Events in BPMN: The Racing Events Dilemma

Sankalita Mandal

Hasso Plattner Institute at the University of Potsdam, Germany
sankalita.mandal@hpi.de

Abstract. Today, business process management is a key for companies to represent their operations using business process models. These business processes are executable using process engines. The process engines can produce and consume events for the completion of the processes. However, to receive the external events, we must rely on outer world sources such as a weather API, a traffic agency, an email from a different organization etc. While the digital world makes these message exchanges very convenient, there might still be some latency between the generation of a message and the detection of that message in a receiving process. This latency between the occurrence time and detection time of an event can cause a dilemma of choosing among the alternative paths triggered by racing events and might lead to wrong execution of a process. This problem is investigated in this paper. Also, some solutions are proposed to mitigate the consequences.

1 Introduction

Business processes according to BPMN 2.0 [1] consist of several activities, events and gateways. Events are something that happens in a specific context [2]. These events can be produced or consumed during process execution. Now, in a distributed setup, it can happen that an event has already occurred but the notification of that event occurrence is yet to be received. This may happen due to several manual or system latency. If we think about an application procedure where candidates mail their documents and an administrator enters the information into the system, then it is obvious that the application reception time will be different in the system than the actual reception time via post. In another scenario, when we transfer money via some online payment method, though the payment is done from our side, it may take several minutes or even days to reflect the transferred money in the system of the recipient.

The discrepancy between the occurrence time and the detection time of an event can cause dilemma for choosing among the alternative paths following event occurrences. This paper introduces racing events dilemma in detail and discusses the significance of it, rather the problems that can arise from it. Using two processes for registering to a workshop and paying for the workshop, it is shown that timestamp discrepancy can cause incorrect execution of processes. This may have a negative impact on time, money, user experience or other valuable resources. Possible solutions to mitigate the dilemma are also discussed.

O. Kopp, J. Lenhard, C. Pautasso (Eds.): 9th ZEUS Workshop, ZEUS 2017, Lugano, Switzerland, 13-14 February 2017, published at http://ceur-ws.org/

2 Foundation

This section introduces the relevant concepts helpful to follow the rest of the paper. Namely, we talk about business processes consisting of activities and events, especially the racing events and discuss the emerging field of event processing.

2.1 Business Process Model

A business process is a sequence of activities performed in an organizational context where all these activities collectively achieve a business goal [3]. The activities and their coordination are visualized with business process models. Nowadays, BPMN 2.0 is the de facto standard for modeling business processes. Each business process model can be considered as a blueprint for several process instances. An activity model can be instantiated for a set of activity instances.

An activity instance goes through different state transitions as shown in Fig. 1. When a process instance is started, each activity in the process is initialized and is in state *init*. As soon as the incoming flow of an activity is triggered, the instance is in state *ready*. The state changes to *running* when the activity starts execution. Finally, the activity goes to *terminated* state. If the activity instance is *not started* and process instance follows a different path, then it directly goes to *skipped* state. Also, due to the occurrence of an attached boundary event, a running activity instance can be *canceled*.

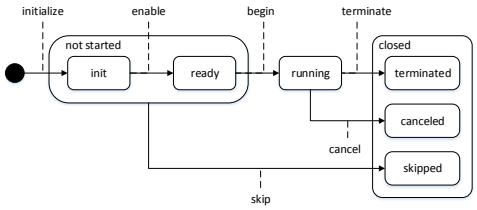

Fig. 1. Activity instance life cycle

For all the activity state changes, the process engine generates transitional events. These are different from BPMN events used in a process model. Let's say t is the function to depict the timestamp of events, be it BPMN event or transitional event. Then to notify the beginning, termination or cancellation of an activity A, the engine logs the events $t_b(\mathrm{A})$, $t_t(\mathrm{A})$, $t_c(\mathrm{A})$, respectively.

Talking about BPMN events, a process model can consist of start events, intermediate events and end events. These events can be of type catching or throwing [1]. For catching events, the control flow enables the event, waits for it to occur and when it occurs, the process consumes it. But there are certain situations where two events race with each other to determine the process flow.

Fig. 2 shows an online booking reservation process for a workshop. After receiving the booking request, the information is processed and the booking is confirmed. Participants can cancel the booking before they receive the confirmation. Once the place is confirmed, booking is not refundable.

