

DENNIS WOLF

INVESTIGATING EVENT SUBSCRIPTION  
MECHANISMS IN BPMN



# INVESTIGATING EVENT SUBSCRIPTION MECHANISMS IN BPMN

DENNIS WOLF



Digital Engineering • Universität Potsdam

< Any Subtitle? >

August 2017 – version 1

Dennis Wolf: *Investigating Event Subscription  
Mechanisms in BPMN*, < Any Subtitle? >, © August 2017

## ABSTRACT

---

Business Processes have become an essential tool in organizing, documenting and executing company workflows while Event Processing can be used as a powerful tool to increase their flexibility especially in distributed scenarios. The publish-subscribe paradigm is commonly used when communicating with complex event processing platforms, nevertheless prominent process modelling notations do not specify how to handle event subscription.

At the example of BPMN 2.0, the first part of this work illustrates the need for a flexible usage of event subscription in process models and derives new requirements for process modelling notations. An assessment of the coverage of these requirements in BPMN 2.0 is presented and shortcomings are pointed out.

Based on the identified requirements, this work presents a new concept for handling event subscription in business process management solutions, predominantly built on the notion of event buffers. The concept includes an extension to the BPMN meta model, specifies the semantics and API of a new event buffering module and describes the changes necessary to the behaviour of the process engine.

For evaluation purposes, the concept has been implemented as a reusable Camunda Process Engine Plugin that interacts with the academic Complex Event Processing Platform UNICORN.

## ZUSAMMENFASSUNG

---

Kurze Zusammenfassung des Inhaltes in deutscher Sprache...



## CONTENTS

---

1	INTRODUCTION	1
2	BACKGROUND	3
3	PROBLEM STATEMENT	5
3.1	Motivating Examples	5
3.2	Event Occurrence Scenarios	5
3.3	Requirements Definition	7
4	ASSESSMENT OF CURRENT BUSINESS PROCESS MANAGEMENT SOLUTIONS	9
4.1	BPMN Models in presence of the Event Occurrence Scenarios	9
4.2	Implementation of Early Event Subscription using standard Camunda	13
4.3	Discussion	13
5	FLEXIBLE EVENT SUBSCRIPTION	15
5.1	BPMN Extension	15
5.2	Buffered Event Handling	15
5.3	Extended Process Engine Behaviour	16
	BIBLIOGRAPHY	17

## LIST OF FIGURES

---

Figure 1	Possible event occurrence times in relation to a process execution life cycle	6
Figure 2	Abstract Process using an Intermediate Catch Event	9
Figure 3	Standard Intermediate Catch Event	10
Figure 4	Intermediate Event with a parallel Timer Event	10
Figure 5	Event Element in parallel process flow	11
Figure 6	Event Buffering through an auxiliary Buffering Process	12

## LIST OF TABLES

---

## LISTINGS

---

## ACRONYMS

---



## INTRODUCTION

---







## PROBLEM STATEMENT

---

This section will further define the problem and derive formal requirements to event subscription mechanisms

### 3.1 MOTIVATING EXAMPLES

- one example for independent. The subscription does not depend on a prior process result, the subscription can be done even before process instantiation
- one example for a process that uses an intermediate event that depends (subscription-wise) on the result of a previous step in the process.
  - ==> If the event occurs at a certain time, the process gets delayed unnecessarily or even run into a deadlock

### 3.2 EVENT OCCURRENCE SCENARIOS

Given the motivating examples, I am deriving a generic set of event occurrence scenarios. Each of these scenarios can occur in the real world and process implementations need to be capable of handling them to avoid negative effects.

**TIME OF EVENT OCCURRENCE** The most important variable to consider is the time of event occurrence. According to the BPMN specification, it is possible to catch an event if it occurs after the event element is enabled. As shown before, it is often impossible to control occurrence time and events do occur outside of these time windows. We specify the possible event occurrence times in relation to the life cycle of a process that utilizes a BPMN Intermediate Event

ref process lifecycle

Figure 1 shows the life cycle steps of a process and an instance from the deployment of the process until the undeployment and uses a timeline to illustrate that an event might occur at any time during this cycle. More precisely, an event is always considered to occur before or after a life cycle step or in between two consecutive steps.

