MODELING AND EXECUTING BATCH ACTIVITIES IN BUSINESS PROCESSES

LUISE PUFAHL

BUSINESS PROCESS TECHNOLOGY GROUP
HASSO PLATTNER INSTITUTE
UNIVERSITY OF POTSDAM
POTSDAM, GERMANY

DISSERTATION
ZUR ERLANGUNG DES AKADEMISCHEN GRADES EINES
"DOCTOR RERUM NATURALIUM"
– DR. RER. NAT. –

March, 2018

Luise Pufahl: Modeling and Executing Batch Activities in Business Processes,
eingereicht an der Digital-Engineering-Fakultät
des Hasso-Plattner-Instituts und der Universität Potsdam. März 2017

ABSTRACT

Business process automation improves organizations' efficiency to perform work. Therefore, a business process is first documented as a process model which then serves as blueprint for a number of process instances representing the execution of specific business cases. In existing business process management systems, process instances run independently from each other. However, in practice, instances are also collected in groups at certain process activities for a combined execution to improve the process performance. Currently, this so-called *batch processing* is executed manually or supported by external software. Only few research proposals exist to explicitly represent and execute batch processing needs in business process models. These works also lack a comprehensive understanding of requirements.

This thesis addresses the described issues by providing a basic concept, called *batch activity*. It allows an explicit representation of batch processing configurations in process models and provides a corresponding execution semantics, thereby easing automation. The batch activity groups different process instances based on their data context and can synchronize their execution over one or as well multiple process activities. The concept is conceived based on a requirements analysis considering existing literature on batch processing from different domains and industry examples. Further, this thesis provides two extensions: First, a flexible batch configuration concept, based on event processing techniques, is introduced to allow run time adaptations of batch configurations. Second, a concept for collecting and batching activity instances of multiple different process models is given. Thereby, the batch configuration is centrally defined, independently of the process models, which is especially beneficial for organizations with large process model collections. This thesis provides a technical evaluation as well as a validation of the presented concepts. A prototypical implementation in an existing open-source BPMS shows that with a few extensions, batch processing is enabled. Further, it demonstrates that the consolidated view of several work items in one user form can improve work efficiency. The validation, in which the batch activity concept is applied to different use cases in a simulated environment, implies cost-savings for business processes when a suitable batch configuration is used. For the validation, an extensible business process simulator was developed. It enables process designers to study the influence of a batch activity in a process with regards to its performance.

ZUSAMMENFASSUNG

Die Automatisierung von Geschäftsprozessen verbessert die Effizienz von Organisationen im Bearbeiten ihrer Aufgaben. Dafür wird ein Geschäftsprozess zunächst als Prozessmodell dokumentiert, der dann als Vorlage für eine Menge von Prozessinstanzen, welche die Ausführung von Geschäftsfällen repräsentieren, dient. In existierenden Prozessmanagement-Systemen werden Prozessinstanzen komplett unabhängig voneinander ausgeführt. In der Praxis jedoch werden Instanzen häufig zur Verbesserung der Prozessperformance an bestimmten Prozessaktivitäten in Gruppen gesammelt, um diese gebündelt auszuführen. Das sogenannte *Batch Processing* wird zurzeit nur manuell oder durch externe Software unterstützt. Wenige Forschungsarbeiten existieren, um Batch Processing-Konfigurationen in Prozessmodellen explizit zu repräsentieren und sie automatisiert auszuführen. Zusätzlich fehlt es diesen Arbeiten an einem umfassenden Verständnis der Anforderungen.

Die vorliegende Dissertation befasst sich mit den oben genannten Fragestellungen, indem ein *Batch Activity*-Konzept entwickelt wird. Dieses erlaubt es *Batch Processing*-Aktivitäten in Geschäftsprozessen zu spezifizieren als auch zu konfigurieren und mittels einer zusätzlich bereitgestellten Ausführungssemantik zu automatisieren. Die *Batch Activity* kann verschiedene Prozessinstanzen auf Basis ihres Datenkontextes gruppieren und deren Ausführung über ein oder mehrere Aktivitäten synchronisieren. Das Konzept basiert auf einer Anforderungsanalyse, welche existierende Forschungsarbeiten zum Thema des Batch Processings aus unterschiedlichen Domänen als auch Praxisbeispiele berücksichtigt. Weiterhin werden zwei Erweiterungen des Basiskonzeptes in dieser Arbeit vorgestellt: Erstens wird ein Konzept zur flexiblen Anpassung der Batch-Konfiguration zur Ausführungszeit basierend auf Techniken der Ereignisverarbeitung vorgestellt. Zweitens wird ein Konzept eingeführt, um Aktivitätsinstanzen von verschiedenen Prozessmodellen zu sammeln und zu konsolidieren. Dabei wird die Batch-Konfigu ration unabhängig von Prozessmodellen zentral definiert, was besonders für Unternehmen mit großen Prozesssammlungen hilfreich ist. Die vorliegende Dissertation beinhaltet eine technische Evaluation als auch eine Validierung der eingeführten Konzepte. Eine prototypische Implementierung in ein bestehendes, open-source Prozessmanagement-System zeigt, dass Batch Processing mit wenigen Erweiterungen integriert werden kann. Zusätzlich wird demonstriert, dass die konsolidierte Darstellung von mehreren Prozessfällen in einer Benutzeransicht die Arbeitsleistung von Endanwendern verbessern kann. Die Validierung, in der das *Batch Activity*-Konzept in unterschiedlichen Anwendungsfällen in einer simulierten Umgebung eingesetzt wird, impliziert Prozesskosteneinsparungen, wenn eine geeignete Batch-Konfiguration gewählt wird. Für die Validierung wurde ein erweiterbarer Geschäftsprozess-simulator entwickelt. Dieser ermöglicht es Prozessmodellierern, den Einfluss einer *Batch Activity* auf einen Prozess mit Hinblick auf dessen Performance zu untersuchen.

PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

- Luise Pufahl, and Mathias Weske. Batch Activities in Process Modeling and Execution. In *Service-Oriented Computing (ICSOC)*, pages 283–297. Springer, 2013.
- Luise Pufahl, Andreas Meyer, and Mathias Weske. Batch Regions: Process Instance Synchronization based on Data. In *Enterprise Distributed Object Computing (EDOC)*, pages 150–159. IEEE, 2014. [Best Student Paper]
- Luise Pufahl, Nico Herzberg, Andreas Meyer, and Mathias Weske. Flexible Batch Configuration in Business Processes based on Events. In *Service-Oriented Computing (ICSOC)*, pages 63–78. Springer, 2014.
- Luise Pufahl, and Mathias Weske. Batch Processing Across Multiple Business Processes Based on Object Life Cycles. In *Business Information Systems (BIS)*, pages 195–208. Springer, 2016. [Best Paper]
- Luise Pufahl and Mathias Weske. Enabling Batch Processing in BPMN Processes.. In *BPM Demos*, pages 28–33. Springer, 2016.
- Luise Pufahl, and Mathias Weske. Requirements Framework for Batch Processing in Business Processes. In *Business Process Modeling, Development and Support (BPMDS)*, pages 85–100, Springer, 2017.
- Luise Pufahl, Tsun Yin Wong, and Mathias Weske. Design of an Extensible BPMN Process Simulator. In *BPM Workshops*, accepted, 2017.

In addition to above publications as part of this thesis, I was also involved in the following research indirectly contributing to this thesis:

- Andreas Meyer, Luise Pufahl, Kimon Batoulis, Dirk Fahland, and Mathias Weske. Automating data exchange in process choreographies. *Information Systems*, 53:296–329, 2015.
- Andreas Meyer, Luise Pufahl, Dirk Fahland, and Mathias Weske. Modeling and Enacting Complex Data Dependencies in Business Processes. In *Business Process Management (BPM)*, pages 171–186. Springer, 2013.
- Luise Pufahl, Ekaterina Bazhenova, and Mathias Weske. Evaluating the Performance of a Batch Activity in Process Models. In *BPM Workshops*, pages 277–290. Springer, 2014.
- Andreas Meyer, Luise Pufahl, Kimon Batoulis, Sebastian Kruse, Thorben Lindhauer, Thomas Stoff, Dirk Fahland, and Mathias

Weske. Automating Data Exchange in Process Choreographies. In *Advanced Information Systems Engineering (CAiSE)*, pages 316–331. Springer, 2014.
- Luise Pufahl, and Mathias Weske. Parent-child Relation Between Process Instances. In *BPM Forum*, pages 20-37. Springer, 2015.
- Luise Pufahl, Sankalita Mandal, Kimon Batoulis and Mathias Weske. Re-evaluation of Decision based on Events. In *Business Process Modeling, Development and Support (BPMDS)*, pages 68–84, Springer, 2017.

ACKNOWLEDGMENTS

"Research only works in a *research ecosystem* to reshape your ideas." One of the valuable sayings I got during working on my PhD from my supervisor Mathias Weske. Looking back, I had a great and very pleasant research ecosystem influencing me in a very valuable way in writing my thesis. This research ecosystem was first and foremost created by my supervisor Mathias Weske to whom goes my special thanks. He introduced me as a former 'Information Systems'-student to the world of computer science research with continuous encouragement, patience, fruitful discussions, and valuable hints to find my way. Additionally, I want to especially thank Jan Mendling who triggered my enthusiasm for Business Process Management during my Master studies with his lectures at the Humboldt University Berlin where he also presented new research topics, such as C-EPCs which I still remember. I also want to thank my second reviewer, Stefanie Rinderle-Ma, for valuable feedback, and interesting talks that we had at different conferences. She is a role model for me as a successful female researcher.

Many thanks to my colleagues within the BPT group who formed a special, warmhearted, and enjoyable research ecosystem. Especially, I want to mention here Andreas Meyer and Dirk Fahland with whom I shared a very intensive and successful research work. Thank you both for your inspiration and your valuable feedback questioning always every detail. Furthermore, I also enjoyed the research collaboration with my colleagues Andreas Solti, Nico Herzberg, Ekatherina Bazhenova, Marcin Hewelt, Sankalita Mandal, and Kimon Batoulis. Also thanks to my teaching partners Matthias Kunze and Kimon Batoulis for the great teaching work together which I always enjoyed and which also helped me a lot to understand things in more depth. Thank you Adriatik, Andreas, Dimka, Francesca, Kimon, Katya, Marcin, and Sankalita for proof-reading my thesis.

All my thanks goes to my husband Marian, my love, for supporting, listing, and pushing me to believe in my work in all those years. Thanks for your patience, your strength, and thanks for our daughter Melina who is our joy and pride. I also want to thank my family and my friends for their support. Especially I want to thank my mum who was always there to take Melina to give me some space to work; my father, my brother Jonas, and my best friends Anne and Jule for always being there and listening.

Luise Pufahl
11th October, 2017

CONTENTS

I INTRODUCTION & BACKGROUND 1

1 INTRODUCTION 3
- 1.1 Motivation 3
- 1.2 Problem Statement 5
- 1.3 Research Objectives and Research Framework 6
- 1.4 Summary of Contributions 8
- 1.5 Structure of the Thesis 10

2 PRELIMINARIES 13
- 2.1 Process Models 14
- 2.2 Business Process Model and Notation 15
- 2.3 Data Aspects in Process Models 18
- 2.4 Process Execution 20
- 2.5 Events and Business Processes 24

3 RELATED WORK 27
- 3.1 Original Domains of Batch Processing 27
- 3.2 Batch Processing in Business Process Management 32
- 3.3 Conclusion 36

II BASIC CONCEPTS 39

4 REQUIREMENTS ANALYSIS 41
- 4.1 Scenarios Requiring Batch Processing 42
- 4.2 Requirements Analysis 47
- 4.3 Objectives and Prioritization of Requirements 54

5 BATCH ACTIVITY 57
- 5.1 Process Instance Grouping with Data Views 58
- 5.2 Modeling 62
- 5.3 Execution Semantics 71
- 5.4 User Involvement 77
- 5.5 Conclusion 79

III EXTENDED CONCEPTS 83

6 FLEXIBLE BATCH CONFIGURATION WITH EVENTS 85
- 6.1 Motivating Example and Requirements 86
- 6.2 Flexible Configuration Concept 89
- 6.3 Validation of Flexible Batch Activity Configurations 96
- 6.4 Conclusion 99

7 BATCH PROCESSING ACROSS MULTIPLE BUSINESS PROCESSES 101
- 7.1 Motivating Example 102
- 7.2 Requirements and Design Objectives 103
- 7.3 Batch Specification in Object Life Cycles 106
- 7.4 Extension to Connected Batch Transitions 110

7.5 Conclusion 114

IV EVALUATION AND CONCLUSIONS 115

8 PROOF-OF-CONCEPT IMPLEMENTATION 117
8.1 Implementation of the Batch Activity 117
8.2 Implementation of Multi-Process Batching 121
8.3 Conclusion 124

9 APPLICATION TO USE CASES 125
9.1 Extensible BPMN Process Simulator 127
9.2 Performance Measures for Batch Activities 129
9.3 Application to an Administrative Process 131
9.4 Application to an Online Retailer Process 136
9.5 Conclusion 141

10 CONCLUSIONS 143
10.1 Summary of Thesis Results 143
10.2 Limitation and Discussion 146
10.3 Future Research 149

V APPENDIX 151

BIBLIOGRAPHY 153

LIST OF FIGURES

Figure 1 Business process life cycle 4
Figure 2 Design science process and its application 7
Figure 3 Contributions of the thesis 9

Figure 4 Elements of BPMN business process diagrams 16
Figure 5 Online retailer BPMN process diagram 18
Figure 6 Data class 'Order' with its object life cycle 19
Figure 7 BPMS architecture 21
Figure 8 Online retailer BPMN process diagram with exemplary running instances 22
Figure 9 Activity life cycle 23

Figure 10 Different type of production processes 28

Figure 11 Shipment process to allow batching of shipments 45
Figure 12 Leave application process to allow batching of vacation request 46
Figure 13 Classification of requirements 47
Figure 14 Requirements framework for integrating batch processing in business processes 48
Figure 15 Comparison of the eight scenarios based on the requirements framework 53
Figure 16 Prioritization of requirements for setting the design objectives 55

Figure 17 Online retailer BPMN process diagram with exemplary running instances 58
Figure 18 Data view example 59
Figure 19 Conceptual model for batch activities 63
Figure 20 Exemplary configurations of batch activities 64
Figure 21 Batch activity configuration integrated into BPMN specification 69
Figure 22 Integration of the Batch Activity in EPCs 70
Figure 23 Execution semantics of the batch activity exemplified 71
Figure 24 Life cycle of a batch cluster 72
Figure 25 Parallel execution 75
Figure 26 Sequential-per-activity execution 76
Figure 27 Sequential-per-case execution 77
Figure 28 Possibility for user involvement in different batch cluster states 78

Figure 29 Comparison of existing batch processing solutions 80

Figure 30 Blood testing process 86
Figure 31 Events influencing a batch cluster 90
Figure 32 Conceptual model for integrating batch activities and event processing concepts 91
Figure 33 Extended batch cluster life cycle 94
Figure 34 Reassignemnt of process instances 95
Figure 35 System architecture to realize batch adjustments 96
Figure 36 Number of expired blood samples for different simulation runs 98
Figure 37 Average waiting time for the technician in the different settings 99

Figure 38 Business process models of an electricity supplier 102
Figure 39 Exemplary task list 103
Figure 40 OLC of the data class 'Customer Letter' 105
Figure 41 OLC of the data class 'Customer Letter' with a batch transition 106
Figure 42 Main steps to batch instances of multiple processes. 107
Figure 43 Illustration of algorithm to batch instances of multiple processes 108
Figure 44 Resulting batch work item 109
Figure 45 Extended OLC considering automatic enveloping 110
Figure 46 Extended OLC considering signing of letters 111
Figure 47 OLC of 'Customer Letter' with sequential connected batch transitions 112
Figure 48 OLC of 'Customer Letter' with similar batch transitions being connected 113

Figure 49 bpmn.io extension to configure batch activities 118
Figure 50 Architecture of the extended Camunda engine 119
Figure 51 Sequence diagram visualizing the interaction among added classes 120
Figure 52 Batch work item in the Camunda Worklist App 121
Figure 53 Architecture of the extended Camunda engine for multi-process batching 122
Figure 54 Choose-Task-form and batch work item 123

Figure 55 Plug-in structure of the BPMN process simulator 128

Figure 56 Application of batch activity concept to an office room relabeling process 132
Figure 57 Simulation results regarding process cost and batch size for the administrative use case 133
Figure 58 Simulation results regarding cycle time for the administrative use case 135
Figure 59 Application of batch activity concept to the online retailer process 137
Figure 60 Simulation results for the retailer use case 139
Figure 61 Average waiting time until enablement of a batch cluster 140

LISTINGS

Listing 1 Threshold rule to exemplify the batch activation rule 66
Listing 2 Threshold rule with due date 66
Listing 3 MinMax rule as extension to consider future instances 67
Listing 4 Activation rule with fast track option 68
Listing 5 Batch activity representation in the BPMN XML 70

Listing 6 Definition of a Machine-Maintance-Event 92
Listing 7 Definition of a batch adjustment rule 93
Listing 8 Exemplary batch adjustment 93

ACRONYMS

BPM	Business Process Management
BPS	Business Process Simulation
BPMN	Business Process Model And Notation
BPMS	Business Process Management System
ECA	Event-Condition-Action
EPP	Event Processing Platform
CMMN	Case Management Model And Notation
CPU	Central Processing Unit
DES	Discrete Event Simulation
EDD	Earliest-due-date
EPC	Event-driven Process Chain
FCFS	First-come-first-served
FMC	Fundamental Modeling Concepts
IT	Information Technology
OLC	Object Life Cycle
UML	Unified Modeling Language
XML	Extensible Markup Language
YAWL	Yet Another Workflow Language

Part I

INTRODUCTION & BACKGROUND

1

INTRODUCTION

This chapter motivates this thesis and describes why an explicit design and configuration of batch processing requirements in business process models is needed. It provides the tackled problems, the research objectives, and the scientific contribution of this thesis. Finally, it concludes with providing an outline of the thesis.

1.1 MOTIVATION

"Businesses are made of processes" [23]. For running a successful business, organizations need to facilitate excellence in running their business processes. A business process is a set of connected activities which is carried out to reach a certain business goal and is continuously repeated [130]. A major driver for companies to focus on their business processes is to reduce their costs, to improve their productivity, as well as to increase the customer satisfaction [37]. Business Process Management (BPM) as a management discipline positions business processes in the center of its efforts. It is a tool for a controlled execution of business processes and their continuous improvement [23] by providing a set of concepts, methods, and techniques for the documentation, analysis, execution, and optimization of business processes [113]. "BPM has the power to innovate and continuously transform businesses and entire cross-organizational value chains" [122].

First interest in business processes

The interest in business processes has arisen in the late 1980s [35]. On the one hand, works were published on *statistical process control* (leading to Six Sigma) analyzing with statistical methods the performance and output of work to isolate *root causes* of performance problems [17]. On the other hand, works were presented on *business process re-engineering* identifying the need for continuous improvement of end-to-end processes [35].

These ideas led in the 1990s to information technology (IT) solutions where processes became first class citizens, particularly in *business process management systems (BPMSs)*, formerly called *workflow management systems* [115, 122]. A BPMS usually consists of a process-design component, in which business processes can be visually captured, and a process-runtime component, which guides the execution of the designed processes and interacts with the users [52]. Each business process executed in such a system mostly follows the process life cycle [26, 130] as shown in Figure 1. In the first phase, the *Design & Analysis*, the process is discovered by conducting interviews or workshops with stakeholders, or identified by analyzing execution data. Then, it is documented by a process modeling language, e. g., Business Process Model and Notation (BPMN) [71], Case Management Model and Notation (CMMN) [73], Unified Modeling Language (UML) Activity Diagrams [72], Event-driven Process Chains (EPCs) [98], or Yet Another Workflow Language (YAWL) [114]. The resulting process model is an abstraction of the real world process and describes its structure [52]. Such a model can be used to find process improvement possibilities for redesigning it. For executing it in a BPMS, the final process model is enriched with technical information, and then deployed on the system in the *Configuration* phase. During the execution in a BPMS in the so-called *Enactment* phase, a process can be monitored, and later execution data can be analyzed in the *Evaluation* phase for further improvements.