To model that, activity `Process Booking` has an interrupting boundary event attached to it. When cancellation is received while this activity is going on, the booking is cancelled and the participant is notified about the cancellation. Here

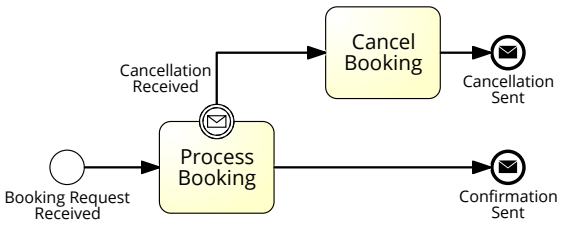

Fig. 2. Interrupting Boundary Event

the boundary event `Cancellation Received` is in a race with the event notifying the termination of the activity `Process Booking`.

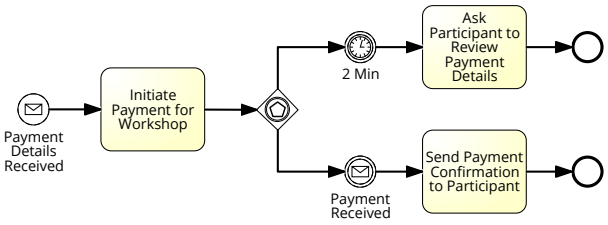

Fig. 3. Event-based Gateway

Another example of racing events is an event-based gateway. Fig. 3 shows the payment process for the workshop. Upon receiving the payment information from the participants, the organizers initiate the payment and wait for confirmation. The confirmation is forwarded to the participant once it is received. If confirmation is not received within 2 min then the participant is asked to check the payment information provided earlier. Here the racing events are the message event `Payment Received` and the timer event after the gateway.

2.2 Event Processing

Though BPMN includes a set of different types of events, it does not say much about the source of the events, how they are generated or how they can be correlated. Event processing, on the other hand, explores such concepts [4]. The single or atomic events do not take time to occur, like a weather update from a sensor. Atomic events can be aggregated to generate complex events, e.g., a flight delay message due to five consecutive bad weather updates. Here, the weather updates are collected, matched with the route of the flight and the message is sent to only those passengers who booked this flight.

Fig. 4 shows the communication between events and processes in a distributed setup. The event source (e.g., a sensor) can directly send the events to a process engine or to an event processing platform. Event processing platforms like Unicorn [5] are able to perform different operations on event streams. It can

receive events from different sources, parse them, aggregate them based on predefined transformation rules to create complex events and notify the subscribers about certain events. These notifications can be sent to a person or a system like a process engine that receives this event and reacts on that according to the process specification. The complex events are mapped to the BPMN events included in the executable process models in the process engine. An event e generally has a timestamp attached to it to describe its time of occurrence, identified as t_o(e). The time when the event is detected by a process engine is called time of detection or t_d(e). The timeline in Fig. 4 shows this. While by definition $t_o(\mathrm{e}) < t_d(\mathrm{e})$, due to the distributed setup, there can be a gap between event occurrence time and event detection time.

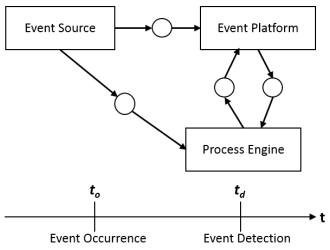

Fig. 4. Event Process Interaction

3 Problem Statement

If there is a significant delay between the occurrence of an event and detection of the event, this can cause a dilemma to choose among the racing events during a process execution. The dilemma can even lead to incorrect process execution. In Fig. 5, we see an example process that describes the general problem. There are four activities A, B, C and D and a boundary event e is attached to the activity A. In this case, the racing events are *e1:* the boundary event e and *e2:* the termination of A, i.e., t_t(A). Now, the problem occurs when $t_o(\mathrm{e1}) < t_o(\mathrm{e2})$ but $t_d(\mathrm{e2}) < t_d(\mathrm{e1})$. For a correct execution, if e1 occurs before e2, then the path lead by e1 should be chosen. But since the detection time of e2 is earlier than the detection time of e1, the process engine thinks that e2 has happened before e1 and follows the branch lead by e2. Thus, the timestamp discrepancy creates the racing events dilemma that results into an incorrect execution of the process.

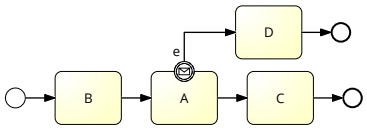

Fig. 5. Problem Statement

If we look back to the example in Fig. 2, it may happen that the moment when the participant pressed the cancellation link was before the `Process Booking` activity finished. Nevertheless, due to server latency the activity terminated before getting notification about the cancellation. In this case, the participants, even if they cancelled the registration timely, do not get the workshop fee back. This problem arises due to the latency between the occurrence time and the

detection time of the boundary event. Here, the problem raised as t_o(Cancellation Received) $< t_t$(Process Booking) $< t_d$(Cancellation Received).

