicat/.

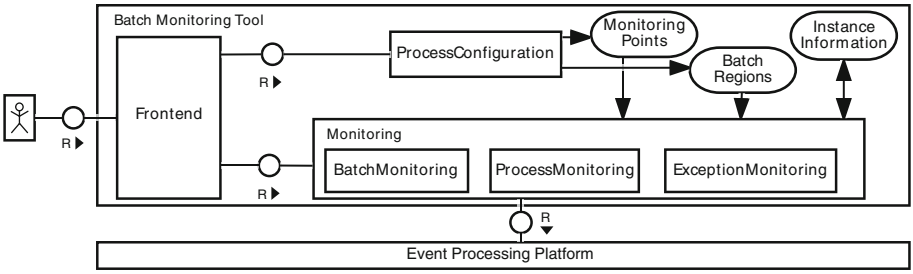

Fig. 2. System architecture of batch monitoring tool

monitoring is demonstrated with an example in Sect. 5 as well as in a screencast of the tool[5], where historic real world data is used to simulate occuring events.

5 Example Use Case

Figure 3 provides a screenshot of the batch monitoring tool, showing an example execution of the scenario described in Sect. 2. Each table row refers to a batch cluster or a process instance. The current activity is indicated by the column in which it is located, i.e. it moves to the right during execution. A circle denotes that the activity is in execution, a green tick marks the completion of the activity. Instances of a cluster are visible only when the cluster is selected (see Fig. 4).

When a freight transport company schedules a container for transport, it also arranges the sea vessel to export its container. On this basis, we identify process instances of the same batch cluster by taking the scheduled sea vessel as the grouping characteristic of the batch region. In Fig. 3, the transport planning for *Container 12* is ongoing. *Container 10* and *Container 11* that have arrived at the port are expected to be grouped in the same batch cluster. The status of a sea vessel progresses as soon as one container of the cluster is updated (follows R1 from Sect. 2).

The three exception types described in Sect. 3 cover the exceptions in our use case (follows R2).

- *Container misses sea vessel*: The corresponding process instance must be removed from the batch cluster for which it was planned. In our example, this applies to the process instance regarding *Container 3*.
- *Sea vessel is not moving*: The whole batch cluster *Sea vessel 4* containing five process instances is affected as shown in Fig. 4. The exception is triggered by an event of the type *ShipNotMoving* bound to the activity *ship container by vessel*.
- *Container has been damaged*: The process instance regarding *Container 2* is removed from its batch cluster and remains at the port.

[5] https://owncloud.hpi.de/public.php?service=files&t=f02387692aaa880428905d30e3f9ab89.

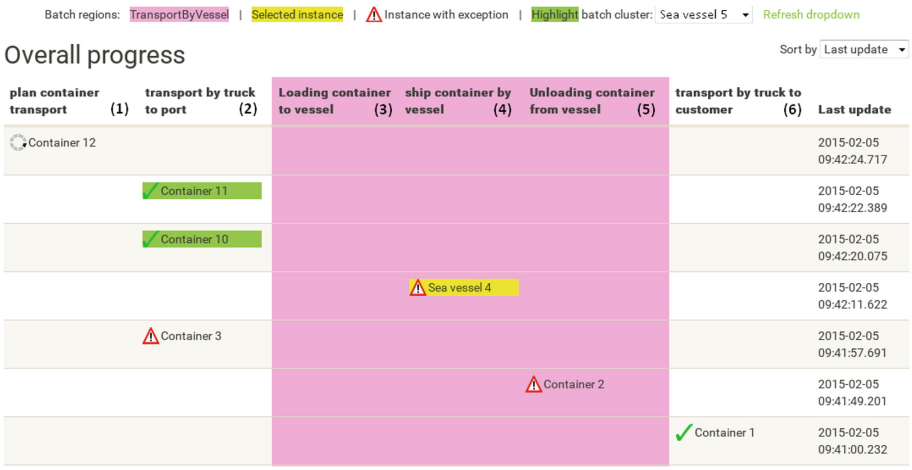

Fig. 3. Visualization of process instances and batch clusters. Containers are represented by process instances, whereas a sea vessel is represented by a batch of all containers on that sea vessel.

The example shows how the batch monitoring concept allows monitoring of single containers, but also of sea vessels containing several containers and their exceptions.

Fig. 4. Details of batch cluster *Sea vessel 4* with five process instances. Its exception has been triggered by an event of type *ShipDelay*.

6 Conclusion and Future Work

In this paper, we have presented an approach with the corresponding implementation which enables the monitoring of batch executions, including their exceptional behaviour. The progress monitoring is driven by monitoring points triggered by events; a direct interaction with our tool to handle exceptions is not in its scope.

As of now, a BPMN process model is loaded into the monitoring tool and then complemented with monitoring points and batch regions afterwards. Future work includes the support of annotations in XML files for BPMN process models as mentioned in [8].

The batch concept presented in [9] includes the application of threshold rules and Event-Condition-Action (ECA) rules. They are currently not considered in our concept and we intend to integrate them to enable the detection of exceptions such as the exceeding of batch clusters.

Since the concept of the monitoring tool is loosely based on workflow exception patterns [6], research in how these patterns are supported in batches is required.

References

1. Weske, M.: Business Process Management: Concepts, Languages, Architectures., 2nd edn. Springer, Heidelberg (2012)
2. 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)
3. Pufahl, L., Meyer, A., Weske, M.: Batch Regions: Process Instance Synchronization based on Data. Universitätsverlag Potsdam (2014)
4. Etzion, O., Niblett, P.: Event Processing in Action. Manning Publications Company, Greenwich (2010)
5. Herzberg, N., Weske, M.: Enriching raw events to enable process intelligence - research challenges. Technical report 73, Hasso Plattner Institute at the University of Potsdam (2013)
6. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Workflow exception patterns. In: Martinez, F.H., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 288–302. Springer, Heidelberg (2006)
7. Bülow, S., Backmann, M., Herzberg, N., Hille, T., Meyer, A., Ulm, B., Wong, T.Y., Weske, M.: Monitoring of business processes with complex event processing. In: Lohmann, N., Song, M., Wohed, P. (eds.) BPM 2013 Workshops. LNBIP, vol. 171, pp. 277–290. Springer, Heidelberg (2014)
8. Baumgrass, A., Herzberg, N., Meyer, A., Weske, M.: BPMN Extension for Business Process Monitoring. In: Enterprise Modelling and Information Systems Architectures, GI (2014)
9. Pufahl, L., Herzberg, N., Meyer, A., Weske, M.: Flexible batch configuration in business processes based on events. In: Franch, X., Ghose, A.K., Lewis, G.A., Bhiri, S. (eds.) ICSOC 2014. LNCS, vol. 8831, pp. 63–78. Springer, Heidelberg (2014)