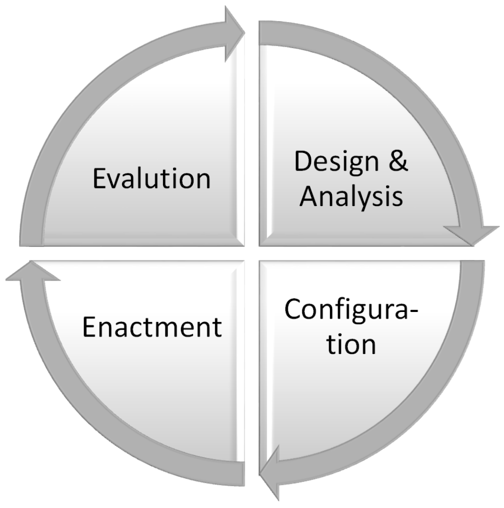


Figure 1: The business process life cycle, adapted from [26, 130]

In general, process models serve as blueprint for a number of process instances where one instance represents the execution of one business process case [130]. As Russell et al. [95] observed in existing BPMSs, "each of these is assumed to have an independent existence and they typically execute without reference to each other". This means that the instances of a process run independently from each other in such systems. Also during process model design, it is assumed that instances have no relation to each other.

Need for batch processing

In practice, we can observe cases where the consolidated execution of several instances is cheaper or faster than to process them individually. This means that groups of instances are collectively executed for specific activities in a process. For instance, in healthcare, it is more time-efficient to first collect a set of blood samples taken from patients to deliver them to the laboratory instead of bringing each one of them separately. In logistics, it is more cost-efficient to combine parcels to be sent to the same recipient instead of handling each parcel separately. Or, when calling a software service in a business process, bundling several service requests can help to reduce the number of service calls in order to save costs or avoid a denial of service. The approach used for such use cases – called *batch processing* [53] – allows business processes, which usually act on a single item, to bundle the execution of a group of process instances for particular activities to improve their performance.

1.2 PROBLEM STATEMENT

Missing support for batch processing in business processes

Currently neither on the process design level nor on the process execution level concepts exist to support batch processing in business processes: Existing process modeling languages (e. g., BPMN, UML, CMMN, EPC, or YAWL) do not support the explicit design or configuration of batch requirements in process models. In *Production workflow* [52] providing well-known concepts for process execution in BPMSs, Leymann and Roller propose a so-called *bundle activity*. However, this is focused on running multiple instances to handle several sub-items in one process. It does not discuss the synchronized execution of different process instances. Also the often referenced set of identified control-flow patterns for business processes by van der Aalst et al. [117], which describe the basic structural patterns for process modeling and execution, does not consider the batching of several process instances.

Batch Processing is well discussed in other research domains, especially in operations management. Here, various techniques are provided for production systems [60, 82] or queuing systems [63, 70] to schedule jobs or customers in a batch. However, it is not discussed how to integrate these techniques explicitly in business process models.

In practice, batching of instances is either enacted manually or hard-coded in software. Some BPMS providers, e. g., the Process Maker [41], allow to manually create batches during process execution in the user interface [40]. Since no support exists to configure the batch processing in the process model, it is also mainly driven at run time by the users. Organized manually, the rules of batch execution might be unclear for all process stakeholders. As the batch processing can not be enforced in the manual setting, it might be forgotten during process enactment resulting in lower process performance. If the batch execution is controlled by an external software, the batch configurations are not traceable for the process owner and the participants. Also, adaptations on them result in high efforts, because the IT department usually has to get involved.

Currently only a small set of theoretical contributions exists on integrating batch processing in business process models explicitly, not hidden in the system implementation, e. g., in [53, 69, 96]. These research works provide solutions for specific scenarios and do not have a complete overview of requirements. They are limited with respect to various aspects, for instance, incomplete automatic support, no support for multiple resources, or technical complex batch definition schema. Additionally, no prototypical implementations are given to evaluate the feasibility of the introduced concepts.

Summarized, batch processing techniques exist in other domains, but they are currently neither integrated in concepts on the process design level, nor on the process execution level leading to manual or hard-coded solutions, which are connected with higher efforts in maintaining, enforcing, and monitoring batch activities.

1.3 RESEARCH OBJECTIVES AND RESEARCH FRAMEWORK

Regarding the given problem statement, the overall goal of this thesis is to provide a concept to specify batch processing requirements on the process design level, and to support their automatic execution in a BPMS. The batch specification in a process model has the advantages that the rules of batch processing are explicit for all process stakeholders and can be better communicated. Further, the batch activity can also be included in the process simulation to validate it and to identify the most optimal configuration. The automatic batch execution in a BPMS avoids a manual work-around and supports an automatic batch monitoring.

Therefore, the main goal of this thesis is to introduce a new process modeling element for *batch activities*. We aim at a language-independent *batch activity concept* describing the syntax and semantics of batch activities which can be integrated in process modeling language having the concept of activities. In contrast to operations management, it is not the goal to design a new batch scheduling techniques, but we want to reuse proposed techniques, such that performance improvements can be achieved with a batch activity. Additionally, we also want to consider in this thesis flexibility needs with regards to batch activities, and as well multi-process batching.

Applied design science process

For reaching the research objectives, the work in this thesis follows the steps of the design science process proposed by Pfeffers et al. [76] and complies with the design science guidelines of Hevner et al. [123]. The design science process provides researchers in Information Systems a procedure to identify real world problems and to create useful research artifacts for solving them, and consequently contributing to the body of scientific knowledge [76]. We consider this process as suitable, because currently no elaborated concept for process modeling languages exists to specify batch activities which can also be automatically executed by a BPMS. The design science process as depicted in Figure 2 includes six steps: (1) problem identification and motivation, (2) definition of solution objectives, (3) design and development, (4) demonstration, (5) evaluation, and (6) communication.

Next to the design science process, it is illustrated in Figure 2 how we applied the process in the work of this thesis. This is described in more detail in the following:

1. *Problem identification and motivation*: The first step of the design science process is dedicated to define a relevant real world problem and to justify the solution [76]. Problem relevance, one of the design science guidelines by Hevner et al. [123], highlights the importance of building technology-based solutions which are

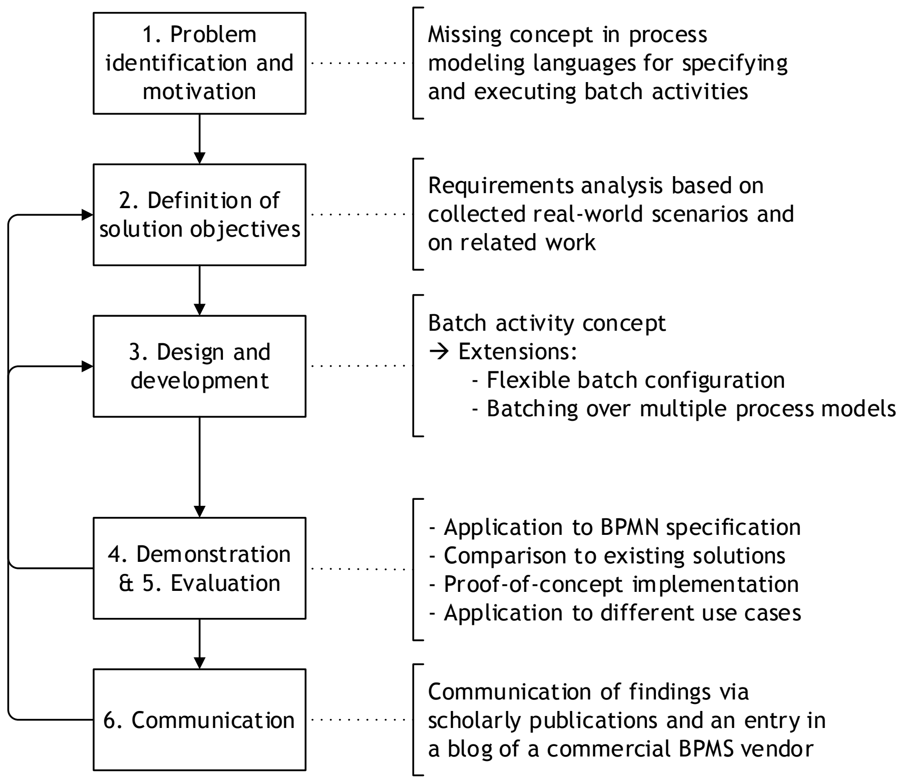

1. Problem identification and motivation
Missing concept in process modeling languages for specifying and executing batch activities
2. Definition of solution objectives
Requirements analysis based on collected real-world scenarios and on related work
3. Design and development
Batch activity concept
→ Extensions:
- Flexible batch configuration
- Batching over multiple process models
4. Demonstration & 5. Evaluation
- Application to BPMN specification
- Comparison to existing solutions
- Proof-of-concept implementation
- Application to different use cases
6. Communication
Communication of findings via scholarly publications and an entry in a blog of a commercial BPMS vendor

Figure 2: Design science process adapted from Peffers et al. [76] and its application in this thesis.

relevant for business problems. In line with this, the thesis aims at developing a concept to make batch activities explicit in process modeling languages, which are used by organizations to document and execute their business processes. Thereby, the general benefits of process models [42, 99] are also used for batch activities as for example, a better communication of batch configurations, their automation, and their monitoring.

2. *Definition of solution objectives*: The second step of the design science process focuses on the solution objectives by describing how a new artifact is expected to support the solution of a problem [76]. It could be observed that existing solutions on integrating batch processing in business processes are limited to the requirements of specific scenarios. This thesis provides a requirements analysis by analyzing the state of the art literature as well as several collected real world scenarios having a need for batch processing. Based on this, the objectives of this thesis are set.

3. *Design and development*: In the third step, the artifact is actually created [76]. The artifact of this thesis is a *batch activity concept* for process models. It is a meta-model for batch activities describing the syntax, more precisely the configuration parameters of a batch activity, and its execution semantics.

4. *Demonstration and*
5. *Evaluation*: The fourth step of the design science process, the demonstration, in which the artifact is applied to solve one or more instances of the problem, provides the input to the fifth step, the evaluation. The evaluation step is dedicated to observe and measure the artifact in the demonstration to determine whether the artifact is able to solve the problem, and whether the objectives are reached [76]. We evaluate the concept by means of an application of the *batch activity* concept to existing control-flow oriented process modeling language (such as BPMN, UML activity diagrams, and EPCs), a comparison of the concept to existing solutions, and a proof-of-concept implementation in an existing open-source BPMS. Further, single-case mechanism experiments [133] are conducted in which the batch activity is applied to different use cases in a simulated environment. Thereby, we want to study the effect of batch activities on the process performance in terms of time and costs – the two major process performance dimensions [26] – in order to predict their influence on real world processes.

6. *Communication*: The sixth and last step of the design science process recommends to communicate the developed artifact and its effectiveness to researchers and also practitioners. Also Hevner et al. [123] see the importance in this by defining the guideline to communicate the research. Following this, the resulted artifact and their evaluations are communicated via scholarly articles. Furthermore, we communicate our results also to practitioners by a entry[1] in a blog of a commercial BPMS.

By evaluating and by communicating the batch activity concept, it is improved and additional artifacts are developed. With several iterations through demonstration, evaluation, and communication back to design and development, this thesis provides two extensions of the batch activity concept. On the one hand, a concept to allow a flexible batch configuration, and on the other hand, a concept for batching over multiple business processes are developed, which are also evaluated and communicated as presented in this thesis.

1.4 SUMMARY OF CONTRIBUTIONS

In the light of the discussed research objectives, we want to outline the contributions of this thesis. Mainly, this work contributes with a concept to explicitly represent batch activities in business process models and to execute them automatically.

This can be further divided into the following sub-contributions which are also illustrated in Figure 3:

1 https://blog.camunda.org/post/2016/10/batch-processing-with-camunda/

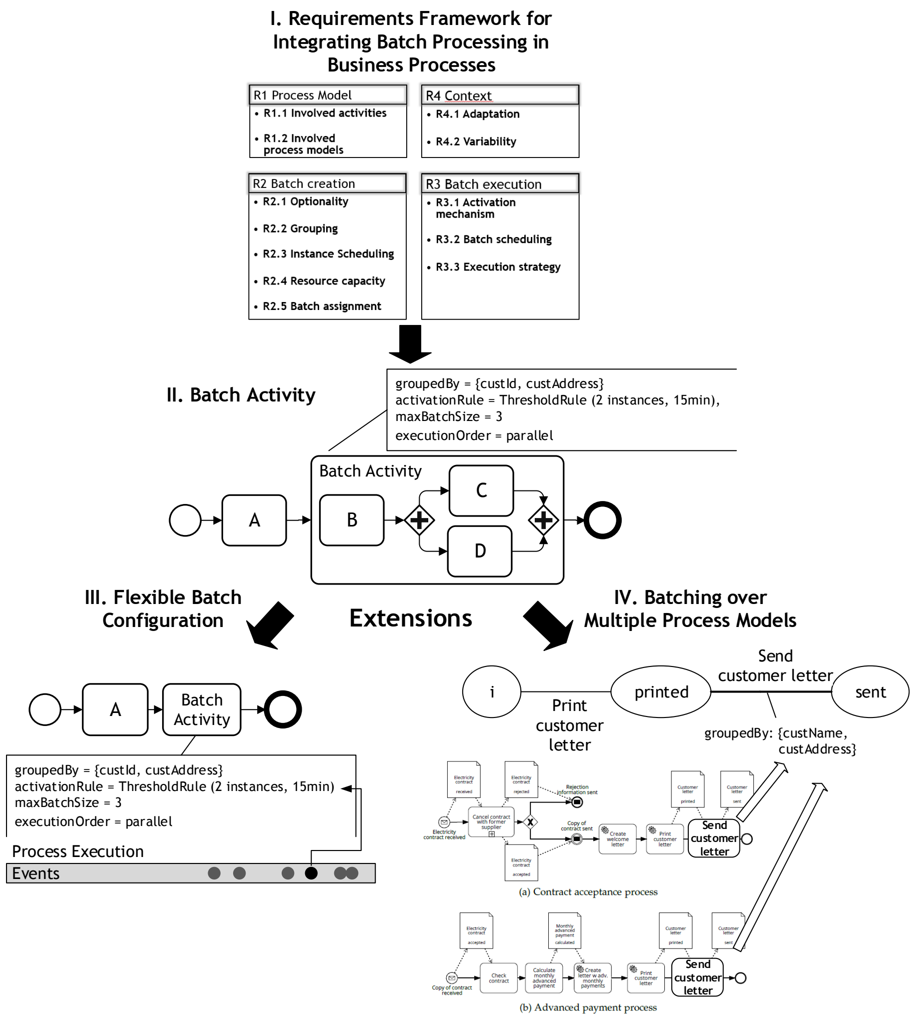

I. Requirements Framework for Integrating Batch Processing in Business Processes
R1 Process Model
R1.1 Involved activities
R1.2 Involved process models
R4 Context
R4.1 Adaptation
R4.2 Variability
R2 Batch creation
R2.1 Optionality
R2.2 Grouping
R2.3 Instance Scheduling
R2.4 Resource capacity
R2.5 Batch assignment
R3 Batch execution
R3.1 Activation mechanism
R3.2 Batch scheduling
R3.3 Execution strategy
II. Batch Activity
groupedBy = {custId, custAddress}
activationRule = ThresholdRule (2 instances, 15min),
maxBatchSize = 3
executionOrder = parallel
Batch Activity
A
B
C
D
III. Flexible Batch Configuration
Extensions
IV. Batching over Multiple Process Models
A
Batch Activity
groupedBy = {custId, custAddress}
activationRule = ThresholdRule (2 instances, 15min)
maxBatchSize = 3
executionOrder = parallel
Process Execution
Events
i
printed
Print customer letter
Send customer letter
sent
groupedBy: {custName, custAddress}
Send customer letter
(a) Contract acceptance process
Send customer letter
(b) Advanced payment process

Figure 3: Contributions of the thesis.

I. *Requirements Framework*: By analyzing the state of the art literature as well as several collected industry examples having a need for batch processing, this thesis provides a requirements framework for integrating batch processing in business process models. The requirements in the framework are structured into four classes: (1) *process model*, (2) *batch assignment*, (3) *batch execution*, and (4) *context*. The resulting framework covers the aspects which need to be considered when developing a concept to explicitly capture batch processing in process models. Further, it can support the comparison of existing solutions.

II. *Batch Activity*: The main contribution of this thesis is the batch activity concept. The batch activity is a new type of process modeling element to explicitly represent batch requirements on the model level. With it, a process designer can define the set of connected activities in a business process for which a batch execution is required. Further, several configuration parameters allow the process designer to individually setup a batch activity. Additionally, the batch activity is able to group process instances according to their specific context data. The resulting modeling concept is applied to the mainly used process modeling language BPMN [36] as well as UML Activity Diagrams, and EPCs to demonstrate its feasibility. For supporting its automatic execution, we provide an execution semantics and prove it through a prototypical implementation in an existing open-source BPMS.

III. *Flexible Batch Configuration*: The configuration of batch processing at design time does not always guarantee an optimal process execution, since exceptions occurring during process execution may influence the batch processing. This thesis presents a concept, based on event processing techniques, to allow flexible adjustments of different configuration parameters, if a relevant event occurs.

IV. *Batching over Multiple Process Models:* Organizations being active in BPM often manage large collections of process models in which also similar activities (or even fragments) in different process models can be found. This offers the opportunity to bundle activity instances of different business processes for an improved performance of several processes. An object life cycle (OLC) shows the data manipulations of a data artifact independently from process activities. With OLCs, identical behavior within business processes on data artifacts across the process model boundaries can be identified. Thus, this thesis provides a concept to specify batch requirements not only in an individual process model, but also in a centrally given OLC valid for several process models.

1.5 STRUCTURE OF THE THESIS

Following the design science process presented before, this thesis is structured as follows:

PART I: BACKGROUND After the introduction, this part covers the preliminaries as well as the related work. The preliminaries (Chapter 2) introduce existing concepts and formalisms for process modeling and execution which will be used in this thesis. In related work (Chapter 3), we elaborate on the state of art literature. On the one hand, the usage and techniques of batch processing in other domains are presented, such as operations management and computer science, where it originated from. On the other hand, support for batch processing in the BPM domain is discussed in detail.

PART II: BASIC CONCEPTS The second part focuses on the main concept. Following the design science process [76], a requirements analysis is firstly conducted based on the related work and on collected examples from industry originating from different domains by interviewing process experts. These scenarios and the requirements framework are presented (Chapter 4) in this part. Further, we discuss here the design objectives of the batch activity concept. Based on this, the concept of batch activities is introduced (Chapter 5). In this chapter, we first evolve *data views* as a technique to group process instances based on their data characteristics. This is followed by the modeling concept of batch activities and their execution semantics. Finally, the presented batch activity concept is discussed by comparing it to existing solutions based on the requirements framework.

PART III: EXTENDED CONCEPTS The discussion of the batch activity concept reveals that two main aspects are not considered so far. On the one hand the flexibility during batch processing, and on the other hand, batching over multiple business processes. This part centers around these two aspects. First, a concept for flexible batch configuration with events (Chapter 6) is presented. In this chapter, an analysis is given on how events may influence the batch execution. Based on this, batch adjustment rules are introduced. These can adapt a configuration of a batch activity, if a relevant event is identified. Secondly, batch processing across multiple business processes (Chapter 7) is introduced. With a motivating example, we start the elicitation of requirements. Then, the batch transition is presented as an extension of an OLC to indicate which related activities can be batched. Finally, an optional batch execution semantics is defined including a user approval to avoid undesired behavior.

PART IV: EVALUATION AND CONCLUSIONS The last part focuses on the evaluation of the batch activity concept and the conclusions. The prototypical implementation (Chapter 8) of the batch activity and of the multi-process batching concept in Camunda [13], an existing open-source BPMS, is discussed which serves as technical evaluation. Further, the application of the batch activity (Chapter 9) to two different use cases in a simulated environment is presented. It is used as validation of the basic concept to predict its effect on the performance of real world processes. Thereby, a simulation tool supporting batch activities, and performance measures for batch activities are presented. Finally, the thesis is concluded in which the contributions are summarized, and open problems as well as future research directions are elaborated.

2 PRELIMINARIES

This chapter proceeds with introducing concepts and formalisms for process modeling and execution. This thesis introduces a new process modeling element to capture batch activities and support their automatic execution. Therefore, in this chapter, process models are firstly formalized and then, the most common process modeling language Business Process Model and Notation (BPMN) is shortly presented. Additionally, concepts and formalisms with respect to a) data aspects in process models, b) the process execution, and finally c) events in business processes are introduced.

Business Process Management

Business Process Management (BPM) inspects the work conducted in an organization and tries thereby to exploit improvement opportunities [26]. While organizing and managing the work, the focus of BPM is on business processes. For this purpose, it provides a set of concepts, methods, and techniques to support organizations in the documentation, analysis, execution and optimization of their business processes [113]. By applying BPM, the key artifact which is created are process models [113].

A business process consists of a set of activities jointly realizing a business goal in an organization and technical environment; each process belongs to one organization, but may interact with other organizations [8, 130]. Whereas process orchestrations give a detailed view on the internal behavior of a business process with its to-be performed activities and their execution constraints, process choreographies describe the interaction between two or more business processes [21]. Choreographies focus on the public behavior of the collaborating entities [6].