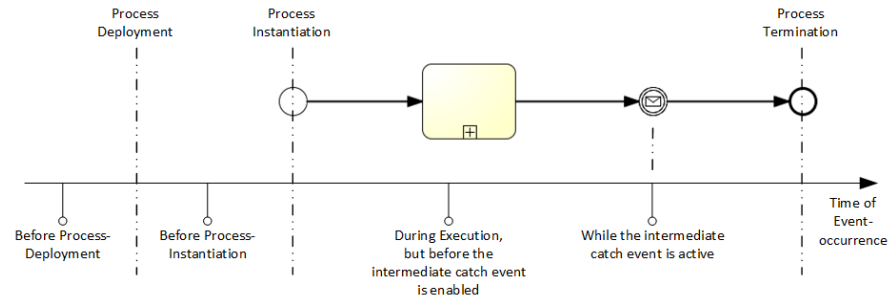


Figure 1: Possible event occurrence times in relation to a process execution life cycle

How about system-deployment/process engine start and process undeployment? show in illustration, but say in text that we simplify this for now. after undeployment is essentially before deployment of a new process; Before Engine start is also before pr. deployment and we presume that an engine is running and does not stop.

Given the relevant life cycle steps, *process deployment*, *process instantiation* and *Event enablement*, the following occurrence scenarios are distinguished in this work:

- A. O1 After the enabling of the BPMN event (BPMN default)
- B. O2 The event does not occur
- C. O3 Between Process instantiation and the enabling of the BPMN event
- D. O4 Between Process deployment and process instantiation
- E. O5 Before Process deployment

add a back reference to the examples? In example XY, events can occur before... whereas in example...

For a flexible and efficient use of events in business processes, it must be possible to use events that occur in any of these phases. To make sure that an event can be caught, no matter at which time during the phase it occurs, the subscription to the CEP platform must happen at the beginning of the occurrence phase. It follows that the event subscription must be possible at system start, at process deployment, at process instantiation, at any time during process execution and when the BPMN Event element is enabled.

**EVENT SUBSCRIPTION DEPENDENCIES** It is important to note that the subscription to an event source can depend on additional context information or process data. This can be a severe limitation to the possible subscription time.

ref to process model

shows a logistics process that uses event data about the GPS position of a certain truck to keep the estimated time of arrival of the transport updated. Whenever it receives an updated GPS position, the ETA is re-calculated; once the *arrival*-event has been received, the process finishes.

this example is not good, because we are not interested in a gps event that occurs earlier. Find an example where you would like earlier events, but subscription is not possible

Before the subscription to that specific truck gps event can happen, the process must determine the *truckId* to use in the event query. Only when the *truckId* is available, the subscription can be executed. This example illustrates how a query filter expression can depend on context data, but it might as well be the event source itself that differs depending on the particular execution.

there could be an xor gateway and following two different events and only one of them can get executed

solution would be to listen to all gps, but potentially too much data. Decision must be made cautiously! <= Where should I mention this? maybe later in the concept

### 3.3 REQUIREMENTS DEFINITION

The previous sections have exemplified how the execution semantic offered by the BPMN specification limits users in the use of events in business processes. Now these shortcomings are formalized into an additional set of requirements that must be met by a process execution environment to enable event handling in the extended set of event occurrence scenarios. The formal requirements will later be used to evaluate the capabilities of current Process Management Solutions and to develop a new concept to handling event subscription in business processes.

ref to chapters

#### R1: FLEXIBLE EVENT SUBSCRIPTION TIME

*R1.1: Explicitness:* For each event that is used in a business process, it must be possible to derive the time of event subscription from the process model. The time of subscription may either be explicitly stated or defined implicitly.

*R1.2: Flexibility:* The time of subscription can be influenced to catch events according to any of the event occurrence scenarios O1, O2, O3, O4. In other words, the process model defines the earliest acceptable time for an event occurrence to be considered in the process

execution. The necessary options are *since system start*, *since process deployment*, *since process instantiation*, from an arbitrary but *explicit time during process execution*, or *since enabling of the Event Process Element*.

limited by subscription dependencies

#### R2: AUTOMATIC SUBSCRIPTION HANDLING

*R2.1: Subscription* The subscription to event sources is handled implicitly by the process execution environment as defined by the process model.

*R2.2: Removal of Subscription* The removal of a subscription from the system is handled automatically as soon as a subscription becomes unnecessary.

#### R3: EVENT BUFFERING

To make all events since the subscription time available during process execution, matching events need to be stored temporarily.

buffer policies and scope?