Considering the example in Fig. 3, there can be a situation where the payment is done within 2 min but the confirmation is not received yet. In that case, the timer fires and the participant is asked to review payment details. Since there was not really any problem with the payment information, it creates confusion. Again, the problem occurs due to the timestamp discrepancy t_o(Payment Received) $< t_o$(Timer Event) $< t_d$(Payment Received). In both the cases, with re-evaluation of the situation, the money might be reimbursed or the participant might be informed, respectively. But still it will have negative impacts such as extra time to communicate the problem with the participants, bad user experience and probably some compensation from the organization's side.

One thing to be noted here, the timestamp discrepancy is more probable for those events where the source of the event and the receiver of the event are different. Since timer events are triggered by the engine itself, they are generally in sync with the clock that is followed for process execution. Although, due to engine workload there can be discrepancy between the scheduled time set for a timer and the actual execution of the event [6]. For other type of events, the timestamp discrepancy can occur due to manual or system delay. If the delay occurs for the start event, the process is instantiated late. In case of an intermediate event, the next activity is started late. But there can be severer impacts due to timestamp discrepancy as discussed above.

4 Proposed Solution

Apart from the proactive measures to mitigate the manual delays or increasing system efficiency to communicate faster in a distributed setup, the following measures can be considered to deal with timestamp dilemma. First, we calculate the maximum delay between the occurrence of an event and the detection of the same using a posteriori analysis. Next, we use this maximum delay to propose some solutions to decrease the number of incorrect executions of a process.

Perform a posteriori Analysis. The process engine produces event logs where the transitional events such as *begin* or *termination* of an activity are listed. Assuming that the BPMN events carry the occurrence timestamp information and the detection timestamp is found in the engine log, we can analyse the traces after a process instance is executed. This tells us if the execution was timestamp correct. To do this analysis, we assume that the systems communicating with each other are using logically synchronised clocks such as Lamport clock [7].

Def: Timestamp Correct. An execution is timestamp correct if for racing events occurrence time and detection time are in same sequence.

- If t_o(e1) $< t_o$(e2) holds, then t_d(e1) $< t_d$(e2) also holds.
- If t_o(e1) $> t_o$(e2) holds, then t_d(e1) $> t_d$(e2) also holds.

According to above definition, the traces 1. and 2. below depict correct execution whereas trace 3. is an incorrect execution for Fig. 5.

1. t_b(B) t_t(B) t_b(A) t_t(A) t_b(C) t_t(C)
2. t_b(B) t_t(B) t_b(A) t_o(e) t_d(e) t_c(A) t_b(D) t_t(D)
3. t_b(B) t_t(B) t_b(A) t_o(e) t_t(A) t_d(e) t_b(C) t_t(C)

The a posteriori analysis can give us insight about the probable event for which there is a discrepancy for most of the instances. Thus, we can try to look for the source of the latency. If it involves manual event generation, we might try to automatize that. If it involves sending events from one platform to another, we might try to increase the efficiency of the platforms such that the delay between occurrence time and detection time of an event is ignorable.

Also, we can calculate the maximum delay for each event from the difference between the occurrence time stated in event timestamp and the detection time logged by process engine. This can be used to apply the following solutions.

Sol. I: Delay Execution. If we know the maximum delay between the occurrence and the detection of the event will be Δ, then we can delay the execution of the process flow accordingly to accommodate the latency. For example, if we know that in case of Fig. 2, $\Delta = 10$ sec, then we can set the rule as following: If after t_t(Process Booking) + 10 sec no cancellation is received, then confirmation is sent. Otherwise booking is cancelled. This impacts event subscription since even after the activity terminates, we do not unsubscribe to the boundary event, rather we extend the subscription time by Δ.

Sol. II: Create Updated Model. Even after detecting the source of latency and taking measures to increase the efficiency it may happen that the discrepancy still occurs. Or it might be the case that the manual or system efficiency cannot be increased very easily. Then we can consider updating the process model to accommodate the latency. In Fig. 3, if the a posteriori analysis suggests that the payment confirmation generally takes longer than expected because of the slow service at the payment gateway, we can set the timer at 5 min instead of 2 min. Thereby, we increase the probability of getting confirmation before the timer fires and decrease number of incorrect execution of the process instances.

5 Related Work

Time is a fundamental concept while working with processes or events. Though BPMN process elements follow sequences based on causality, a sequence in time is also introduced while executing the elements in a certain order. E.g, if activity A must happen before activity B, it implies that the termination of A should be before the beginning of B. But BPMN does not differentiate between the occurrence time and detection time for an event. The existing process engines like Camunda [8] or Chimera [9] react on the events only when they receive it. Therefore, they consider only the detection time of the events.