Process life cycle

BPM initiatives usually follow a process life cycle [26, 130] (cf. Figure 1) in which the process model is first discovered, often captured with a process modeling language. After analyzing the discovered process with validation or verification techniques (e. g., soundness [121]), the process might be redesigned. The resulting process model is then mostly used for process implementation. The process implementation phase (or also called the process configuration phase) can be supported by different means. For example, a set of policies and procedures can be introduced, which employees need to follow, certain software services supporting the process can be implemented, or the complete process can be deployed in a business process management system (BPMS) – a dedicated software to support business processes [26]. Weske [130] defines a BPMS as a "generic information system driven by explicit process representations, e. g., process models, to coordinate the enactment of business processes." For executing a process in a BPMS, the process model needs to be enriched with technical information. During process enactment, the process gets monitored and controlled. Data captured in this phase can be used for process evaluation in which on the one hand, defined key performance indicators can be checked [61], or on the other hand, the conformance between execution and the existing model can be analyzed by using process mining techniques [112]. The evaluation results can lead to new process redesign requirements which trigger a new iteration of the process life cycle. In the following, concepts and formalisms regarding process modeling and execution are introduced which are later used as basis for developing the batch activity concept.

This chapter is structured as follows: First process models are formalized in Section 2.1, followed by a introduction into BPMN in Section 2.2, the state-of-the-art language for the graphical representation of business processes. Then, the representation of data in process models is defined in Section 2.3. This is followed by an introduction into aspects related to the process execution in Section 2.4, and finally events in the context of business processes are defined in Section 2.5.

2.1 PROCESS MODELS

Process models depict the behavior of a set of similarly executed procedures in an organization. As all models, they are an abstraction of the real world, and they are created with a purpose [131]; each process model has a modeling goal which can range from process documentation and staff training, over process redesign, to process implementation [8]. In [130], Weske distinguishes between three different abstraction levels for business processes: organizational, operational and implemented processes. Whereas *organizational processes* are usually high-level textual description of processes including the process inputs, outputs and responsibilities, *operational process models* capture the process activities, their relation as well as organizational information such as the assignment of activities to resources. *Implemented process models* extend and adapt the operational ones with technical aspects needed for the implementation, e. g., necessary data used in the process or service calls. In general, the set of activities and their control flow structure are captured by business process models. An extensive list of control flow structures, which might exist, are presented as control flow patterns by Aalst et al. [117]. In this thesis, we also want to consider the data perspective layer representing the processed data within a process, hence, we formally define a business process model as follows:

Process model definition

Definition 2.1 (Process Model).
A *process model* $m = (N, CF, D, DF, type_a, type_t, type_g)$ maps to

- a finite non-empty set $N = A \cup E \cup G$ of control flow nodes being activities A, events E, and gateways G (A, E, and G are pairwise disjoint),
- a control flow relation $CF \subseteq N \times N$ such that (N, CF) is a connected graph,
- a finite non-empty set D of data nodes whereby $N \cap D = \emptyset$,
- a data flow relation $DF \subseteq (D \times A \cup E) \cup (A \cup E \times D)$ specifying input and output data dependencies of activities or events.

Function $type_a : A \rightarrow \{task, subprocess\}$ assigns each activity a type, the function $type_t : A \rightarrow \{user, service, unspecified\}$ specifies the type of each task, and the function $type_g : G \rightarrow \{and, xor\}$ gives each gateway a type. ◀

An activity of a process model can be either non-decomposable – a $task$ – or can have internal behavior – a $subprocess$. A task can be further specified by a task type: a task can be either executed by users (i. e., $user$), usually via a user-interface, or automatically executed by a software service (i. e., $service$). Further, task can send or receive information from other business processes with which the process model interacts. This is usually realized in a BPMS with a user or service task [26]. Manual tasks as discussed by Weske [130] are not considered here, because we require that each process model can be executed by a BPMS. Therefore, manual tasks have to be transformed into user or service task as described by Dumas et al. [26]. An and-gateway allows for concurrent behavior in a process and xor-gateway allows to select between several paths in a process model. In the next section, the industry-standard for business process modeling, BPMN, is presented in which more details on process models are discussed.

2.2 BUSINESS PROCESS MODEL AND NOTATION

Several languages exists for specifying process models, for example BPMN [71], Unified Modeling Language (UML) Activity Diagrams [72], Event-driven Process Chains (EPCs) [98]. In this thesis, processes are represented with the BPMN notation as it dominates the process standards space [36]. An exhaustive discussion is not in the scope of this section; thus, we refer the reader to [11, 15, 32, 130, 132] for a more detailed introduction into the BPMN specification.

BPMN provides support for process modeling efforts on different abstraction levels – from an operational level to an implementation level. Thereby, BPMN focuses on business processes including organizational, data, information technology (IT)-system, and decision aspects. In case that more details are needed on the enumerated aspects, process models can be supplemented with other models, such as organizational charts, data models, or architectural models [11]. The standard was released in its current version BPMN 2.0 in January 2011 by the Object Management Group [71] and it includes different types of diagrams not only for process orchestrations, but also for process choreographies. Since this thesis will introduce a batch activity concept for process orchestrations, we will focus on *BPMN business process diagrams*. In the following, the core elements of BPMN process diagrams are presented followed by the running example of this thesis.

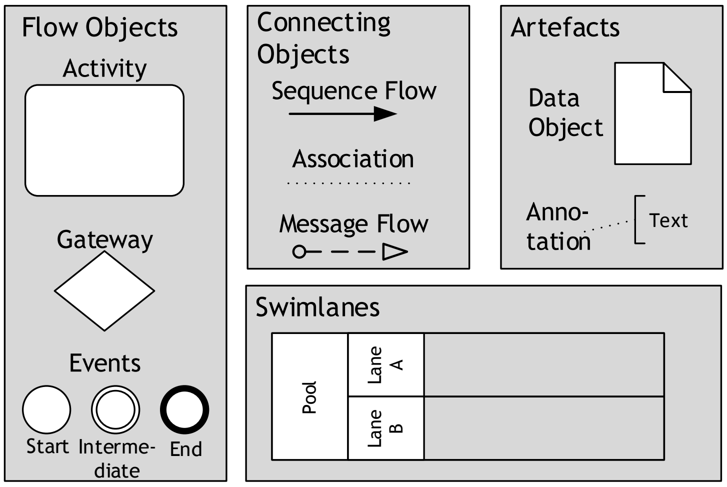


Figure 4: Elements of BPMN business process diagrams used in this thesis.

BPMN core elements

The core element set of BPMN business process diagrams can be distinguished in four categories [130]: flow objects, connecting objects, artifacts, and swimlanes as shown in Figure 4. Each element has a set of own attributes which can be used to refine the element in order to prepare the process implementation [132]. The BPMN specification is flexible enough to also allow adding customized elements or attributes to the meta-model by so-called *extension elements* and still being BPMN-compliant [71]. In the remaining, we want to present for each main category those elements which are used in this thesis.

Flow objects BPMN describes a process as a graph of flow objects that can be activities, gateways, or events [71]. Activities are units of work; they require time to be performed and can be atomic or non-atomic. An atomic activity is a task which can not be further decomposed, and a non-atomic activity is a sub-process with internal behavior. Atomic activities can be supplemented with a task type characterizing it further, e. g., a *user task* conducted by task performer by interacting with a system, a *service task* calling a software service, or a *message task* sending or receiving a message to/from a partner.

Gateways are required to specify more advanced behavior than sequences of flow objects. They have the generic shape of a diamond and can be enriched with a symbol specifying their behavior. Here, we want to focus on exclusive gateways (expressed by a $\times$ marker) and parallel gateways (expressed by a $+$ marker). Exclusive or xor-gateways are used to select one of several alternative paths or to merge them. For selecting a path, the condition expressions of the outgoing flows are evaluated and the first one evaluating to true is triggered. In case none of the condition expression evaluate to true, the default path is selected, if available. Parallel or and-gateways can trigger and synchronize several concurrent sequence flows.

The *Event* element is a dedicated mechanism of the BPMN specification to represent that processes can handle certain events (catching events) of their environment or that they provide events (throwing events) to their environment. Events in BPMN take no time and represent that a certain state has been reached. Three event types are distinguished in the BPMN specification: start, intermediate, and end events. Whereas start events are always catching and indicate when a process is instantiated, intermediate events specify events occurring during process execution which can be catching or throwing. End events are throwing and indicate where a path of a process will end. Each type of event can be further differentiated by triggers, e. g., message, signal, or error.

Connecting objects and swimlanes

Whereas sequence flow connects the flow objects describing the order between them, associations represent a link between an artifact and a flow element or a pool. Message flows are used to indicate message exchange between different process participants (i. e., individual, an organizational entity etc.), each running an individual process represented by a pool. Pools can be further sub-divided by lanes which represent different task performers within an organization; a *task performer* is responsible for the activities inside its lanes. A task performer can be a specific individual, a group, an organizational role or position [11].

Artifacts

Artifacts are contextual information which can be added to business process diagrams, e. g., data objects and annotations. The data objects are structured information which are consumed or produced by activities or events. Each data object has a name which usually references a data class. It can optionally reference a data state representing the state of the data contained in the data object. We will discuss data aspects in process models in more depth in the next section. Process designers can use annotations to add explanations to any flow object.

Running example

In the following, we want to introduce an online retailer process as running example shown as BPMN diagram in Figure 5. As indicated by the message start event, the process is initialized as soon as an order

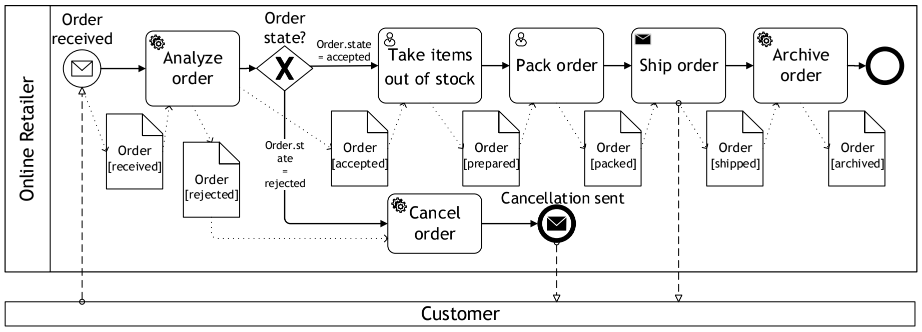


Figure 5: Online retailer process depicted as BPMN process diagram.

by a customer is received. The customer is depicted as collapsed pool which has an own process, but from which we abstract. The receiving message event creates as output an *Order* data object in state *received*. The created *Order* object is read by the subsequent service task *Analyze order* which decides whether the order is accepted or rejected. This is indicated by the two output data objects, one in state *accepted* and one in state *rejected*. If the order is accepted, the upper branch is executed on which a robot takes the ordered items out of the stock, and then a warehouse employee packs them into a parcel. Afterwards, the parcel is shipped represented by a sending message task, and then archived by a software service. We assume that the sending or receiving message tasks are implemented in a BPMS as service or user task. In case the order is rejected, a service task cancels the order and the process ends with a cancellation message sent to the customer.

2.3 DATA ASPECTS IN PROCESS MODELS

Following the BPMN specification, created and consumed structured data in processes is represented by so-called *data objects*. As this term is usually associated with the run time representation, we use the term *data nodes* representing the access on data in process models as introduced in Definition 2.1.

We refer to a data node $d \in D$ being read by an activity $a \in A$ or an event $e \in E$, i. e., $(d, a), (d, e) \in DF$, as *input data node* and to a data node d being written by an activity a or event e, i. e., $(a, d), (e, d) \in DF$, as *output data node*. The above given process model in Figure 5 has one start event, two alternative end events, six activities, one gateway, and multiple data nodes read and written by activities and events. Each data node has a name, e. g., *Order*, and a specific data state, e. g., *received* or *sent*. Data nodes sharing the same name refer to the same data class;

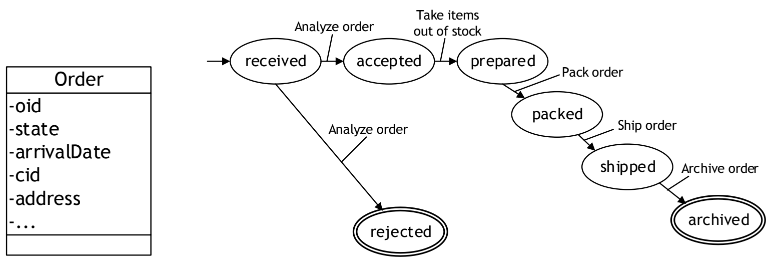
Data class

here: *Order*. A data class describes the structure of data nodes (or data objects) in terms of attributes and possible data states. A data object is the run time representation of a data node consisting of a value for each attribute. Data objects are formalized in the subsequent section focusing on process execution aspects. A data node (object) maps to exactly one data class being formalized as follows:

Definition 2.2 (Data Class)**.**
A *data class* $c = (name, J, olc)$ maps to

- a $name$,
- a finite set J of data attributes
- an object life cycle olc.

◂



(a) Data class attributes.

(b) Object Life Cycle.

Figure 6: Data class *Order* for the *online retailer* process with its object life cycle.

We refer to C_m as set of data classes utilized in one process model m. Figure 6 shows the *Order* data class of the retailer process with an excerpt of its data attributes, such as *oid*, *state*, *arrivalDate*, *cid*, and *address*. Further, its object life cycle (OLC) is depicted as labeled state transition system. An OLC [45] represents the logical and temporal order of data states in which a data node (or a data object) could be. A data state [64] denotes a situation of interest for the execution of a business process. For instance, state *accepted* of object *Order* indicates that the activity *Analyze Order* is completed successfully and that the items of the order can now be taken out of the stock. Supplementing process models, OLCs describe allowed data manipulations by process activities for one data class.

Object life cycle

Definition 2.3 (Object Life Cycle)**.**
An *object life cycle* $olc = (S, s_i, T, \Sigma, \zeta)$ maps to

- a finite non-empty set S of data states, and an initial data state $s_i \in S$,
- a data state transition relation $T \subseteq S \times S$ describing the logical and temporal dependencies between data states,
- and a finite set Σ of actions representing the manipulations on data objects (S and Σ are disjoint).

Function $\zeta : T \rightarrow \Sigma$ assigns an action to each data state transition. ◂

OLC denotes the set of all object life cycles, such that the funtion $lc : C \rightarrow OLC$ returns for a given data class c its olc. Based on this, a data node is formalized as follows:

Definition 2.4 (Data Node).
A *data node* $d = (c, name, s)$ maps to

- a data class c,
- a $name$,
- and a state $s \in S, lc(c) = (S, s_i, S_F, T, \Sigma, \zeta)$ where S denotes the set of data states of the olc of the corresponding data class c.

◀

A data transition of an OLC with its assigned action can be used by activities in different process models. Such an activity has a input data node being in the input state of the transition and has an output data node in the output state of the transitions. For instance, the activity *Pack order* in Figure 5 transforms the *Order* object from *prepared* to *packed* as also described in Figure 6b. In case OLCs are not available, it is also possible to derive them automatically from the process models in a repository [65] by extracting the life cycle states from the data nodes and the transitions from the process activities. Regarding the interplay of the process and the data side, we assume process models and the OLCs of the data classes C_m to be consistent; i. e., all data manipulations induced by some activity in the process model are covered by data state transitions in the corresponding object life cycles (cf. the notion of *object life cycle conformance* [50, 66]). Further, we assume that each activity acts at least on one data node. That means that an activity utilizes at least one single data state transition of an object of some data class such that function $\epsilon : A \rightarrow \mathcal{P}(U)$ with $U \subseteq \bigcup T_{OLC}$ (all transitions of all object life cycles) returns a set of transitions for a given activity. For reducing complexity of discussion, we apply the connection between process models and object life cycles through consistent label matching, i. e., the activity label of the same activity refers to the same action. Alternatively, in practice, string matching techniques (e. g., in [110]) could handle this connection.

2.4 PROCESS EXECUTION

In this work, we assume that business processes are executed by means of a BPMS. Therefore, the architecture of such a system is discussed in this section. Furthermore, process instances representing the execution of business process cases are formally defined in this section.

BPMS architecture

An architecture of such a BPMS is given, for example, in [26, 52, 130]. In Figure 7, we represent the BPMS architecture as *Fundamental Modeling*

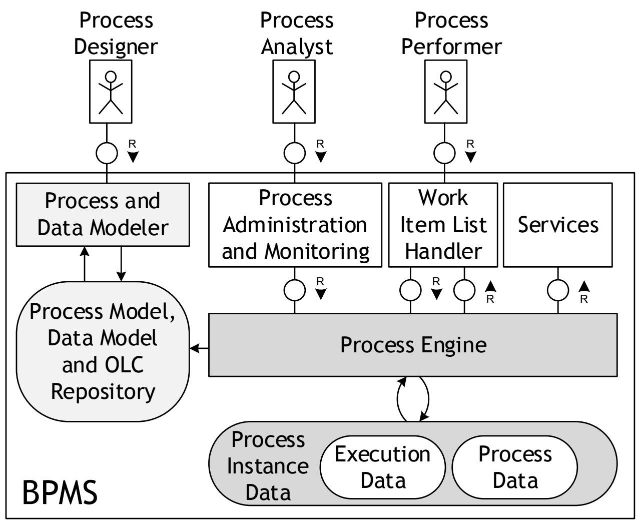


Figure 7: Architecture of a BPMS focusing on the execution of business processes and considering process data aspects.

Concepts (FMC) block diagram [47] and extend it by data aspects. FMC block diagrams are used to model the static compositional structure of software systems. Whereas the ellipses represent data storages on which the active software components – shown as rectangles – can have read or write access, actors which interact with the system are represented as rectangles with a stylized figure of a human. Process designers shown as actors can design business processes, data classes, and object life cycles with the *Process and Data Modeler* component which stores them in a repository. The main component which guides the execution of business processes is the *Process Engine* which can access all repository elements. Process participants can start processes over the *Work Item List Handler*. Then, a process instance is initiated; executions of process models are represented by process instances with each instance i belonging to exactly one process model m. As soon as a process instance is started, the process engine initiates the process activities in their prescribed sequence.

Work items

In case of a user task, the process engine creates a work item, which realizes the link between a to-be executed activity instance and its task performer [130]. A *task performer* is a process participant which is responsible to execute a specific task. If more than one task performer for an activity are available, several work items are created. The work item which is selected by one of the task performers is executed while the others are *withdrawn* [39, 130]. Several work items are, for instance, created in case of *Role-based Distribution* [95], where one or several task performers are part of a role, e. g., a clerk. If this role is assigned as resource for an activity, then all members of this role get a work item provided as soon as it is executed. An extensive list on resource allocation patterns can be found in the work of Russell et al. [95].

In case of a service task, the respective service is called by the process engine. For each state change of a process instance, e. g., start or termination of an activity instance, the process engine logs it with a time stamp in the *Execution Data* storage, such that a BPMS is able to provide execution logs for tracking the execution of a process. Further, the engine is able to read and update data processed by the business processes in the *Process Data* storage. We assume that all data objects created by the process instances are made persistent and can be access by the process engine. The process analysts can monitor the process instances by means of the *Process Administration and Monitoring* component. After introducing the BPMS architecture, the following subsection continues with formalizing process instances.

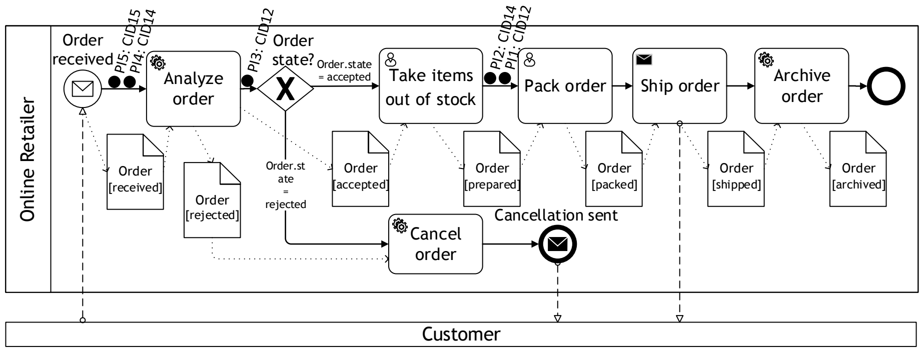


Figure 8: Online retailer process diagram with exemplary running instances represented as labeled token. The label, e.g., PI1: CID12, references the respecting `process instance id` and `customer id`.

Process instance

In Figure 8, several process instances are visualized in the example retailer process by labeled tokens. The label of the token gives information about the `process instance id` and the `customer id` referencing the customer who has placed the order. Each process instance has an id and contains a set of activity instances – the run time representation of an activity – and a set of processed data objects. The current state of a process instance is given by the states of its activity instances and its processed data objects. For example, process instance with the `id 5` in Figure 8 has for all activities an initialized activity instance, besides the *Analyze order* activity instance being *enabled*, and its *Order* object is in state *received*. In the following, first activity instances and then data objects and their states are formalized.