## ASSESSMENT OF CURRENT BUSINESS PROCESS MANAGEMENT SOLUTIONS

The lack of flexibility in handling event subscription in business processes has been outlined in the previous chapters and a set of extended requirements to process management solutions have been presented. In this section I take a closer look at the capabilities of current solutions with regards to the event occurrence scenarios to get a better understanding of the issues that arise when working with event subscription in business processes. The assessment will be carried out using BPMN and Camunda, a state-of-the art and widely adopted business process engine. The main goal is to identify and illustrate the shortcomings of the current process technology stack. These shortcomings will be referenced in addition to the presented requirements to develop a more refined subscription handling model in the following chapter.

"subscription handling model"?

### 4.1 BPMN MODELS IN PRESENCE OF THE EVENT OCCURRENCE SCENARIOS

Chapter X has revealed that processes can run into deadlocks if events do not occur at the right time

Figure 2 shows a generalized process that uses an Intermediate Catch Event just before process termination. In this section I first describe for each Event Occurrence Scenario how this simple event implementation behaves in presence of the given scenario. I then evaluate if it is feasible to create a BPMN model that is free from deadlock in these situations.

**SCENARIO 01: THE EVENT OCCURS AFTER THE ENABLING OF THE BPMN EVENT** The first scenario represents the most simple case,

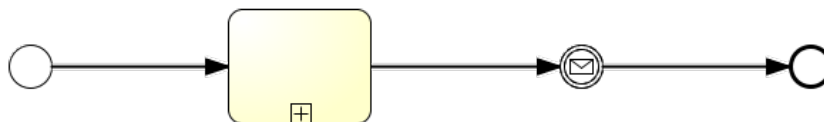


Figure 2: Abstract Process using an Intermediate Catch Event

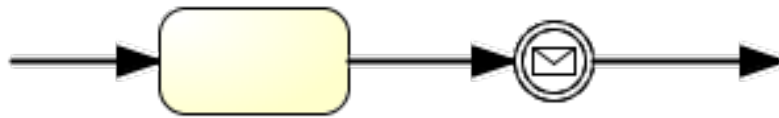


Figure 3: Standard Intermediate Catch Event

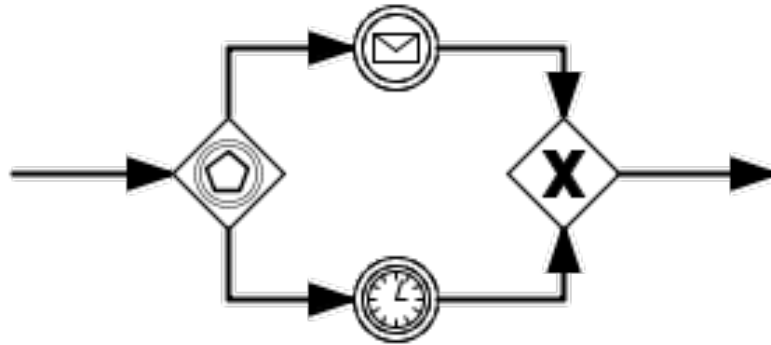


Figure 4: Intermediate Event with a parallel Timer Event

that is also natively supported by the BPMN 2.0 specification. When the event occurs after the Event element has been enabled, the event will be received and the process can proceed normally. The use of a standard Intermediate Catch Event does suffice to cover this situation.

**SCENARIO 02: THE EVENT DOES NOT OCCUR** In certain situations an event might not occur at all. Given a basic event implementation like in [Figure 2](#), the process flow will get to a halt once it reaches the Intermediate Catch Event and will not be able to proceed. While, depending on the process design, this might be the desired behavior, in many situations this is not acceptable.

Let's consider a process that is supposed to wait for approval for a certain amount of time and trigger an additional request if the approval has not been issued before the deadline. [Figure 4](#) shows how this behavior can be implemented using an Event-based Gateway which puts a Timer Event in parallel to the Intermediate Catch Event. This extension will make sure that a process does not run into a deadlock state if the expected event does not occur.

I mention an example, but that example is not exactly illustrated in the process

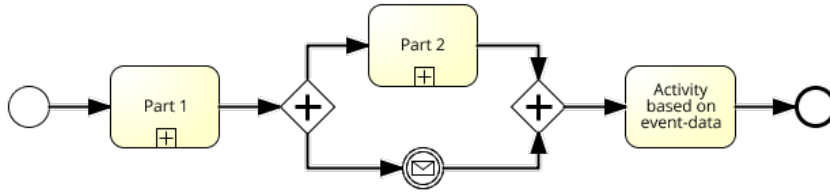


Figure 5: Event Element in parallel process flow

according to the spec: what exactly will happen to the active catch event once the timer fires?