In a distributed setup, the clocks of different participants should be logically synchronized using Lamport clock [7] or similar algorithm. Though it makes sure that the detection time will always be greater than the occurrence time of the event, it does not solve the problem concerning racing events. The Time-out Reordering mechanism described in [10] stores the events in a queue and delays their detection by a time-out which specifies the maximum delay due to processing of the event. Using this time-out, the events are reordered such that occurrence and detection time are in same order. Woo et. Al [11] have also dealt with the timestamp discrepancy in a similar way. In their frameowrk PTMON, they assume the upper bound of the delay is known and generate RTL formulas to detect probabilistic timing constraints violation. However, none of these methods consider the causal relationship of events with other elements of the process, such as termination of an activity.

In previous discussion, we highlighted the situations like listening to an event while an activity is going on or delaying execution even after an activity with boundary event is terminated. This raises the concerns about event subscription. The CASU Framework proposed by Decker and Mendling [12] discusses about subscriptions for events which are needed for process instantiation. This can be a basis to explore further about the right point in time to make a subscription, the duration of the subscription and so on. The authors in [13] proposes a causal ordering between event subscription, event occurrence and event consumption. Our work adds another dimension to it, i.e., event detection.

The work by Niculae [14] and Eichenberg [15] discuss about different time patterns in workflow management systems where the minimum or maximum time between termination of one activity and beginning of the next activity can be specified. These time patterns can guide us to optimize between the increased waiting time for detecting an event but still starting the next activity timely so that the overall process duration remains acceptable.

6 Conclusion and Future Work

The BPMN racing events determine the path a process execution should follow. Due to manual delay or processing time needed for communication in a distributed system, there might be discrepancy between the occurrence time of the events and the detection time when the events are received in the process engine. This timestamp discrepancy can cause dilemma between choosing the alternative racing events and even lead to wrong execution of the process. In this work, an investigation into timestamp discrepancy leading to racing events dilemma has been done that shows the possible occurrences of the problem and the impact caused by it. Also, a few proactive and reactive measures are presented to mitigate the discrepancy that may eventually cause the dilemma.

However, further investigation is needed to check the frequency and severance of problems caused by timestamp discrepancy in reality and effectiveness of proposed solutions. Another future direction can be to explore if there exist an alternative to model the processes without racing events so that the problem

can be avoided. The future research should find an efficient way to execute the a posteriori analysis combining the event timestamp with engine generated log. Also, instead of doing a trace analysis after the process has been executed, a run-time trace analysis would be interesting that can validate timestamp correctness on the fly and propose compensation as soon as incorrect execution is detected.

References

1. OMG: Business Process Model and Notation (BPMN), Version 2.0. http://www.omg.org/spec/BPMN/2.0/ (January 2011)
2. Etzion, O., Niblett, P.: Event Processing in Action. Manning Publications (2010)
3. Weske, M.: Business Process Management - Concepts, Languages, Architectures, 2nd Edition. Springer (2012)
4. Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley (2010)
5. UNICORN: Complex Event Processing Platform. https://bpt.hpi.uni-potsdam.de/UNICORN/WebHome
6. Ferme, V., Ivanchikj, A., Pautasso, C.: A framework for benchmarking bpmn 2.0 workflow management systems. In: 13th International Conference on Business Process Management (BPM 2015), Springer, Springer (August 2015)
7. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM **21**(7) (1978)
8. Camunda: camunda BPM Platform. https://www.camunda.org/
9. Chimera: Case Engine. https://bpt.hpi.uni-potsdam.de/Chimera
10. Hinze, A., Buchmann, A.P.: Principles and applications of distributed event-based systems. Hershey, PA : Information Science Reference (2010)
11. Woo, H., K. Mok, A., Chen, D.: Realizing the potential of monitoring uncertain event streams in real-time embedded applications. IEEE Real-Time and Embedded Technology and Applications (2007)
12. Decker, G., Mendling, J.: Process instantiation. Data Knowledge Engineering **68**(9) (2009) 777–792
13. Barros, A., Decker, G., Grosskopf, A.: Complex events in business processes. In: Business Information Systems, Springer (2007)
14. Niculae, C.C.: Time patterns in workflow management systems. Master thesis, Eindhoven University of Technology (2011)
15. Eichenberg, M.: Event-Based Monitoring of Time Constraint Violations. Master thesis, Hasso Plattner Institute (2016)