Activity instance Each activity instance traverses through different life cycle states during its execution [130] as shown in Figure 9. With start of a process instance, each activity is initialized; the instance is in the *init* state. As soon as the incoming flow(s) of an activity are triggered, the instance gets *enable*d and is in state *ready*. With the *disabled* state, activity instances can be temporally disabled, where it is not accessible by the

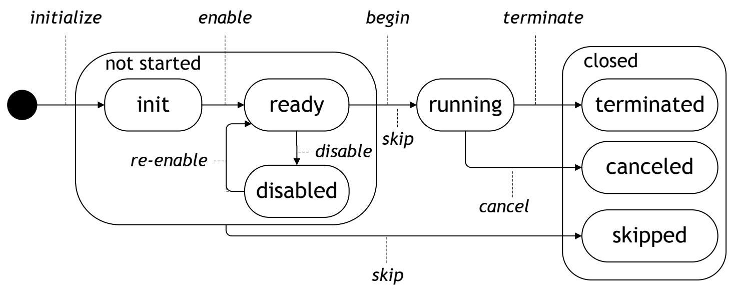


Figure 9: Life cycle state transitions of activity instances, adapted from [130].

process performer or a software service, and can later be *re-enable*d. When the task performer *begin*s to execute an activity instance in state *ready*, the state changes to *running*, and finally the activity instance is *terminated*. In case that a different path is selected as the one on which the activity of an instance is during the execution of a process instance, then the activity instance transfers from *not started* into the *skipped* state. If an attached boundary event occurs, a running activity instance transitions into the state *canceled*. Based on this activity life cycle, an activity instance is formally defined as:

Definition 2.5 (Activity Instance)**.**
An *activity instance* $ai = (i, a, z)$ maps to

- a process instance i, $ai \in AI_i$ where AI_i is a set of activity instances of the process instance i,
- an activity a, and
- a current state $z \in \{init, ready, disabled, running, terminated, skipped, canceled\}$

◀

Data object

For each process instance, an arbitrary set of data objects exists. Formally, a data object is defined as follows:

Definition 2.6 (Data Object)**.**
A *data object* $o = (c, \rho)$ maps to

- a data class c describing its structure and
- a data state ρ which comprises a valuation of all attributes J_c of the data class c. Let V be a universe of attribute values. Then, the *data state* $\rho : J_c \to V$ is a function which assigns each attribute $j \in J_c$ a value $v \in V$ that holds in the current state of object o.

◀

At any point in time, a data object is in exactly one data state. The state may change over time through updates performed by activities or events which are represented in process models by data output nodes, i. e., $a \in A, e \in E, (a, d), (e, d) \in DF$. We assume that a data class attribute $j_{state} \in J_c$ exists which represents the explicit OLC state and that, for any data object, $\rho(j_{state})$ returns as value a OLC state $s \in$

S, $lc(c) = (S, s_i, S_F, T, \Sigma, \zeta)$ of the data class c to which the object belongs. O denotes the set of all data objects stored in the BPMS process data storage.

With the definition of activity instances and data objects, a process instance is formalized as follows:

Definition 2.7 (Process Instance)**.**
A *process instance* $i = (m, AI, O_i, \alpha)$ maps to

- a process model m,
- a finite, non empty set of activity instances AI,
- a finite set of data objects $O_i \subseteq O$ which are created or updated by the process instance i.

The function $\alpha : AI \rightarrow A_m$ is surjective and ensures that to every activity $a_m \in A_m$ in the set of activities of the process model m at least one activity instance $a_i \in AI$ for the process instance i exists. ◄

The state of a process instance i changes either if an activity instance $ai \in AI$ has transitioned into a new life cycle state or the state of a data object $o_i \in O_i$ has been updated.

2.5 EVENTS AND BUSINESS PROCESSES

In Section 2.2, *events* are presented as modeling construct of BPMN. *Event* elements are used to represent that a specific type of event (i. e., happening in the process environment) is relevant for a process and can be received by it (catching event), or that a specific type of event is produced by a process (throwing event) [71]. Catching events in process models can either be produced by other business processes running in the same BPMS, or external events.

Internal events of a business process are "state changes of objects within the context of a business process" [138] where such objects can be, among others, activities, data objects, or the entire process. Typical events are for example the activity transitions which are shown in Figure 9, such as the *enabling* of an activity. The logging and distribution of such events is usually controlled by the process engine.

External event An external event represents a "real world happening occurring in a particular point in time at a certain place in a certain context" [38], for instance, weather warnings, traffic updates, stock ticks. The integration of event processing and business processes is an emerging field [48]. It takes the advantage of a wide variety of event sources to improve business processes [55], for example, by triggering processes or specific activities to react flexible on certain situation, by using information of events for an improved decision making in processes, etc. An external event is either present in an external event source, or an Event Processing Platform (EPP). As event sources produce events in diverse formats, and one event source can be relevant for several business processes, it is more efficient that the detection of events being relevant for process execution is conducted with EPP [7] to which the BPMS then needs to be connected.

A set of associated events which are totally ordered in time form an *event stream* [29]. EPPs are able to perform different operations on event streams (e. g., transformation or aggregation) and can derive "higher-level knowledge from lower-level system events in a timely and online fashion" [43]. Further, the EPP notifies subscribers about certain events and provides them in a structured form [29]. A BPMS can be such a subscriber, and hence, can receive events, and react on them according to the process specification [43].

Structured events

A structured event is a derivation of an event object consisting of an identifier, a time stamp, and some event content in the payload, e. g., a set of key-value-pairs or a tree-structure expressed in the Extensible Markup Language (XML) [38]. A structured event type describes a class of structured events that have the same format. A business process executed in a BPMS can be the receiver of such an event by subscribing to it via a catching event in the process model. Such a structured event can either start a process instance (if the structured event is referenced in the start event), continue with the process execution (if it is referenced in a intermediate event), trigger an exception (if it is referenced in a boundary event), etc. Thereby, such an event can also update a data object of a process instance i, if there is a data output node d written by the catching event e, i. e., $(e, d) \in DF$ in the process model m of i.

After having introduced the needed concepts and formalisms for the work of this thesis, the next chapter discusses the related work based on which the objectives of this thesis are set.

3 RELATED WORK

This chapter elaborates on the work that is related to the research presented in this thesis. It discusses existing works on batch processing in general and in the Business Process Management (BPM) domain. These works serve as basis for a requirements analysis for integrating batch processing into business processes, which is introduced in succeeding chapter. This chapter is partly based on the published paper "Requirements Framework for Batch Processing in Business Processes" [86].

The term *"Batch processing"* has been around for several decades: In *operations management*, batch processing is a method to produce similar products in batches. Those so-called *batch processes* give the flexibility to provide a variety of products, but try to reach efficiency by producing in batches [107]. Further, batch services handling groups of customers are studied in operations management with the help of queuing theory. In *computer science*, batch processing is a common term used for programs processing a series of jobs one after another without any user interaction [108].

In this chapter, we first look in Section 3.1 into operations management and computer science where batch processing originates from. Afterwards, the current support of batch processing in the BPM domain is studied in Section 3.2, and in particular, existing concepts to explicitly represent batch processing on the process model level are discussed. Finally, a summary of related work is given in Section 3.3.

3.1 ORIGINAL DOMAINS OF BATCH PROCESSING

For introducing batch processing to business process models, we first want to study how batch processing is used in other domains. Thereby, important concepts might be observed which are also relevant to be considered for business processes. First, batch processing in *operations management* is discussed, where it originates from, followed by batch processing in *computer science*.

Operations Management

Batch processing is especially studied in the field of operations management focusing on the processes which produce the products or services of an organization [107]. In particular, operations management supports the design, execution, and control of operations which convert resources into desired goods and services [2, 14, 59]. Manufacturing processes can include batch (flow) processes [59] where identical jobs are processed or transported in batches to utilize the same machine or tool setup [107]. In service systems, customers may be also treated in batches. Especially, entertainment services (e. g., roller coasters) operate in this way that the service is only provided, if a certain number of customers is available [134]. Such *batch services* are studied with the help of queuing theory [63]. This subsection has a closer look on *batch processes* as well as on *batch services*.

BATCH PROCESS In operations management, several types of processes are distinguished which are different in terms of how the materials or information are processed [107]. When focusing on production processes, those are categorized in *jobbing processes*, *batch processes*, *mass processes*, and *continuous processes* as shown in Figure 10.

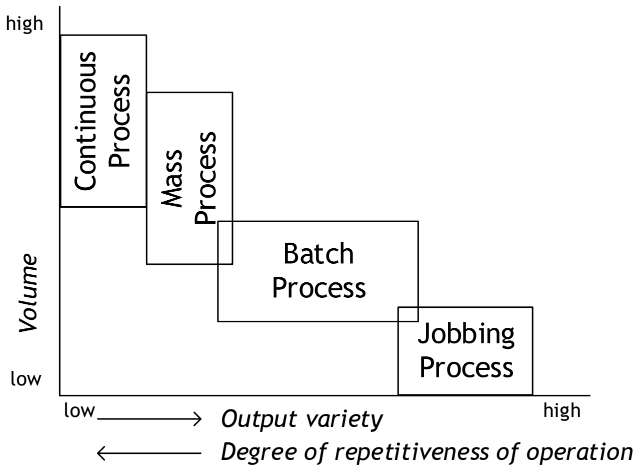


Figure 10: Different type of production processes compared with regards to the volume produced, the output variability, and the degree of repetitiveness of the operation, adapted from Arora [2]

Jobbing processes are characterized by producing a great variety of products with low volumes; they are designed to meet specific customer requirements [107]. A *batch process* is a special type of *jobbing process* which is used to be more efficient by processing similar jobs in a batch, but still having the flexibility to produce a variety of products in different volumes [59]. Because of this, batch processes support a wider range of volumes and variety levels than other processes [2, 107]. Usually, in such a system, general purpose machines are utilized which can be adjusted to produce different outputs [2]. Finally, *mass processes* and *continuous processes* are much more efficient by producing high volumes, but allowing a low variety of products [107]. Whereas in *mass processes*, the production follows a pre-defined sequence of steps, in a continuous production, the flow is continuous rather than discrete [14].

For the design of a batch process, especially the physical layout of such a system is important where the diversity of products, their individual flow patterns, and production volumes have to be considered [2].

Together, with the physical layout also the sequence of activities has to be designed. Applied process modeling languages in manufacturing, such as *scientific-management* flowcharts (representing the flow of production) or *information-system* flowcharts [107], do not represent batch activities explicitly in the charts.

Scheduling of batches

Further, a research challenge is the scheduling of jobs in a batch process which is concerned to determine the batch size as well as the order of the jobs processed [2]. One the one hand, the time required to process a job strongly varies because of differences in setups, processing requirements, etc. [59]. On the other hand, the jobs run through different stages, where each stage is scheduled individually, but the stages need to be coordinated [59]. With scheduling, organizations can reach different goals as for example, minimizing the average flow time, minimizing average number of jobs in the systems [59]. Thereby, different scheduling policies can be used, such as earliest-due-date (EDD) (i. e., jobs are ordered according to their due date), critical-ratio (i. e., jobs are ordered according to the amount of time until the due data divided by remaining amount of processing) etc. [59]. Depending on the scheduling goal, an optimization problem is designed considering the scheduling environment, as well as job and family characteristics [34]. The optimization problem is then solved with mathematical programming method (e. g., , dynamic programming), heuristics, or simulations [60]. In [60, 82], surveys on scheduling algorithms with batching are given. Thereby, two types of scheduling models are distinguished, the *serial* batching (i. e., similar jobs are scheduled together to save setup costs being still executed in sequence) and *parallel* batching (i. e., batches of jobs which can be executed at the same time). The surveys show that scheduling algorithms are designed specific to an individual situation of an organization, for instance, processing stage, type of jobs, and goal of scheduling.

Additionally, exceptions and variability during a batch process are also recognized in scheduling, and studied [107]. Several works on dynamic scheduling due to resource-related (e. g., machine breakdown) or job-related (e. g., job cancellation) issues are presented in the review of Ouelhadj and Petrovic [74].

Optimal batch size

An important question of batch processing is the size of a batch. Whereas large batches in a batch process have the advantage that the machine utilization is high, because of a reduced number of setups, smaller batches can decrease the overall processing time of a customer order, because a customer order might be scattered over different batches [82]. As large batches are connected with relatively high in-process inventories between the processing stages, *Just-in Time* initiatives try to reduce the batch size in batch processes to shorten the cycle time of jobs [59].

BATCH SERVICE Service operations can also profit from batching [105, 106]. Simons and Russel [105] present a case study of batching in mass service operations selecting the example of a court. The authors infer from interviews and observations that batching can also help to reduce setup and processing time/cost in mass services, but may increase flow time similar as in the manufacturing domain. Additionally, they recommend to consider for batching decisions in service operations the *complexity of scheduling* and the *utilization of resources* which can fluctuate due to batching [105].

In queuing theory, it is a long tradition to study the batch service problem [63] which was first considered by Bailey [3] who describes it as follows: "Customers arrive at random, form a single queue in order of arrival and are served in batches." Queuing theory provides different techniques to analyze systems with queues, and allow to calculate important parameter on them, such as the expected waiting time, or the expected length of a queue [63, 137]. These techniques are also used for quantitative analysis of business processes, but they can only give measures on one activity at a time [26].

In queuing theory, *parallel* batch processing is mainly discussed, where it is assumed that a resource with a capacity greater one exists which can handle several customers in parallel (e. g., a roller coaster, or a blood testing machine) [3]. The primary object of investigations is the identification of the optimal point in time to start a batch based on analytical methods [63]. Thereby, different optimization policies are proposed.

Optimization policies

An often discussed optimization policy is the *threshold rule*, originally called the *general bulk service rule* [70]. It states that a batch is started, when the length of the waiting queue with customers is equal or greater than a given threshold (i.e., a value between one and the maximum server capacity) and the server is free [70]. The rule can be also extended by a maximum waiting time, such that a group of less than the threshold is also served, when a certain waiting time of the longest paused customer is exceeded [70]. Several studies (e. g., [100, 102, 104]) investigate how to determine an optimal threshold value, as well as expected waiting time and service cost under varying assumptions concerning the distribution of arrival and service times, as well as capacity constraints of server and queue (an overview is, for example, given by Medhi [63]). The *versatile threshold rule* presented, for example, by Maity and Gupta [57] is a generalization of the rule where the threshold is a random value between a given lower and upper bound. Cost can be also used as control limit in an optimization policy as studied by [20, 126, 127]. Thereby, the waiting cost of customers are also considered by associating waiting times with certain cost, for instance, the losses in future sales [18]. If the cost of waiting are higher than the service cost and the server is free, then a batch is activated.

Computer Science

Batch Processing Operating Systems

Batch processing in computer science originates from the days of punch cards where it was time-consuming to provide input to mainframes [108]. The earliest operating systems were *batch processing operating systems* with the main goal to avoid cost-intensive idle time of the central processing unit (CPU) by processing batches of user jobs without any need of a user interaction [22]. In such a system, a human computer operator provides a sequence of user jobs, the so-called *batch*, where the jobs are independent to each other. All functionality on a batch, such as accepting the batch, scheduling it, allocating memory, and returning the results in a file are taken over by a *batch monitor*. Similar to operations management, different scheduling algorithm are used with the focus to complete a batch in a limited amount of time [108]. Examples are the first-come-first-served (FCFS) (i. e., jobs are ordered as provided), or the shortest-job-first (i. e., jobs are ordered based on the predicted time a job needs) [108]. The performance of such systems is evaluated by the *CPU utilization*, by the *throughput time* (i. e., the amount of work done per unit of time), and by the *turnaround time* (i. e., the time needed from proving a job until returning the results of it) [108]. The *turnaround time* includes not only the time required for the batch execution, but also the time needed to accept a batch and to return the results. Still, the utilization of the CPU was the most important aspect, because running a mainframe was connected with high costs in the 1960s and 1970s [108]. With the development of computers, other operating systems (e. g., time sharing or distributed operating systems) were developed to provide more user convenience and faster response times [22].

Batch applications

Still, batch (processing) applications are today used in interactive systems to process large amount of data on a regular basis [9]. For example, they support the computation of the monthly bill of telephone customers, the generation of tax reporting documents, or the bulk loading of data from a online transactional system to a data warehouse [9]. In computer science, mainly the design, the implementation, and the optimization of batch application is studied, for example in [5, 9, 27]. Also different frameworks exist for supporting the development of batch applications, for instance `Jem The Bee`[1] or `SpringBatch`[2] for Java. The general idea is that a batch application takes as input a record-oriented file, whose records represent a sequence of request messages [9]. The main goal of batch applications is to process large amount of data in a short time frame. Thus, the emphasize is to reduce the number of data base accesses [5], such that the order of the records might be optimized with regards to the processed data. For realizing a fast data access, also persistence frameworks should support this by enabling bulk-reads and bulk-updates, as well as streaming of persistent data [5]; an evaluation

1 http://jemthebee.org/
2 http://projects.spring.io/spring-batch/

of different frameworks in this regard can be found by Barcia et al. [5]. As result of a batch job, a new file is created with the output for each job, such that the input file is still unmodified to repeat the batch program in case it fails [9]. A batch job can be scheduled at a given time, when sufficient capacity is available (typically at night or weekend) [9]. The response time of a batch job is not important, but usually it has to finish in a certain batch window after which the results are awaited, and online transactions on the data have to be again possible [9, 27]. Strategies to optimize batch jobs, specifically in the *IBM mainframe z/OS* environment, are discussed by Ebbers et al. [27].

In business process management systems (BPMSs), batch applications might be also used in their implementation or in the implementation of some service tasks. For example in Camunda [13], an open-source java-based BPMS, a batch processing interface is provided to process developers. The batch processing interface allows process developers to design and execute asynchronous maintenance jobs over multiple process instances in the process engine, e. g., for migrating, for canceling, or for deleting process instances [12]. However, such batch features are usually hidden in the system implementation and cannot be used on the process model level to configure a batching of several process instances. An exception is commercial BPMS provider *Process Maker* [41]. In this system, process designers can specify for a process activity that a consolidation of activity instances is allowed, and they can design how the consolidation is visualized. The actual selection of activity instances for a batch has to be done manually in the user interface during process execution [40]. Thus, batch activities are here only supported on the user interface, but not by the process engine itself. After discussing batch processing in BPMS, the next section elaborates on research works with regards to batch processing in the BPM domain.

3.2 BATCH PROCESSING IN BUSINESS PROCESS MANAGEMENT

Need for batching

In general, a need for batch processing in business processes was identified since a decade until today by different research works: For instance, van der Aalst et al. [118] list *batching* as an escalation strategy to avoid deadline violations. They propose to group tasks in batches, for instance based on their location, and assign them to one resource. As limitation of this strategy, the authors mention that batching efforts, e. g., the time to form a batch, are involved. In business process redesign in which an existing process is improved regarding its flow time, cost, quality, or flexibility [26], batch processing can be an option for improvement. Reijers and Mansar [93] have collected a set of heuristics which can stimulate redesign ideas including the *case-based work* heuristic. This heuristic states that it might be helpful for some processes to remove activities with batching, but in others it might be beneficial to introduce batch processing to improve the process flow time or cost.

Recently, Fdhila et al. [30] present a classification and formalization of instance-spanning constraints, among which the execution of instances in batches is mentioned. In the area of process mining, Martin et al. [58] recently recognized that resources can organize their work in batches, and that the batching behavior also influences the process performance. With their work, different types of batch activities (i. e., serial, simultaneous, concurrent batching), and metrics (e. g., batch size, duration of a case in a batch, or waiting time) on them can be discovered to get insights in their business value. However, the mentioned research works give no details on how exactly batch processing should be incorporated in business process modeling and execution. In the following paragraphs, we first discuss the limited support of batch processing in the BPM domain. Then, closely related work and their limitations are presented, followed by works which focus on additional aspects, such as flexibility during batch processing.

SUPPORT FOR BATCH PROCESSING IN BPM Existing process modeling notations (e. g., Business Process Model and Notation (BPMN) [71], Case Management Model and Notation (CMMN) [73], Unified Modeling Language (UML) Activity Diagrams [72], Event-driven Process Chains (EPCs) [98], or Yet Another Workflow Language (YAWL) [114]) do not support the explicit representation of batch processing needs in business processes. This is further illustrated in the next chapter, the requirement analysis. There, we present examples how batch processing is integrated in BPMN processes with the current set of modeling elements and discuss resulting issues and limitations.