**SCENARIO O3: OCCURRENCE BETWEEN PROCESS INSTANTIATION AND THE ENABLING OF THE BPMN EVENT** In case the event occurs during process execution, but before the BPMN event element is enabled and thus listening for events, the occurrence will not be considered in the execution. The process will get stuck at the Event Process Element as if the event did not happen at all. To avoid a deadlock in this scenario, a solution is to execute the Intermediate Catch Event in parallel to the rest of the process flow using a Parallel Gateway. This is illustrated in [Figure 5](#). The time of subscription to the event can be controlled by the position of the parallel split: To implement an event subscription right after process instantiation, the Parallel Gateway has to be the first element after the Start Event (that means *Part 1* in the illustration is empty). To implement event subscription at a specific point during process execution, part of the process can execute before reaching the Parallel Gateway. In [Figure 5](#), the event may occur at any time during the execution of the collapsed sub-process *Part 2*.

**SCENARIOS O4 AND O5: BEFORE PROCESS INSTANTIATION** Any Events that happen before process instantiation will not be considered in a standard Intermediate Catch Event. That applies to both scenarios, the occurrence between deployment and instantiation (*O4*) and an occurrence time before the deployment of the process in the Process Engine (*O5*).

To create a Process Model that allows to catch an event before the process instance exists, three new elements are introduced: (1) An additional *Auxiliary Buffering Process* that can catch an incoming event, (2) an *Event Buffer*, a temporary data-store that keeps event data until required by the *Original Process*, (3) an *Auxiliary Event Delivery Process*, that retrieves events from the buffer and makes them available to the *Original Process*. [Figure 6](#) reveals the interaction of the *Original Process*, the two auxiliary processes and the data-store. To start listening for an event, the *Auxiliary Buffering Process* has to be instantiated through

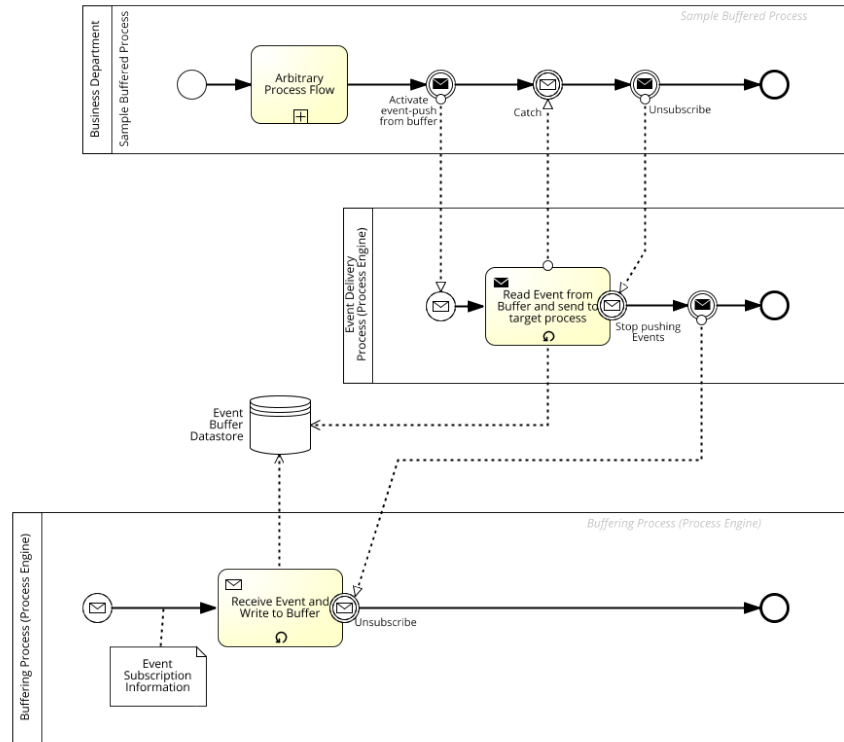


Figure 6: Event Buffering through an auxiliary Buffering Process

the a message start event containing the information necessary for the event subscription. The process starts listening for the event and writes the received event to the temporary data-store. The given process design is able to handle multiple event occurrences, because the receiving activity is looping. The buffering process terminates once the *Unsubscribe* event is received.

how should the ABP be started? manually?