Production Workflow by Leymann and Roller [52] summarizes and presents important technical concepts for BPMSs. Thereby, the authors introduce a so-called *bundle activity*. This allows on running multiple activity instances to handle several sub-items of one process instance and to bundle their results at their termination to provide them to main process instance. However, it provides no support for collecting different process instances at specific activities to execute them in groups.

The control flow patterns [117], an extensive list of control flow structures that might exist in process modeling languages or BPMSs, include no pattern related to batch activities. A common assumption is that batch requirements can be solved with *Multi-Instance* control flow patterns [117], which provide a means to trigger multiple instances of an activity in the scope of a process instance and synchronize their termination; these instances are still independently handled. This pattern is similar to the *bundle activity* proposed by Leymann and Roller [52]. As batch processing aims at collectively executing a set of instances, batch requirements needs to be differently incorporated. The resource patterns *Piled Execution* and *Simultaneous Execution* in [95] cover some aspects of batch processing: While *Piled Execution* pipelines and allocates similar tasks to one resource whereby the execution is still done in sequence, *Simultaneous Execution* allows users to work on several tasks simultaneously while focusing on the switch between them rather than on a collective execution of several work items.

CLOSELY RELATED RESEARCH WORKS Currently, only few research works, e. g., [49, 53, 69, 77, 96, 116] exist in BPM on the integration of batch processing which are discussed in the following.

Batching in process fragment-oriented approaches

The process fragment-oriented approaches *Proclets* by Aalst et al. [116] and *Philharmonic Flow* by Künzle and Reichert [49] highlight the need for batch-oriented activities to handle sets of data entities (e. g., a set of reviews for a paper, a set of applications for a job offer) in an activity. This means, batch-oriented activities are simply used to handle a set of multiple data objects created within the context of one process instance, but not to synchronize the execution of several process instances. In control-flow oriented process modeling languages as BPMN, this is solved by a read or write association between an activity and a data object list.

The research works by Sadiq et al. [96], Liu et al. [53], and Natschläger et al. [69] incorporate batch processing explicitly into process models by introducing a new type of activity, called *combound activity*, *batch activity*, or *combined-instance activity*. Since these works are closely related to the work of this thesis, they will be discussed in detail in the next paragraphs.

Combound activity by Sadiq et al.

Sadiq et al. refer in their work [96] to a "contradiction between the preferred work practice and some fundamental principles behind workflow systems", and identify in different scenarios, an e-learning process and an order process, the need to group several activity instances for specific tasks in order to work on them in parallel. The authors differentiate between an auto-invoked and user-invoked assignment of activity instances to batches. They also observe that certain scenarios require a specific grouping of instances regarding their data characteristics, whereby the authors state that this should be mainly user-driven. Batch processing is enabled in their work by establishing *compound activities* in process models with a grouping- and ungrouping-function. It generates one static activity instance based on several ones and splitting it after task execution. As stated, the grouping-function can be either auto-invoked with a predefined number of required instances, or user-invoked, creating a batch with user-selected instances. Waiting on a pre-defined number of instances has the drawback that process instances might get stuck or have high waiting time if the required threshold is not met. The user-invocation leads to a manual batch organization where rules are not explicitly defined and errors can occur. Since this work sketches a first solution proposal and does not aim at a complete solution, a detailed execution semantics is not given.

Batch activity by Liu et al.

The *batch activity* by Liu et al. [53] aims at integrating the *batch service problem* of operations management into business processes. Hence, they request instances arriving at a batch activity that they firstly have to be collected in a queue and, then they are assigned to batches based on an algorithm. Thereby, the activity instances should be grouped according their data characteristics. The batch assignment and the scheduling of a batch on a resource should be conducted at an optimal point in time. For this, they follow only the threshold rule by [70] according to which the batch assignment and scheduling is activated when the resource is free, and a certain time (or number of instances) is reached. Thereby, the authors have a strong limitation that the batch activity is only processed by one resource. The functionality of the grouping and scheduling algorithm is not further discussed in this work. Proposals for solving this can be found in a later works [54, 129]. In [54], a technique is proposed to group activity instances regarding their input data based on attributes selected at design time. As the author assume that a group of instances is then consolidated into one batch instance, their provide additional data operation techniques for this transformation. Wen et al. [129] discuss a batch scheduling algorithm for minimizing the total waiting time and cost of the activity instances, but ignore the grouping of instances, and do not discuss the influence of the algorithm on the overall batch activity concept. Based on the introduced batch activity [53], also a discovery technique by Wen et al. [128] is proposed to support the design of batch activities. Thereby, the authors limit themselves to parallel batch activities, and assume that only one representative instance of a batch has event data for the batch execution and the others not. Based on this assumption, the discovery algorithm identifies the batch activities and to which batch each instance belongs in order to complement the event data. The configuration details of a batch activity are not discovered.

Combined-instance activity by Natschläger et al.

Natschläger et al. propose in [68, 69] a *combined-instance* activity to process several instances of an activity in parallel. They highlight that constraints, such as resource capacities and the type of instances that can be processed together, should be incorporated. Batch processing is considered by the authors as non-compulsory, only if an optimal batch for an activity instance exists, it is activated. For each activity instance arriving at a combined-instance activity, different batch candidates might exist and the most optimal batch has to be identified. Thereby, instances which can arrive in future should be also taken into account. As the authors are focused on the production and logistic domain, a batch should be scheduled on a resource. A batch is activated if all future instances have arrived at the combined-instance activity or a certain due date is reached. For the identification of the optimal batch, possible batch candidates are created considering the given constraints. Then, the optimization function provided by the process designer is solved, and the most optimal batch is taken. This selected batch has then to be monitored, if the future instances will still arrive (in time). In case that a batch does not come off, its assigned resource was unnecessarily reserved. A concrete implementation in a BPMS is not given for evaluating the proposed optimization and scheduling algorithm regarding the system performance. During batch execution, instances of a batch are merged and split later again. Thereby the authors do not discuss how it is ensured that instances keep their data. Further, a complex definition of constraints and an optimization function are required for solving the optimization and scheduling problem which might have a negative influence on the usability of the combined-instance activity. All presented concepts in this section focus on single batch activities only.

WORKS COVERING FURTHER ASPECTS Additionally to the presented parallel batch concepts above, Aalst et al. defined in [115] that "batch processing is when an employee is able to perform a number of work items of the same type ... without switching back to the worklist handler", similar to the *Piled Execution* resource pattern. Based on this idea, Pflug and Rinderle-Ma [77] provide a dynamic queuing approach to classify arriving instances at critical activities based on instance attributes. The work by the authors is further extended in [78] to sequences of activities. In these approaches, instances need to be collected, and if the resource is free, they are dynamically clustered into groups, which are then allocated to resources for sequential execution. This approach is only useful for activities with long waiting queues. Classification is done by an algorithm, but certain setup time is needed to identify the most suitable classification mechanism for a use case.

Sequential batching by Pflug et al.

Flexibility

Flexibility is beside time, cost, and quality, one important performance dimension and multiple research efforts focus on the enabling of flexibility in business processes. Reichert and Weber distinguish four major flexibility needs, namely support for variability, looseness, adaptation, and evolution [92]. Research work by Wong et al. [135] exists to allow flexibility of batch activities, in particular it supports the adaptation ability. In their work, the authors present a monitoring concept of manual batch activities in order to react on exceptions.

3.3 CONCLUSION

Batch processing is a common technique in *computer science* to process large amount of data without any user interaction. Although batch processing might be used in the system implementation of existing BPMSs, batch activities in process models to handle groups of similar process instances together are not fully supported yet. Nevertheless, the design concepts of batch processing systems given in computer science and the evaluation measures (e. g., *turnaround time*) might be beneficial for the realization of batch activities in a BPMS. A difference is that in case of batching user tasks, the task performer should be able to enter input to the batch.

Batch processing in *operations management* is used to process similar products or groups of customers in batches to be more efficient. Thereby, the size of a batch is an important objective of investigation, because larger batches help to reduce the execution costs, but can lead to increased cycle times as it requires certain time to fill the batch. This leads additional to longer queues or extended in-process inventories. Also in business processes, batch processing can be used to lower the average execution costs per instance as several instances are served in one batch. Batching can be done in parallel, but as well as in sequence to save setup costs. Nevertheless, cycle time is an important performance indicator in business processes [26]. On the one hand, the cycle time of an instance can increase by waiting for other instances to be batched with them. On the contrary, batching can also decrease processing times by handling a set of instances in one step. Summarized, an concept on integrating batch processing into business processes needs to assure an optimal trade-off between time and cost. For this, in manufacturing processes, optimization problems for scheduling customer jobs are defined and solved, each being specified for a specific setting. In contrast, queuing theory assumes that to-be processed cases arrive randomly and are assigned to batches which applies well for many business processes. They provide optimization policies for the trade-off between time and cost – batch activation rules – which can be reused for business processes.

A small set of research works exist in the *BPM domain* which allow an explicit representation of batch requirements on the process model level, not hidden in the implementation. The existing solutions are currently bound to specific use cases and do not have a comprehensive understanding of requirements of different scenarios: The *compound activity* by Sadiq et al. [96] is mainly driven by users selecting and assigning manually instances to batches. Whereas Liu et al. [53] target processes, where the batch activity is only executed by one single resource. Natschläger et al. [69] requires the definition of complex constraints and an optimization function for solving an optimization and scheduling problem. Such a complexity is mainly relevant in transportation and manufacturing processes. In the work of this thesis, we target the limitations of existing works by developing a concept to model and execute batch activities being applicable to different type of business processes.

Part II

BASIC CONCEPTS

4 REQUIREMENTS ANALYSIS

This chapter presents a requirements analysis for integrating batch processing in business processes. The related work shows that existing solutions are linked to specific scenarios and lack on a complete understanding of requirements. The following requirements analysis is based on related work, and furthermore, on real world scenarios taken from different domains. The resulting requirements framework provides an overview of aspects which need to be considered when developing a batch activity concept for business processes. Further, it fosters the comparison of existing solutions. This chapter is based on the published paper "Requirements Framework for Batch Processing in Business Processes" [86].

The previous chapter, which has discussed related work, revealed that existing solutions to explicitly represent batch processing in process models are linked to specific scenarios, and lack on a complete understanding of requirements. This chapter provides a requirements analysis based on which a batch activity concept is introduced in the following chapters.

For the requirements analysis, we also interviewed experts from different domains about business processes with batch activities. Further, we scanned forums of existing business process management system (BPMS) providers (e. g., Bizagi [10], Camunda [13]) to collect scenarios having a need for batch processing. Thereby, we could identify examples how batch processing is currently integrated in process models with existing process modeling elements. In this chapter, two of those example are presented, and related issues with the proposed solutions are discussed. With the insights from the scenarios and the requirements of the related work, this chapter presents a requirements framework for integrating batch processing into business processes. The requirements framework captures aspects to be considered, while designing a batch processing concept for business processes. These aspects are used to set the design objectives of the *batch activity concept* introduced in this thesis. Additionally, it should foster a comparison of existing and future solutions.

The chapter is structured as follows: Firstly, Section 4.1 introduces the collection of scenarios from different domains requiring batch processing and study their needs to complement existing research work. These, together with the discussed literature, provide the basis for the requirements framework presented in Section 4.2. This section also discusses the completeness of the framework as well as an application of the framework to structurally compare the requirements of the collected scenarios. Finally, the objectives for the *batch activity concept* and a prioritization of the given requirements are discussed in Section 4.3.

4.1 SCENARIOS REQUIRING BATCH PROCESSING

Scenarios from industry and their requirements were collected by interviewing experts, and scanning forums of existing BPMS providers (e. g., Camunda [13], Bizagi [10]). The interviews were conducted with internal process analysts of the organization. They were asked open questions regarding details on the batch processing use case, its current implementation, and challenges. First, the scenarios are presented in an overview. In the second half of this section, two examples are given, illustrating how batch processing is tried to be integrated with existing modeling elements, and their related issues are discussed.

Overview of scenarios

Table 1 presents eight identified scenarios from five different domains: health care, retail, manufacturing, business administration, and finance. For each scenario, its origin, a short description and the problem context are introduced.

Table 1: Scenarios requiring batch processing and their requirements

Scenario	**Origin**	**Description**	**Problem Context**
Health Care			
SC-1 Blood Sample Test Process	Expert interview with a Dutch Hospital	If a blood test is needed, it is taken and brought by a nurse to the laboratory where the test is conducted. Usually a nurse delivers several samples to save transportation time. In the laboratory, several blood samples from different wards are collected to save machine costs.	- Batch activities are currently manually organized; automation can help to fill batches optimally and to consider the expiration of blood samples - Machine and nurse have an upper limit in the number of blood samples which they can handle simultaneously - Blood samples are ordered based on their expiration time at the laboratory - In case of emergencies, blood samples have to be immediately transported to the laboratory

Scenario	Origin	Description	Problem Context
Retail			
SC-2 Order process	Expert interview with a German Retailing Company; Entry in Camunda Forum[a]	Customers place orders on the online retailer website. For each accepted order, the articles are taken out of stock. Then, they are packed in a parcel and shipped. Often, no transportation costs are charged with the effect that customers place multiple orders in short time frames. In such situation, orders of the same customer could be packed and shipped together to save shipment costs.	- Expert interview $\rightarrow$ batch processing rules are hard-coded, cannot be easily accessed by stakeholders - Camunda Forum $\rightarrow$ tries to implement it with existing BPMN elements leading to a complex workaround where activation rules cannot be specified - Grouping of cases in specific batches by customer is needed - Priority customers need a special treatment - Batch processing spans over several activities - Need for batching is optional, only if other orders of a customer exist batch processing should take place
SC-3 Process of shipping orders abroad	Expert interview with a German Retailing Company	Shipments to other countries are collected. Collected orders of one day to one country are then transported to the respective country, where they are handed over to a local shipment company.	- Manual rule that activates a batch of orders for one country once a day $\rightarrow$ efficiency could be increased by using a batch activation rule balancing the cost savings with the waiting times of the customers - Grouping of cases in specific batches by country is needed
SC-4 Return handling process	Expert interview with a German Retailing Company	Customers can return some or all ordered items in a defined time frame. In case of advanced payment, they are reimbursed. Every financial transaction is associated with costs such that the retailer waits some time in case that another return is received to bundle them. In case that all items are returned or the time frame is closed, reimbursement is done immediately.	- Batch processing rules are hard-coded, cannot be easily accessed by stakeholders, the activation considers whether the return is complete and whether the return time frame is still open - Need for batching is optional, only if returns are still open and the return time frame is not closed - Grouping of cases in specific batches by customer is needed
Manufacturing			
SC-5 Manufacturing process	Expert interview with a British Manufacturer of Glasses	On the production line, some activities require that the work orders are processed as a batch. Here, the employees manually release the batches to the line when the previous batch was finished. Further, all work orders with a manufacturing fault are collected and batches of them are sent back for reprocessing.	- Batch activities are currently manually organized $\rightarrow$ automation would help to fill batches efficiently and to consider maximum processing time - Production line has an upper bound in the number of work orders which it can handle simultaneously - Batch processing spans over several activities

Scenario	Origin	Description	Problem Context
Business Administration			
SC-6 Leave application process	Entry in Camunda Forum[b]	If an employee wants to take vacation, the request needs a manager's approval. In case of many requests, manager prefer to work on the bulk of cases in one user view and decide each case.	- Batch is manually created and activated by the manager - User processes the instances here in sequence, despite the visualization should be in one user view - Grouping of cases in specific batches by responsible manager is needed
SC-7 Invoice processing process	Entry in Bizagi Forum[c]	If an invoice is received, it is forwarded to the responsible individual for approval. A common practice is not to approve each incoming invoice immediately, but to check regularly the set of received invoices to minimize the time to get familiar.	- Stakeholders tried to implement use case with existing BPMN elements leading to a complex workaround - Batch should be activated after a certain time - User processes the instances of a batch here in sequence - Grouping of cases in specific batches by responsible person is needed
Finance			
SC-8 Send customer notification	Expert interview with a German Online Bank	Certain events, e. g., the opening of a bank account, trigger several processes which all send out a customer notification (e. g., welcome letter, credit card, ATM card). Although those notifications are created by different processes, they can be batched for sending only one letter to save shipment costs and increase the customer experience.	- Batch processing of customer notification created by multiple processes is considered as important, but not realized today - Grouping of cases in specific batches by customer is needed - Need for batch processing is optional, only if other notifications for a customer should be sent out, batch processing should take place

[a] `forum.camunda.org/t/building-a-batch-through-a-process/1722`
[b] `https://groups.google.com/d/msg/camunda-bpm-users/nJoPZg7dLo4/0Q-OpHVHFQAJ`
[c] `feedback.bizagi.com/suite/en/topic/add-existing-entities-to-a-collection`

Suitable activities for batching

The given scenarios in Table 1 show that batch processing is needed in different domains. In the given scenarios, batch processing is often manually organized where an automated approach can increase efficiency. It can be observed that most of the candidates for batching are routine activities with a high processing rate, such as transportation tasks or fully automated tasks on machines. Some candidates, like the leave application or the invoice processing, involve also a decision by a user, but they are still highly repetitive activities. Based on this, it can be assumed that batch processing is useful for routine activities processing a high number of cases which can be executed automatically, but as well with user-involvement. These types of processes have a higher rate of created batches which have in turn an higher influence on the overall process performance.

From the forum entries, it becomes apparent that stakeholders already try to support batch processing by applying existing process modeling elements, but these workarounds are complex and error-prone. In the following, two examples are shown and discussed.

Workarounds to integrate batch processing

Master-instance approach

The first example depicted in Figure 11 is about the shipment part of the *order process* presented in SC-2. Here, batch processing should allow to process the goods of shipments to the same address together. As soon as a shipment is received, it is checked whether another exists to the same address. If not, then some time is passed – expressed by a timer event– until it is further processed. After waiting, it is checked again whether other shipments to the same address exist. But now the current shipment instance is the so-called *master* instance to which the other existing ones might be added. If similar shipments exist, the loop sub-process organizes that the goods of each similar shipment are added to the currently processed one and a message is sent out to each one to terminate it. Independently whether instances are added or not, the shipment is processed and sent. With this, the process terminates.

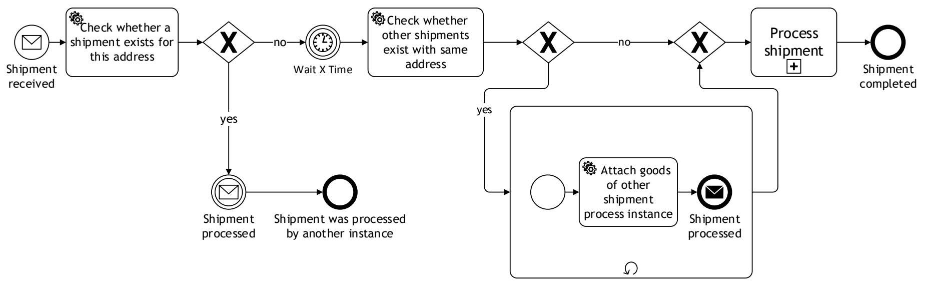
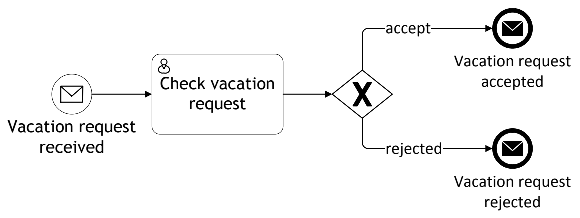
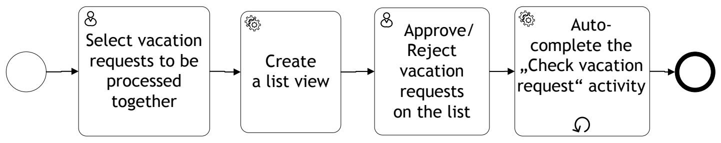

Shipment received
Check whether a shipment exists for this address
no
yes
Wait X Time
Check whether other shipments exist with same address
no
yes
Process shipment
Shipment completed
Shipment processed
Shipment was processed by another instance
Attach goods of other shipment process instance
Shipment processed

Figure 11: Shipment process as proposed in the Camunda Forum[1] to allow batching of shipments to the same address.

The presented solution has two major draw-backs. On the one hand, the process structure is quite complex, and the batch activity (here the *Process shipment* activity) as well as its configuration is not immediately visible, because several preparation activities had to be added. On the other hand, it includes several issues: For instance, message events are indented for inter-process communication by the BPMN specification, but not for intra-process communication [71]. Further, no upper bound of a batch can be indicated in this solution. The synchronization between the process instances has to be organized indirectly over the process data, which is error-prone. Thereby, it has to be ensured that after the *master* instance finishes the *Check whether other shipments exist*-activity, no new instantiated process instance can identify this shipment instance again. Otherwise, it would wait forever without being processed. Another issue is that the instances of the attached shipments are already terminated before the shipment actually is conducted. Results of it can not be written back into the original shipment instance, and the monitoring of those attached instance is not correct, because they end before the shipment itself happens.


Vacation request received
Check vacation request
accept
rejected
Vacation request accepted
Vacation request rejected

(a) Leave application process


Select vacation requests to be processed together
Create a list view
Approve/ Reject vacation requests on the list
Auto-complete the „Check vacation request" activity

(b) Administrative batch processing process

Figure 12: Leave application process as proposed in the Camunda Forum[2] to allow batching of vacation requests.

Approach with administrative batch processing process

Another option to the *master-instance* approach is to have an administrative process organizing the batch processing as shown in the example in Figure 12. This example shows the *Leave application process* as presented in SC-6. The basic process of the leave application as modeled in Figure 12a prescribes that a received vacation request is checked by a manager who either approves or rejects it. The process ends with the respective message to the employee who has sent the request. In order to check several vacation requests together, the manager does not work on the individual *Check vacation request*-work items. Instead, the manager would start the *Administrative batch processing process* shown in Figure 12b. The first activity of this process allows the manager to select a number of vacation requests. The selected list is then visualized by the next service activity in one user view. After rejecting or approving vacation requests in the list view, the *Auto-complete* service activity terminates then the enabled user tasks of the *Leave application* process and stores the respective data in each instances of the process.

Also this solution is structurally complex, because it involves a second process model including several pre-and post-processing steps for the batch processing. Additionally, the batching is mainly organized manually in this solution. The user has to start the administrative batch processing process, has to select the instances for batching, and has to start the batch activity. The administrative process only supports in creating the consolidated work list, and in writing the results back . Further, no correct monitoring is possible as in the previously example, because the actual start of the *Check vacation request*-activity can not be correctly logged.

Summarized, both solutions have several issues, and are individual solutions for specific use cases. In this chapter, requirements for batch processing in business processes will be analyzed to create a general batch activity concept easy to use, and useful for different application areas.

4.2 REQUIREMENTS ANALYSIS

The requirements framework presented in this section gives an overview on the requirements which have to be considered when developing a batch processing concept for business process models. Besides, the framework can help comparing the capabilities of existing and future solutions.

Applied method

For the requirement analysis, we firstly deduced all mentioned requirements from examining related work, and summarized them. The result was compared with external scenarios and extended where necessary. Then, the collected requirements were classified. The resulting framework was then discussed, and improved with a group of six BPM experts. The final result is presented below.

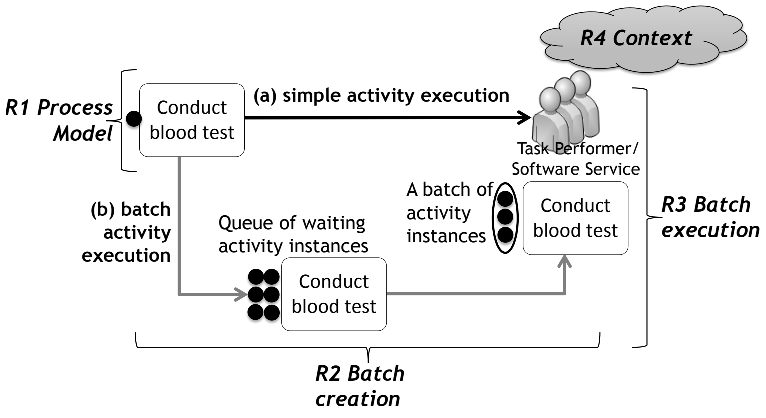

R4 Context
R1 Process Model
Conduct blood test
(a) simple activity execution
Task Performer/ Software Service
(b) batch activity execution
Queue of waiting activity instances
Conduct blood test
A batch of activity instances
Conduct blood test
R3 Batch execution
R2 Batch creation

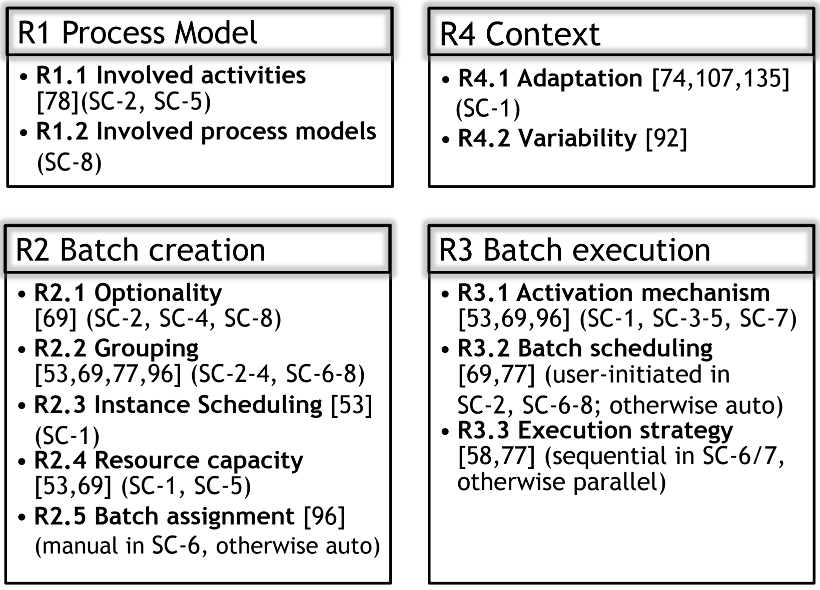
Figure 13: Classification of requirements for integrating batch processing in business processes.

Classification of requirements

First, the requirements classification is introduced. Figure 13 visualizes a simple activity execution in comparison to an activity execution with batch processing behavior – a batch activity execution. As an example, the *Conduct blood test* activity from the presented healthcare process (cf. Table 1) is taken. Based on this, Figure 13 also shows the four requirement classes: *R1 Process Model*, *R2 Batch Creation*, *R3 Batch Execution*, and *R4 Context*. Unlike the simple activity execution, an activity instance in the batch execution is not directly offered by the process engine to a task performer or software service, because it is firstly paused to allow the instance being grouped with others. Since batched activity instances can originate from one or several connected activities in one process model as well as in different process models, a *Process Model* class is defined. From the queue of activity instances, batches are created, and then provided to task performers (or services), where the batch execution takes place. Hence, a *Batch creation* phase can be distinguished from a *Batch execution* phase; each one of them is represented as a class in the framework. Changes in the execution context of a process instance can influence both, simple and batch activity execution. These changes can require certain reactions, e.g. handling of exceptions during the batch creation or execution, which are reflected by the *Context* class.

Requirements Framework

Figure 14 depicts the resulted requirements framework. After each requirement's name, its origin – the related work in brackets and the scenarios in parenthesis – are shown. Next, the requirements of each class will be presented in detail.



R1 Process Model
- **R1.1 Involved activities** [78](SC-2, SC-5)
- **R1.2 Involved process models** (SC-8)

R4 Context
- **R4.1 Adaptation** [74,107,135] (SC-1)
- **R4.2 Variability** [92]

R2 Batch creation
- **R2.1 Optionality** [69] (SC-2, SC-4, SC-8)
- **R2.2 Grouping** [53,69,77,96] (SC-2-4, SC-6-8)
- **R2.3 Instance Scheduling** [53] (SC-1)
- **R2.4 Resource capacity** [53,69] (SC-1, SC-5)
- **R2.5 Batch assignment** [96] (manual in SC-6, otherwise auto)

R3 Batch execution
- **R3.1 Activation mechanism** [53,69,96] (SC-1, SC-3-5, SC-7)
- **R3.2 Batch scheduling** [69,77] (user-initiated in SC-2, SC-6-8; otherwise auto)
- **R3.3 Execution strategy** [58,77] (sequential in SC-6/7, otherwise parallel)

Figure 14: Requirements framework for integrating batch processing in business processes.

R1 Process Model

R1 PROCESS MODEL. A batch operation in a business process can involve one or several activities being part of one or several processes. Thus, aspects which have to be considered during setting up batch processing in business processes are the involved activities and involved process models.

(R1.1) Involved activities: In several scenarios, we observe that batch processing operations can span not only over one activity but also over several connected ones, whereby the activities can be also executed by different actors. In the scenarios, batch operations are not interrupted. Therefore, the activities need to be connected. A single-instance activity between two batch activities would lead to two different batch operations.

(R1.2) Involved process models: Batch processing can be conducted for process instances of one process model, but as well for process instances of various process models. In most of our given scenarios, batch processing is bound to one process, but in SC-8 the activity `Send out notification` is used by several processes, such that several created notifications to one customer can be collected and sent together. Organizations being active in Business Process Management (BPM) often manage large collections of processes where similar activities (or even process fragments) can be found in different processes [124]. Repetitive activities (or process fragments) offer the chance to use batch processing benefits and reduce costs over instances of multiple process models.

R2 Batch Creation

R2 BATCH CREATION. In the batch creation phase, activity instances are assigned to batches. *Batch assignment* requires the following five aspects to be considered.
(R2.1) Optionality: First of all, it must be taken into account whether batch processing is *optional*. Either each process instance of a batch activity is used for batching (i.e., (1) batching is required), or only the ones which can form a batch (i.e., (2) batching is optional) whereas all other instances are processed as single instance. For optional batch processing, the decision can be based on all enabled instances of a batch activity, as well as on instances which will reach the batch activity in a future state [69].
(R2.2) Grouping: In studies from queuing theory, it is discussed that customers may be homogeneous or heterogeneous in their demand [75]. In case of homogenous demands, the process instances can be assigned to any available batch. If they are heterogeneous, then they have to be grouped in homogenous batches, such that different types of batches have to be formed. Also in several scenarios, a grouping of cases in specific batches by customer (SC-2, SC-4, SC-8), by country (SC-3), or by responsible employee (SC-6-7) is necessary. In [53, 69], grouping is conducted based on context data of process instances, where a grouping specification is given at runtime. A dynamical grouping for each batch assignment iteration is proposed in [77].
(R2.3) Instance Scheduling: All relevant activity instances which are used for batching are queued for the assignment to a batch. Hence, the application of a scheduling policy is necessary, e. g., first-come-first-served (FCFS) [108], earliest-due-date (EDD) [59]. If no prioritization of process instances is necessary for the batch assignment, FCFS can be applied as it could be also observed in most of the collected scenarios. For a prioritization of instances, for example based on their due date, other scheduling policies need to be used. For instance, EDD is useful in the health care scenario SC-1 to make sure that blood samples do not expire. Further, scheduling policies are presented in operations management, e. g., in [59].
(R2.4) Resource Capacity: An important constraint of a batch operation is the maximum capacity of the resource responsible for batch execution

[53, 69], also discussed in operations management (e. g., in [3, 82]. For instance, the blood testing machine may be able to test at maximum 50 blood samples in a run. This capacity determines the maximum size of a batch.

(R2.5) Batch Assignment: The process instances collected at a batch activity need to be assigned to a batch. Sadiq et al. [96] distinguish between a *user-invoked* batch assignment, creating a batch with user-selected instances, or an *auto-invoked* assignment, where the system assigns process instances to batches based on a specific mechanism. The user-invoked batch assignment is a manual approach where rules are not explicit, and benefits of batch processing might not be fully usable. However, in certain use cases, it might be required. If execution data is captured, then it is possible after a while to identify the rules for the auto-invoked mechanism with process mining techniques [58]. An auto-invoked mechanism has to consider the grouping, order of the instance queue and the resource capacity. Thereby, batch scheduling techniques as reviewed in [82] might be used. The auto-invoked mechanism can be further sub-divided in *continuous* assignment, where each arriving instance is assigned, e. g., in [69], or *resource-dependent* assignment, where the assignment is conducted as soon as the resource is free, e. g., in [53, 77].

R3 Batch Execution

R3 BATCH EXECUTION. In the batch execution phase, a batch has to be first activated by an *activation mechanism*, and then be provided to the task performer.

(R3.1) Activation Mechanism: Batch processing is used to save execution costs. Thereby, instances have to be stopped and paused in their execution in order to wait for others with which the paused instances can be executed in a batch. Pausing instances can increase their cycle time. Hence, time and cost appear to be in conflict, and the trade-off between them has to be managed. This is the responsibility of the batch activation mechanism that determines when a batch operation can be started. The resource availability is always the prerequisite for the execution of a batch such that an activation mechanism usually describes the earliest possible moment when the execution of a batch can be started. We can distinguish between (1) *user-invoked* activation (i.e., starting self- or system-created batches) and (2) *auto-invoked* activation. User-invocation has similar disadvantages as discussed for Requirement R2.5. For *auto-invoked* activation, rules or constraints have to be defined by the process designer which can be checked by the system to activate a batch. If a batch scheduling algorithm is used, the batch activation is implicitly given by scheduling a batch on the resource.

(R3.2) Batch Scheduling: After activation, enabled batches need to be queued and scheduled for execution. In some use cases, often where machines are involved, an automatic scheduling (regarding a scheduling policy) as proposed in [53, 77] is needed. In use cases where hu-

mans are mainly involved (e. g., SC-2, SC-6-8), user-initiated scheduling (i.e., the system shows a possible schedule, but the user can decide which batch to execute) might be used, as done for work items in [95]. For the latter, batch processing concepts should consider that batches are not always immediately executed by the task performer.
(R3.3) Execution Strategy: Batch processing occurs in two versions: (1) parallel and (2) sequential execution [60, 82]. Also, Martin et al. [58] distinguish between simultaneous/concurrent (i. e., parallel) and serial (i. e., sequential) batch activities. In parallel batch execution, the activity instances of a batch can be processed by the task performer simultaneously, because its capacity is greater than one. All instances of a batch are terminated together. In sequential batch execution, the task performer enacts the activity instances one after another. They are processed as batch because they share the same setup, e.g., a familiarization phase [77]. Activity instances of a batch are in different activity life cycle states during the batch execution, some might be already terminated, whereas others are still executed. In the latter case, the order of the batch might have an influence. Scheduling strategies, e.g., FCFS, EDD might be relevant here as well [115].

R4 CONTEXT. Context information reflects changing circumstances during process execution [97] and can support the design of a flexible batch processing concept. Flexibility is defined as the ability to react on changes [26]. Current work and scenarios reveal mainly requirements targeting adaptability of batch processing.

R4 Context

(R4.1) Adaptation: (1) Exception and (2) special cases can have influence and might require adaptations during (a) batch creation and (b) batch execution as it was discussed in operation management work, e. g., in [74, 107] and in the BPM domain by [135]. For example, if the blood testing machine has to be maintained, exception handling is triggered, such that all batches are finished before the start of maintenance. Special cases often require a special treatment. In case of batch processing, special cases are process instances which cannot follow the normal batch procedure. For example, if a customer has selected the fast delivery option, their order has to be sent out as soon as possible. For these cases, the option of fast handling should be provided.
(R4.2) Variability: In addition to process adaptation, three other major flexibility needs are differentiated [92]. Whereas looseness and evolution are needs targeting the modeling paradigm respectively the BPMS, variability is also relevant for batch processing in business processes. Several customer groups, several product types etc. which are handled by one process model can require different batch configuration variants, e. g., different activation mechanisms. For instance, a premium customer group can require that its members are only handled optionally as batch, whereas for all other customer groups the batch processing mechanism is always applied.

Discussion of Completeness

The requirements framework consists mainly of aspects which were elicited based on a structured state-of-art review. Due to this methodological choice, it includes mainly requirements which are already supported. Since only a limited number of contributions enabling batch processing in business processes exists, we included also use cases from practice in the requirements analysis. Thereby, we interviewed interested domain experts and scanned the forums of existing BPMS providers. Nevertheless, the interviewees have currently no solution in place with which they can explicitly represent batch processing configurations in process models. Hence, they might not be able to think about every aspect needed yet. This can lead to further requirements in the future. We think that especially the requirements targeting flexibility in *Context* group can get get more detailed in future. With the elicited requirements from literature and scenarios, a given classification of requirements was proposed which was discussed and validated with a group of BPM experts. In future, it should be further validated with BPMS providers and domain experts. We are confident that the framework covers the most important aspects, but we can not guarantee its completeness.

Application of the Requirements Framework

Next, we apply the framework to structurally compare the requirements of the given scenarios in Section 4.1 for setting the objectives of the batch activity concept introduced in the work of this thesis.

In Figure 15, all scenarios are listed with their needed configuration for each aspect of the requirements framework. For example for the blood sample test in SC-1, we can observe that the batch processing is required for a single activity and a single process, instances have to be always executed in a batch (i. e., no optionality allowed), no grouping is necessary, etc. The table indicates that use cases exist with a strong need for an automatic batch processing support, e. g., no optionality, auto-invoked batch assignment, activation, and scheduling as observed in SC-1, and SC-3-5. On the contrary, the other use cases request more user involvement during assignment, activation, or scheduling and even optionality. Therefore, we distinguish between two preliminary types: (1) automated batch activities and (2) user-involved batch activities. These are described in more detail in the following.

Automated batch activities (e. g., in SC-1, SC-3-5) are either executed by machines or by software services (e. g., the automatic triggering of a financial transaction in the return handling scenario). Batch processing can span over a single or multiple activities. Batching is here required (i. e., no optionality allowed) and the instances are executed in parallel. In the works of operations management, we could also observe that sequential batching on machines is important [82]. For full-automation,

	SC1 - Blood Sample Test	SC2 - Order process	SC-3 - Shipping Abroad	SC-4 - Return Handling	SC-5 - Manu-facturing	SC-6 - Leave Application	SC-7 - Invoice	SC-8 - Customer notifications
Process Model								
R1.1 Involved activities	single	multiple	single	single	**multiple**	single	single	single
R1.2 Involved processes	single	**single**	single	single	single	**single**	**single**	**multiple**
Batch Creation								
R2.1 Optionality	no	yes	no	no	no	yes	no	yes
R2.2 Grouping	no	**yes**	**yes**	**yes**	no	**yes**	**yes**	**yes**
R2.3 Instance scheduling	**EDD**	FIFO	FIFO	FIFO	FIFO	FIFO	FIFO	FIFO
R2.4 Resouce capacity	**limited**	unlimited	unlimited	unlimited	**limited**	unlimited	unlimited	unlimited
R2.5 Batch assignment	auto-invoked	auto-invoked	auto-invoked	auto-invoked	auto-invoked	user-invoked	auto-invoked	user-/auto-invoked
Batch Execution								
R3.1 Activation mechanism	auto-invoked based on cost and due date	user-/auto-invoked based on cost and time	auto-invoked based on cost and time	auto-invoked based on return completed	auto-invoked based on cost and time	user-invoked	auto-invoked based on time	user-/auto-invoked
R3.2 Batch scheduling	automatic	user-initiated	automatic	automatic	automatic	user-initiated	user-initiated	user-initiated
R3.3 Execution strategy	parallel	parallel	parallel	parallel	parallel	sequential	sequential	parallel
Context								
R4.1 Adoption	needed	-	-	-	-	-	-	-
R4.2 Variability	-	-	-	-	-	-	-	-

Figure 15: Comparison of the eight scenarios based on the requirements framework resulting in two types of batch activities: automated (in white) and user-involved (in grey).

instances are automatically assigned to batches, an auto-invoked activation mechanism is applied, and the batches are automatically scheduled on the resources. In case of machine-support, the capacity is usually limited.

User-involved batch activities (e. g., in SC-2, SC-6-8) can involve multiple activities and in scenario SC-8 we see an example of batching over several processes. However, this might be also possible for automated batch activities, as *process model* aspects seemed to be independent from the types. Specific for user-involved batch activities is that batching is, in most cases, optional and some user involvement might be desired, for instance, in the batch assignment (cf. SC6 and SC8), in the batch activation (cf. SC-2, SC-6 and SC-8), or in the batch scheduling (cf. SC-2, SC-6-SC8). Reasons for user-involvement might be to increase the flexibility, to handle special cases, or to fulfill legal requirements which might request that process experts have to check created batch clusters first. This means that user-involved batch activities are user tasks which can be executed more efficiently by introducing additional batching, if possible. Depending on the use case, it has to be decided to which degree the user-involvement is desired: process performers might simply get auto-generated batches on which they can decide when to work. Or, they might be also able to activate, or create batches on their own. Additionally, user-involved activities may be executed in parallel or in sequence by the users. For the latter, it is important that a decision can be made for each individual case. From a user interface perspective, two options are possible; cases could be individually represented, or a comprehensive user view can be chosen, where individual data for each case can be entered.

Currently, information on the flexibility needs is limited. A reason is that since many scenarios are currently manually executed, flexibility is always possible, and thus, practitioners do not foresee which flexibility needs they might have. This needs to be further investigated. Further, the two types of batch activities should be further validated on other use cases in future.

Based on these insights, the following section discusses the objectives for the batch processing concept presented in this thesis.