The *Original Process* can be started any time after the buffering process. In Figure 6, the Intermediate Catch Event has been explicitly split into three events: An initial Send Event to request events, a Catch Event to receive and a final Send Event to signal that no events shall be received anymore. The initial Send Event instantiates the *Auxiliary Event Delivery Process*, which tries to read from the Event Buffer and deliver the event to the *Original Process*. Once there is data available in the buffer, it is sent by the sending activity. The central looping activity will retry reading from the buffer until data becomes available and will only be terminated once the *Stop*-event occurs. The *Original Process* can receive the event using a standard Intermediate Catch Event even when the event occurs before the instantiation of *Original Process*, so it handles scenario *O4*. Moreover the *Auxiliary Buffering Process* is not bound to a specific event, it works generically with any event information that is passed on to it. For that reason it is also

not bound to a specific process deployment and can buffer events even before a process has been deployed, so it handles scenario O5. Given that the buffering process can alternatively be started using an explicit Message Send Event during process execution and the process does not stop listening until the Original Process has received the event, scenarios O1 and O3 are also supported.

what exactly is passed around | how many instances of each process | overwrite or append to buffer

#### 4.2 IMPLEMENTION OF EARLY EVENT SUBSCRIPTION USING STANDARD CAMUNDA

Is it possible to implement this using out-of-the-box Camunda? Which aspects cannot be (sufficiently) implemented? How can ... be implemented in Camunda? Show details. Diagrams in appendix. (3 pages)

#### 4.3 DISCUSSION

What are the shortcomings when using out-of-the-box business process solutions to implement Early Event Subscription? What can be implemented without problems? (2 pages)



## FLEXIBLE EVENT SUBSCRIPTION

---

Present an abstract framework for flexible event subscription. > Including: Model <> Process Engine <> Buffer <> CEP > How does event subscription currently affect the workflow? > What should a workflow look like that allows early event subscription? > What must be explicitly stated by the user? What should be done automatically in the background? (1 page)

### 5.1 BPMN EXTENSION

To fulfill requirements R1.1 and R1.2, additional information has to be included in the BPMN model. By default, a BPMN intermediate event does not have information on the time of subscription or the event query. The BPMN specification offers BPMN-X extensions to add custom properties or elements to a model.

To accomodate the required information, the following extension is proposed: > The extension should apply to `MessageIntermediateCatchEvent` and `MessageBoundaryEvent` > extend `tMessage` => `tBuffered-CEPMessage`, so that the `messageRef` can be reused > OR extension to `messageEventDefinition`: `ExplicitSubscriptionMessageEventDefinition` > [`subscriptionQuery`, `subscriptionTime`, `bufferPolicy`]

A buffer shared across multiple instances or events is more complex than a simple single-event-buffer (that one does not require buffer policies). As soon as the `requestEvent` call can be executed multiple times for the same `queryId`, we need to specify the following aspects:

Buffer policies: (widely based on [Ref paper Sankalita]) `RetrievalPolicy`, `ConsumptionPolicy`, `LifetimePolicy` + buffer maximum age (= combination of lifetime policies)

### 5.2 BUFFERED EVENT HANDLING

Why do we need a buffer to allow early event subscription?

What is the desired functionality of the event buffer? What functionality (API) does it expose? > this could be seen as an extension to the API that is exposed by a standard CEP Platform > standard platform API: `registerQuery(queryString, notificationRecipient) : queryId`, `deleteQuery(queryId)` > extended API: `registerQuery(queryString): queryId`, `requestEvent(queryId, notificationRecipient)`, `unsubscribe(queryId)`, `deleteQuery(queryId)`

### 5.3 EXTENDED PROCESS ENGINE BEHAVIOUR

=> As a link between the BPMN model and the Buffered Event handling

there must be a "subscription-garbage-collection" for any events that cannot be reached anymore in the current process execution!  
e.g. two different events behind an xor-gateway. the garbage collection could be executed on every transition



## DECLARATION

---

Put your declaration here.

*Potsdam, August 2017*

---

Dennis Wolf



## COLOPHON

This document was typeset using the typographical look-and-feel classicthesis developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". classicthesis is available for both  $\text{\LaTeX}$  and  $\text{\LyX}$ :

<https://bitbucket.org/amiede/classicthesis/>

Happy users of classicthesis usually send a real postcard to the author, a collection of postcards received so far is featured here:

<http://postcards.miede.de/>