4.3 OBJECTIVES AND PRIORITIZATION OF REQUIREMENTS

Non-functional objectives

We aim at a basic batch activity concept for business processes supporting automated as well as user-involved batch activities which is useful and easy applicable by practitioners. Further, the concept should be generic, such that it is applicable to different control-flow oriented process modeling languages. Next, we elaborate on these objectives:

O1 Usability: Our goal is to provide a concept for batch activities with which practitioners can quickly specify batch processing needs in process models They should be able to identify areas in a process model where batch processing should be conducted and can easily configure them. Further, we also want to provide an execution semantics, such that the batch activity can be added to existing BPMSs for the automatic execution of the specified batch configuration.

O2 Usefulness: The process performance dimensions time, cost, and quality are in close relation to each other [26]. If one dimension should be improved, it has influence on the others. If, for example, the quality of a service should be improved, time or cost of the service will increase. It means that cost reductions as aimed by batch processing are usually associated with an increase in time With our batch processing concept, we want to provide a mean for process stakeholders that helps them to create batch activities with the respective process configuration which reduces process costs with almost no negative influence on time.

O3 Generalization: The concept introduced in the work of this thesis should be generic concept, such that it can be applied to process modeling languages which conform the process model definition introduced in Definition 2.1.

Functional objectives

After identifying the non-functional requirements, we now prioritize the functional requirements which are supported by the batch activity concept introduced in the next chapter. A summary of the requirements prioritization is shown in Figure 16. Regarding the category *R1 Process Model*, the focus is on single processes only as nearly all collected scenarios (besides scenario SC-8) are bound to one process, but batch processing over several activities is supported.

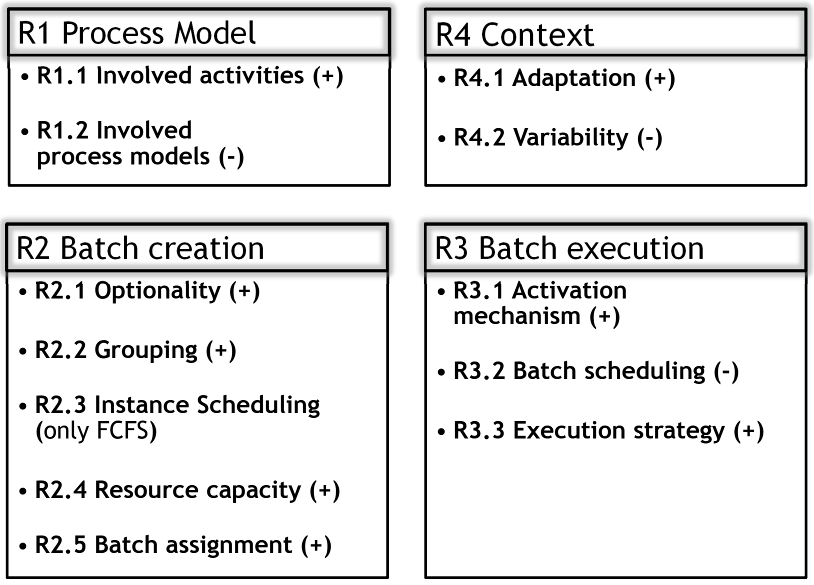

R1 Process Model
• R1.1 Involved activities (+)
• R1.2 Involved process models (-)
R4 Context
• R4.1 Adaptation (+)
• R4.2 Variability (-)
R2 Batch creation
• R2.1 Optionality (+)
• R2.2 Grouping (+)
• R2.3 Instance Scheduling (only FCFS)
• R2.4 Resource capacity (+)
• R2.5 Batch assignment (+)
R3 Batch execution
• R3.1 Activation mechanism (+)
• R3.2 Batch scheduling (-)
• R3.3 Execution strategy (+)

Figure 16: Prioritization of requirements for setting the design objectives.

In category *R2 Batch Creation*, almost all requirements are prioritized besides the *instance scheduling*. In existing works of enhancing process models with batch processing capabilities, and also in most BPMSs, FCFS is applied as scheduling policy. Hence, we also plan to use it, and do not support the selection between different policies. Restrictive batch processing is especially required if activity costs are high. Optional batch processing is needed for use cases, where the customer satisfaction is more fragile. We will consider both, such that *optionality* is supported. The necessity of *grouping* activity instances can be observed in most of the presented scenarios, as well as in most of the discussed related work of the BPM domain. *Resource capacity* defines the maximum size of a batch, and is therefore, an important factor to be considered. For batch assignment, we aim at an auto-invoked mechanism to fully use the advantages of batch processing. For supporting also user-involved batch activities, we want to strengthen the user involvement in the batch assignment by allowing users to adapt batches.

In category *R3 Batch Execution*, the most important requirement is the activation mechanism, if no scheduling technique is applied. Batch scheduling techniques are not utilized for the batch activity concept of this thesis, because many input data is requested by them, such as constraints and goal function, and the instances have to be available in advanced to define useful schedules. In contrast, we plan to apply batch activation rules proposed in queuing theory which help to balance process cost and time, such that the pausing of instances has no notable influence on the overall process cycle time. The *Batch scheduling* after the batch activation is not in focus of this work. Each BPMS usually has an allocation mechanism implemented for work items, as well as service calls. A batch is either a consolidated work item or a service call of several activity instances. Hence, the batch scheduling is outsourced to the respective component of a BPMS (cf. Figure 7). As related work discusses the importance of both, parallel and sequential *execution strategy*, we pursue the support of both strategies.

Regarding the category *R4 Context*, we focus here on the adaptation to special cases which is not discussed so far. From the collected scenarios, it could be observed that batch processing should be not applied to each process instance, e.g., emergency cases in the blood testing process, orders where the customer has selected a fast delivery. Thus, we want to consider this to allow a successful application of batch activity in practice. Further, we want to support the user involvement in the batch creation and execution, targeting flexibility in general.

After presenting the requirements framework, and using it for setting the design objectives, the next chapter introduces the batch activity concept.

5 BATCH ACTIVITY

After setting the design objectives, this chapter introduces the batch activity *for explicitly specifying the batch execution of process instances in a process model. Thereby, the grouping of process instances regarding their context data is an important aspect. Hence,* data views *are defined to identify similar instances. While presenting the design of a batch activity with its configuration parameter, details are given on the batch activation rule managing the trade-off between cost and time, and on integrating the batch activity in process modeling languages, such as BPMN. This chapter provides an execution semantics for the automatic batch activity execution. Further, user involvement strategies are discussed. The chapter concludes with a discussion of the presented concept by comparing the batch activity with related work. This chapter is based on the published papers "Batch Activities in Process Modeling and Execution" [83] and "Batch Regions: Process Instance Synchronization based on Data" [89].*

Based on the requirements analysis and the deduced design objectives of the previous chapter, this chapter introduces a new type of activity – the *batch activity*. It enables batch processing in a business process for automatic as well as user-involved batch activities which can be single activities or activities with an internal behavior. This chapter discusses on the example of BPMN [71], UML Activity Diagrams [72], and EPC [98] how to integrate the concept in a process modeling language.

With the configuration parameters of a *batch activity*, a process designer is able to specify the batch execution in a process model, e. g., when a batch is started (activation rule), how many instances are allowed at maximum in a batch (batch size), and how the batch is executed (parallel vs. sequential). Further, batch activities are able to group process instances regarding their specific instance data. This is enabled with *data views* on process instances, an abstracted view on the relevant process instance data.

Data views are formalized, and an algorithm to identify clusters of process instances with the same data view is presented in Section 5.1. Section 5.2 provides the concept of batch activities, and describes its integration into process modeling concepts. The batch activity configuration parameters are presented in general, whereby details on the grouping of instances, and on the batch activation rule is provided in this section. Also the integration into different control-flow oriented process modeling languages is discussed in this section. In Section 5.3, the corresponding execution semantics for batch activities being user tasks, service tasks, or sub-processes is presented. As observed in the previous chapter, also user involvement during batch assignment, and execution is needed in certain use cases. A proposal for the batch activity concept is discussed in Section 5.4. A comparison on the requirements framework of the introduced batch activity to other related solutions can be found in Section 5.5. The prototypical implementation of the concept is discussed in the evaluation part in Chapter 8.

5.1 PROCESS INSTANCE GROUPING WITH DATA VIEWS

In the requirements frameworks, it is shown that grouping of instances in specific batches is important for most of the use cases. This section presents a concept to group instances based on their data characteristics. As described in the preliminaries, each instance acts on multiple data objects which give the instance a context and characterize it. However, not all data is relevant to compare process instances and to identify similar ones; only specific attributes of the utilized data objects are of interest. In database systems research, the concept of views allows to extract relevant data by projection [103]. In this section, this concept of views is applied to business processes instances to identify related ones based on data – being called *data views* for process instances. In the remainder of this section, a motivation for the grouping of process instances is given, followed by the formalization of data views, and algorithms to cluster instances based on data views.

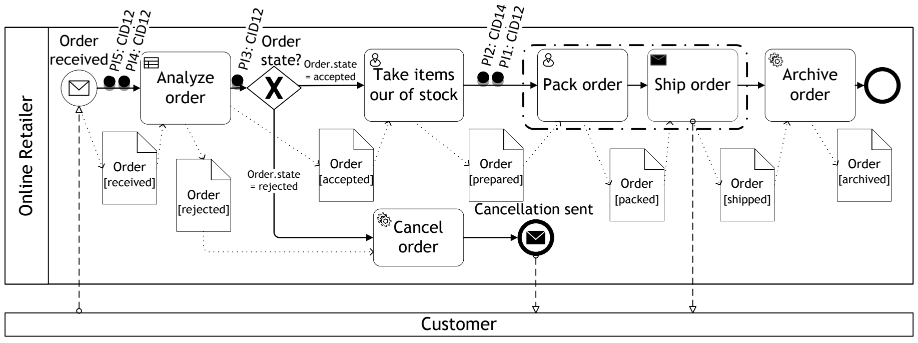


Figure 17: Online retailer process diagram with exemplary running instances represented as labeled token. The label, e.g., PI1: CID12, references the respecting process instance id and customer id.

Motivation for Grouping Process Instances

Let us return to the running example introduced in Chapter 2, given in Figure 17. In times where customers often do not have to pay any shipment costs, online retailers face the situation that customers place several orders with the same shipping address within a short time. Batch processing can be used for this *Online retailer process* to batch orders of the same customers during the packing- and sending-activity in order to save transportation costs. It has to be assured that only orders are batched which are from the same customer, and have the same shipping address. For example, by looking on the customer ids of the illustrated process instances in Figure 17, process instance *P1* can be potentially grouped with *P3*, *P4*, and *P5* acting on an order of the same customer with the id `CID12`, but not with *P2*. This handles an order of the customer with the id `CID14`.

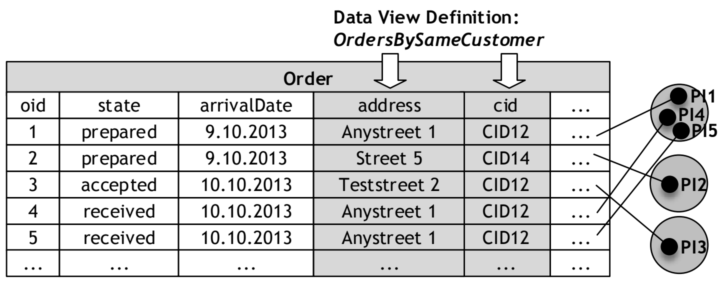
Instance similarity metric vs. data views

Similar instances can be identified, for instance, with a similarity metric as proposed by Pflug and Rinderle-Ma in [79]. This attribute similarity metric returns a similarity score for two given instances based on their attribute values. The disadvantage is that instances have to be compared pairwise, and the process engine has to request the data storage for each comparison. Therefore, we propose the concept of *data views* – an abstracted view on the process instance data – for grouping process instances.

Data View Definition

A *data view* is a projection on the values of multiple data object attributes of a process instance which might change with the progress of the process instance. Thereby, only attributes defined as relevant by the process designer specified in a *data view definition* are considered. Process instances with identical values for all defined attributes are considered *similar*. Formally, we define the data view as follows:

Definition 5.1 (Data View)**.**
Let $X = [x_1, x_2, \ldots, x_k]$ be a list of fully qualified data attributes of interest in process model m referred to as *data view definition*, where for each $x \in X$ holds that $x = c.j$, such that $c \in C_m \wedge j \in J_c$. Then, a *data view* for a process instance i is a list of values $V' = [v_1, v_2, \ldots, v_k]$. Given the relevant attribute $x_l = c.j, 1 \leqslant l \leqslant k$, the corresponding value v_l is calculated by data view function $\varphi(x, i) = \varphi(c.j, i) = \rho_o(j)$, where the data state function ρ as given in Definition 2.6 returns a value for the given attribute j, and object o is an instance of class c used in process instance i. ◀



Order					
oid	state	arrivalDate	address	cid	…
1	prepared	9.10.2013	Anystreet 1	CID12	…
2	prepared	9.10.2013	Street 5	CID14	…
3	accepted	10.10.2013	Teststreet 2	CID12	…
4	received	10.10.2013	Anystreet 1	CID12	…
5	received	10.10.2013	Anystreet 1	CID12	…
…	…	…	…	…	…

Figure 18: Example of a data view definition and the resulting data view clusters.

The data view function $\varphi(x, i)$ returns for a given relevant data attribute $x = c.j$ the value of a data object o of class c of the given process instance i. To identify the corresponding data object o from the given fully qualified data attribute x and the process instance i, the auxiliary function $\theta : I \times C \rightarrow O$ is used. $\theta(c, i) = o$ scans the set of data objects and returns one object corresponding to data class c by name matching. Then, the data view function φ uses the data state function $\rho_o(j)$ to return the value of the given data attribute j of the object o. The data view function is executed once for each relevant data attribute $x = c.j$ to construct the data view V' for one process instance i.

Let us illustrate this on the running example. In the online retailer process, it is aimed to group, i. e., distinguish, the process instances with respect to the customer address and the customer identifier. Thus, the list of fully qualified relevant attributes X comprises x_1 = `Order.address` and x_2 = `Order.cid`; we reference this data view definition as *OrdersBySameCustomer* (cf. Figure 18). Both attributes refer to data class *Order*. Considering process instance *PI1*, after execution of activity *Take order out of stock*, the current state of the corresponding *Order* data object is ρ_{Order}`(state) = prepared`. Based thereon, the data state function returns the following values for the given attributes: ρ_{Order}`(address) = Anystreet 1` and ρ_{Order}`(cid) = CID12`. Thus, the resulting data view for *PI1* is `[Anystreet 1, CID12]`. Referring to Figure 18, it is highlighted together with the data views for the other process instances in the presented data object table.

Data views are not only simple projection on one single data class. In a data view definition, attributes of multiple data classes being used by a process model can be used. It enables to consider the complete data characteristic of a process instance for the grouping.

Data view cluster

Instances of one process model can now be grouped based on their data view by assigning each instance to one *data view cluster*. Process instances with identical data views are collected in the same cluster. A data view cluster is defined as follows.

Definition 5.2 (Data View Cluster)**.**
Let I be a set of process instances of one process model m. A *data view cluster* q is a set of process instances $I' \subseteq I$ of m, where all process instances $i \in I'$ share the same data view V'. ◂

The set of all data view clusters of one process model is denoted as Q. As illustrated in Figure 18, grouping the process instances *PI1* to *PI5* based on data view definition *OrdersBySameCustomer*, process instances *PI1*, *PI4*, and *PI5* belong to one cluster while *PI2* and *PI3* belong to two separate clusters because either the customer identifier (*PI2*) or the customer address (*PI3*) differs.

Based on these formalisms, we now present and explain in the next subsection the algorithms to first calculate the data view for a process instance in a specific state and to second assign a process instance to the corresponding data view cluster based on its data view.

Clustering Algorithms

Algorithm 1 describes the implementation of the data view function introduced in Definition 5.1.

Algorithm 1 Data view creation.

```
Input: $X, i$
Output: $V'$
1: $V' \leftarrow$ new List();
2: for all $x \in X$ do
3:     $v \leftarrow \varphi(x, i)$;
4:     $V'.add(v)$;
5: end for
```

As discussed, the data view function requires a list X of relevant and fully classified data class attributes – which may come from different data classes – and a process instance i to compute its data view. First, V', the variable to hold the data view, i. e., the list of values for the given attributes X, is initialized as an empty list (line 1). Line 2 to 5 iterate over all specified attributes $x \in X$. For each attribute x, function $\varphi(x, i)$ is applied. It returns the current value v of x of the corresponding data object of process instance i. This value is added to the list V' (line 4). The result of the algorithm is the list V' representing the data view for the process instance i.

Algorithm 2 Data view cluster assignment.

```
Input: $X, i, Q$
Output: $Q$
1: $foundCluster \leftarrow$ false;
2: $V'_i \leftarrow dataView(X, i)$;
3: for all $q \in Q$ do
4:     $V'_q \leftarrow dataView(X, \gamma(q))$;
5:     if $V'_i = V'_q$ then
6:         $foundCluster \leftarrow$ true;
7:         $q \leftarrow q \cup \{i\}$;
8:         break;
9:     end if
10: end for
11: if $foundCluster$ = false then
12:     $p \leftarrow \{i\}$;
13:     $Q \leftarrow Q \cup \{p\}$;
14: end if
```

Algorithm 2 assigns a process instance i to a corresponding data view cluster by using Algorithm 1. As input, it requires the list X of data class attributes, the corresponding process instance i, and the set Q of currently existing data view clusters for the respecting process model;

this set may also be empty. The output of the algorithm is the updated set Q of data view clusters with either an additional cluster $q \in Q$ with the given instance i or the same number of clusters with one of them now containing instance i. For computation, first, a Boolean variable $\mathtt{foundCluster}$ indicating whether a matching cluster is found gets initialized with value *false* (line 1). Then, the data view for the process instance i is calculated using Algorithm 1 and is saved in list V'_i. Lines 3 to 10 iterate over all input clusters Q. First, for a cluster q, the data view is calculated. Therefore, an instance assigned to it is taken with the auxiliary function $\gamma : Q \to I$, which returns a process instance j from a given data view cluster $q \in Q$, and this instance is provided as input to Algorithm 1. The result is assigned to list V'_q. Afterwards, the data view V'_q of cluster q is compared to data view V'_i of the given process instance i. If they are equal, variable $\mathtt{foundCluster}$ is set to *true* (line 6), the process instance is added to the corresponding cluster q (line 7), and the iteration is aborted (line 8). If no corresponding cluster is found during the iteration, the process instance i is added to a new data view cluster p (line 12). This new cluster in turn is added to the set Q of data view clusters (line 13).

By applying the second algorithm to all instances of one process model, they can be grouped into data view clusters based on the data information they carry at that point in time. The algorithm can also be applied on a subset of instances. Referring to the batch activity being discussed in the next section, this subset may comprise all process instances for which the batch activity is enabled.

5.2 MODELING

In the following section, the modeling concept of batch activities and their configuration parameters is presented. One of the configuration parameters, the batch activation rule which is responsible for balancing cost and waiting time, is introduced in detail in the next subsection. Further, an application of the concept to process modeling languages is shown at the end of this section.

Batch Model and its Configuration Parameters

For introducing batch activities, the process model definition in Definition 2.1 has to be extended. A process model consists of nodes and edges, and acts as a blueprint for a set of process instances which are related to exactly one process model. A node in a process model can represent an event, a gateway, or an activity. Each activity is associated with an arbitrary number of activity instances which are in a certain life cycle state (see Figure 19). An activity can be either a single task (i. e., a user, a service, or unspecified task), or an activity with internal behavior.

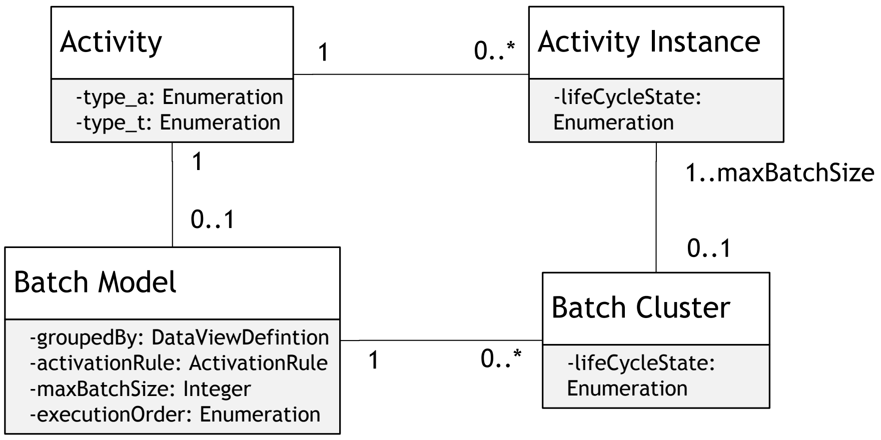


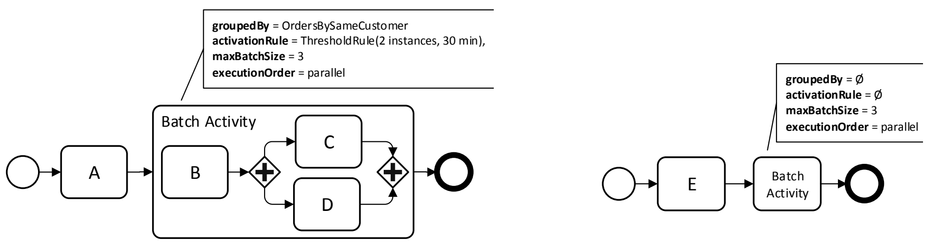
Figure 19: Conceptual model for batch activities represented as UML class diagram [72].

These concepts are extended by the batch model shown in the class diagram of Figure 19; an activity becomes a batch activity, if it is associated with a batch model which in turn can only be related to exactly one activity. The internal behavior of a batch activity can consist of activities, events, and gateways with the limitation that conditions on exclusive gateways (type = xor) must be designed, such that all process instances of one batch follow the same path. A batch model describes the conditions for the batch execution, and can be configured by the process designer. Next, we describe the four given configuration parameters:

- The *groupedBy* is from type *DataViewDefinition* and defines how the process instances are grouped by specifying the relevant attributes for identifying similar ones. We assume that process instances do not change their data view during the execution of a batch activity, i.e., no task within a batch activity updates an attribute specified in the data view definition. If no data view definition is provided, process instances are grouped upon their arrival order.

- The *activationRule* provides the possibility to specify when a batch of instances is enabled. The process designer selects an activation rule type and provides the required user inputs. In case that no activation rule is given, batch processing is optional. That means that instances do not wait explicitly for others, but in case matching partners exist, it is grouped with them. The details on activation rules are provided later in this section.

- The *maxBatchSize* is of type integer and limits the batch capacity by specifying the maximum number of instances processed in a batch. It can be used to incorporate limits of involved resources.


groupedBy = OrdersBySameCustomer
activationRule = ThresholdRule(2 instances, 30 min),
maxBatchSize = 3
executionOrder = parallel
Batch Activity
A
B
C
D
groupedBy = ∅
activationRule = ∅
maxBatchSize = 3
executionOrder = parallel
E
Batch Activity

(a) Example Process 1. (b) Example Process 2.

Figure 20: Exemplary configurations of batch activities in two abstract processes.

- The *executionOrder* is of type enumeration and describes whether instances of a batch are executed in *parallel* or *sequential*. Parallel execution means that all instances of one task are executed simultaneously, and are terminated before the next task is executed the same way. In case of sequential execution, instances of a batch are executed one after another, but it needs only one initial setup or familiarization phase. For batch activities being a sub-process, sequential execution can be *task-based*, the sequential variant of parallel execution, or *case-based* where all nodes within a batch activity are executed for one process instance (case) before the next instance can be started.

Each batch activity has an arbitrary number of batch clusters. A cluster's behavior (instance level) is defined by the batch model's configuration. A batch cluster is responsible for the batch execution and comprises a positive natural number of activity instances with *maxBatchSize* specifying the upper limit. The instances of a batch are not merged into a single one to retain their autonomy outside of the batch activity and to allow exception handling of single instances.

Batch activity examples

Figure 20 shows exemplary batch activity configurations for two abstract processes. The batch activity in Figure 20a is a sub-process with three activities *B*, *C*, and *D*. For the *groupedBy* parameter, the process designer selected the data view definition *OrdersBySameCustomer* introduced in the previous section; thus only instances which have the same *cid* and *address* are grouped in a cluster. As activation rule, a *ThresholdRule* is chosen which activates a batch cluster either if two instances are available, or a maximum waiting time of 30 minutes is reached. The maximum batch size is three; as the maximum batch size is higher than the maximal threshold, further instances can be added to the batch cluster although it was already activated as long as the processing is not started by the task performer. Finally, the *executionOrder* defines that all instances are processed in parallel. *Process 2* presented in Figure 20b illustrates an example where batch processing is conducted for one activity with an empty *groupedBy* parameter. This means instances are assigned to clusters based on their arrival time. If a batch cluster has reached its maximum batch size, a new cluster is created as soon as a new instance arrives. Further, no activation rule is given; optional batch processing is selected. Thus, batch clusters are immediately activated, and provided to the task performer/service similar to the simple activity execution. If further instances arrive, and the cluster was not executed yet, then instances can still be added to it.

In the next subsection, we provide details on the activation rule.

Batch Activation Rule

As described, the process designer selects an activation rule type for a batch model, and configures it with the required user inputs. Specifying the activation rule requires to find an optimal trade-off between reduced execution cost and increased waiting time. The optimal configuration settings are derived from expert knowledge, simulations, or statistical evaluations.

We assume that business process management system (BPMS) suppliers provide different types of activation rules in advance. In this section, the general concept of batch activation rules, and four different types of rules are introduced, two versions of the *threshold rule* and two extensions of it. The *threshold rule* is an often considered rule in queuing theory which states that a batch is started, when the length of the waiting queue with customers is equal or greater than a given threshold (i.e., a value between one and the maximum server capacity) and the server is free [70]. This rule can be extended by a maximum waiting time, such that a group smaller than the set threshold is also served. Further, in this section, an extension of the threshold rule is discussed to consider also instances arriving in future, and an activation rule considering a fast track option is shown.

In general, an activation rule relies on the concept of Event-Condition-Action (ECA) rules. Originally, ECA rules were used in active database systems [19]. Basic elements of an ECA rule are an event E triggering the rule, a condition C which has to be satisfied, and an action A being executed in case of fulfillment of the condition [51].

Thus, we define an activation rule as a tuple $E \times C \times A$:

Definition of the batch activation rule

- An event E is either an atomic event or a composite event being a composition of atomic events through logical operators, as for instance *AND* or *OR*. Differently to the preliminaries, here we focus on internal system events of the process engine, such as a change of the batch cluster size, or a specific time event.

- A condition C is a boolean function. The input elements to such a function can be system parameters (e.g., *actual length of waiting queue*), user inputs (e.g., THRESHOLD), or a combination of both (e.g., total service costs = (VARIABLE COSTS $*$ *actual length of waiting queue*) + CONSTANT COSTS) connected by a relational expression. The composition of several atomic conditions with logical operators is called composite condition.

- The action A is always the enablement of the associated batch cluster.

An example for the batch activation rule, the *threshold rule*, is given below.

Threshold rule

```
1 ActivationRule ThresholdRule
    On Event
      (bc.size_increased) OR (timer_event)
    If Condition
      (bc.size ⩾ THRESHOLD )OR
6     (bc.lifetime ⩾ MAXWAITINGTIME)
    Do Action
      Enable bc
  End ActivationRule
```

Listing 1: Threshold rule to exemplify the batch activation rule.

In this activation rule in Listing 1, the user inputs are indicated by capitals. It consists of a composite event saying that the rule is triggered when a new activity instance is added to the associated batch cluster *bc* or when no new one was added for a specific period, such that a timer event is fired. With triggering the rule, the given composite condition is checked. It states that either the batch cluster size has to be equal or greater than the user-specified threshold, or the lifetime of *bc* has to be equal or greater than the user-specified maximum waiting time. If the condition evaluates to true, *bc* gets enabled (i. e., activated).

Threshold rule with due date

Next, we want to discuss an alternative variant of the threshold rule. In some scenarios (e. g., in SC-1), we could observe that processed data artifacts can have a due date until when they need to be handled, e. g., blood samples have an expiration time.

```
1 ActivationRule ThresholdRule
    On Event
      (bc.size_increased) OR (timer_event)
    If Condition
      (bc.size ⩾ THRESHOLD )OR
6     (system.time ⩾ (bc.dueDate − BUFFER))
    Do Action
      Enable bc
  End ActivationRule
```

Listing 2: Threshold rule with due date.

The due date could be considered in the threshold rule by adapting line 6 as done in Listing 2. This line compares instead of maximum waiting time, the current system time with *bc.dueDate* being the due date of the batch cluster from which a BUFFER (i. e., a given time duration by the user) is subtracted. The due date of a batch cluster is defined by the earliest due date of its containing activity instances. The buffer is subtracted from the due date such that the batch can be still processed before its due date is reached.

The threshold rule in its current design has the disadvantage that it only considers instances which are already contained in the batch cluster. Instances which might arrive in future are not taken into account. Therefore, we want to extend the current threshold rule, such that running instances which still have to pass the batch activity are considered as well. This information may be integrated into the event or condition definition of an activation rule. In Listing 3, we provide the so-called *MinMax* activation rule which considers the existence of future instances in its condition.

```
ActivationRule MinMaxRule
  On Event
    (bc.size_increased) OR (timer_event)
  If Condition
    ((Minimum condition) AND !(SimilarPI())) OR
    (Maximum condition)
  Do Action
    Enable bc
End ActivationRule
```

Listing 3: MinMax rule as extension to consider future instances.

MinMaxRule

This rule activates a batch cluster, if a minimal number of instances are assigned to it, and no other similar instance exist which can arrive at the activity in a future state (i. e., batch activity instance of the process instance is still in state *init*). Otherwise, if at least one process instance with the same data view can be observed, the activation is postponed until the maximum condition is fulfilled. Similar to the *ThresholdRule*, the *MinMaxRule* triggers the check of the condition, if the size of the batch cluster *bc* is increased, or when no new one was added for a specific period, such that a timer event is fired. The condition is a logical expression requiring that either the minimum condition is true while there exists no other instance with the same data view, or the maximum condition is true to trigger the action. The existence of other process instances is checked with function `SimilarPI()`. The function first creates the data views for all running process instances whose batch activity instance is still in state *init*, and then assigns each one to a data view cluster. If there exists a data view cluster with the same data view as the batch cluster *bc*, the function returns *true*. The configuration of the maximum and minimum conditions is similar to the threshold rule, but is up to the process designer. We propose to include timing constraints in order to avoid deadlocks.

FastTrackRule

The general case discussed so far assume that the execution of a single activity instance can be postponed in favor of optimizing the overall business process. However, some cases may require an immediate execution of an activity instance. For instance, if the blood sample analysis is part of an emergency case, the analysis must be started as soon as the blood sample is received. However, this does not mean that already waiting blood samples which fit the same batch cluster as the emergency case are ignored. Instead, the analysis of all blood samples of this cluster should be started immediately. The fast track option can be included in the activation rule. An example of such a rule is shown in Listing 4.

```
ActivationRule FastTrackRule
  On Event
    (bc.size_increased) OR (timer_event)
  If Condition
    ((dataView(ATTRIBUTES, i_ai) = VALUES) OR (...)
  Do Action
    Enable bc
End ActivationRule
```

Listing 4: Activation rule with fast track option.

For the given activation rule, the process designer provides a data view definition Attributes and defines the expected values as Values. This defines the data characteristic of instances for activating the fast track option. If the activation rule is activated by one of the specified events, then the data view function is called with the given data view definition Attributes and the process instance i_{ai} being parent of the last assigned activity instance ai. The resulting data view is compared to the Values. In case both values are overlapping, the batch cluster is enabled. For instance, if Attributes = $\mathsf{Order.priority}$ and Values = high are given for our retailer example, then process instances processing a 'high priority'-order would enable batch clusters. The activation rule can have further conditions.

The different presented rules show that the activation rule which is based on ECA rules is a flexible concept based on which different types of activation rules can be provided.

Application to Process Modeling Languages

In the remainder of this section, it is described how the batch activity concept can be integrated into a process modeling language on the example of the BPMN standard [71] which dominates the process model standard space. Additionally, it is discuss how it can be added to other control-flow oriented process modeling languages as UML Activity Diagrams and EPCs.

Integration into BPMN

In section Section 4.1, it is shown that with the existing BPMN modeling elements batch processing can not be precisely specified. In the BPMN specification, the `Activity` class being the abstract super class for all concrete `Activity` types in BPMN has currently no possibility to define batch activities. Therefore, additional properties have to be added to it.

As described in the preliminaries in Section 2.2, the BPMN specification [71] supports explicitly to add new attributes and properties to existing constructs by so-called `extensionElements`. The data class diagram in Figure 21 shows a small extract of the `Activity` class from the BPMN specification. Each BPMN activity can contain a set of modeler-defined `Properties`. For extending the BPMN specification, a `BatchActivity`-class is added which inherits from the `Property`-class, and consists of the previously introduced configuration parameters.

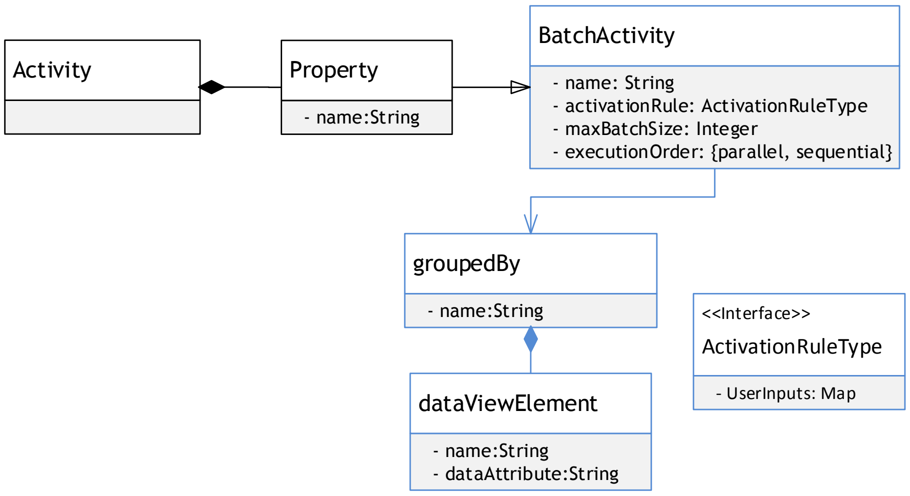


Figure 21: Batch activity configuration integrated into BPMN specification: small excerpt from the BPMN specification [71] extended by the *batch activity* class (shown in blue lines).

The attributes of the `BatchActivity`-class are an `activationRule` which references an `ActivationRuleType`-interface. This interface can be used by different types of activation rules which can be specified (e. g., a *ThresholdRule*, a *FastTrackRule* etc.). Further, the `maxBatchSize`-attribute being from type *Integer* can be used to define the maximum number of instances being allowed in a batch, and the `executionOrder`-attribute can be used to either select a *parallel* or *sequential* batch execution. Additionally, a `groupedBy`-class is associated to the `BatchActivity`-class for specifying `dataViewElements`, each of them referencing a data attribute.

After introducing the general concept of extending BPMN by batch activities, we also want to show how the batch activity configuration can be integrated in the interchangeable BPMN XML specification. In Listing 5, a proposal is shown on the exemplified sub-process given in Figure 20a. The `extensionElements` of the sub-process include a batch activity element Several data view elements can be added to the *groupedBy*-parameters (cf. line 5-6 in Listing 5). For the activation rule, a rule with its specific inputs has to be selected. The inputs are specified as properties in the selected activation rule (cf. line 9 in Listing 5). The maxBatchSize and the executionOrder are simply defined as elements. The presented `extensionElements` can be also added to BPMN tasks, such as a user task, service task etc.

```
<bpmn:subProcess id="Task_0z95iaa" name="Batch Activity">
 <bpmn:extensionElements>
  <batch:batchActivity>
   <batch:groupedBy>
    <batch:dataViewElement>Order.cid</batch:dataViewElement>
    <batch:dataViewElement>Order.address</batch:
        dataViewElement>
   </batch:groupedBy>
   <batch:activationRule>
    <batch:thresholdRule threshold="2" timeout="PT30M"/>
   </batch:activationRule>
   <batch:maxBatchSize>3</batch:maxBatchSize>
   <batch:executionOrder>parallel</batch:executionOrder>
  </batch:batchActivity>
 </bpmn:extensionElements>
 <...>
</bpmn:subProcess>
```

Listing 5: Batch activity representation in the BPMN XML shown on an example sub-process.

Integration into UML and EPCs

In a similar way, also UML Activity Diagrams [72] can be extended. The UML standard also provides an extension mechanism; in a so-called *Profile*, meta-classes can be extended to adapt them for different purposes [72]. The batch activity can be integrated by designing a profile in which a `Batch Activity` would be defined as new stereotype (i. e., "defines an extension for one or more meta-classes, and enables the use of specific terminology or notation in place of, or in addition to, the ones used for the extended meta-class"'[72]). meta-classes" [72]) This `Batch Activity`-stereotype would then inherit the attributes and restrictions from the `Activity` meta-class, and include the batch configuration parameters as discussed above. EPCs [98] are another often used process modeling language, however their focus is rather related to a semi-formal process documentation than a formal process specification [130]. EPCs do not provide an explicit extension mechanism. The modeling language of EPCs consists of events (shown as hexagon), functions (shown as rectangles), and connectors (shown as

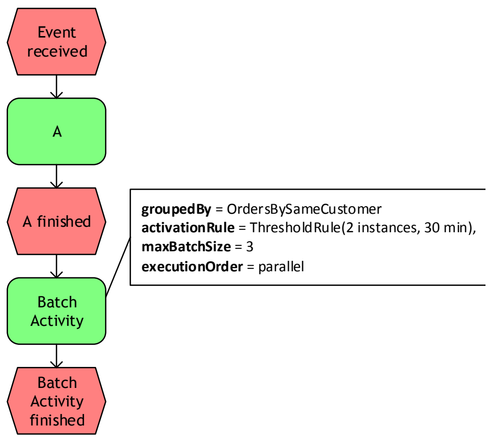

Event received
A
A finished
groupedBy = OrdersBySameCustomer
activationRule = ThresholdRule(2 instances, 30 min),
maxBatchSize = 3
executionOrder = parallel
Batch Activity
Batch Activity finished

Figure 22: Integration of the Batch Activity in EPC shown on the exemplified process given in Figure 20a

circles) which are connected via control flow. As functions represent the unit of works, those can be used and extended visually by the batch configuration parameters as illustrated in Figure 22. This does not allow an automatic batch execution, but it can be used as a starting point for discussion with other process stakeholders.

Summarized, we applied the introduced batch activity concept to BPMN, and also discussed how an application would look like for UML Activity Diagrams and EPCs. Thereby, the feasibility of the concept could be shown. Currently, we focused on control-flow orientated process modeling languages; flexible process modeling languages, such as Case Management Model and Notation (CMMN) [73] and declarative workflow modeling [119] are not discussed so far. The listed languages also have the notion of activities and activity instances, such that an integration seems to be possible which need to be further analyzed.

5.3 EXECUTION SEMANTICS

This section discusses the execution semantics of a batch activity. Thereby, the execution semantics for a batch activity being a user task, a service task or a sub-process is presented. First the execution semantics is explained on an example. Then, the lifecycle of a batch cluster is described, followed by the execution details for batch activities with an internal behavior.

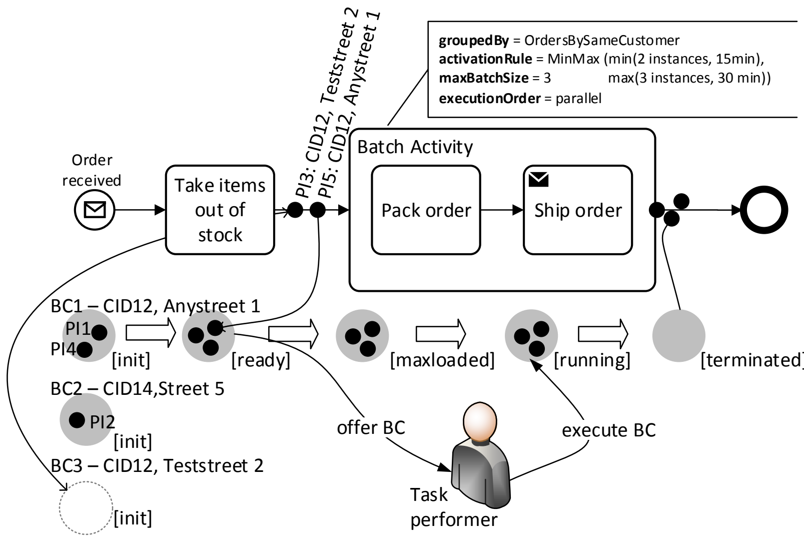


Figure 23: Execution semantics of the batch activity exemplified on a condensed version of the online retailer process.

Exemplified execution semantics

Figure 23 illustrates the online retailer example in a condensed version. In this example, batch processing is used to pack orders of the same customer in one parcel and send them together in order to save shipment costs. Hence, the batch activity consists of two activities. For the *groupedBy*-parameter, the process designer selected the data view definition *OrdersBySameCustomer* as introduced in Section 5.1. Thus, the process instances are grouped based on the customer ID and the shipping address of their processed *Order* object. Furthermore, the *MinMaxRule* is selected with two instances or 15 minutes for the minimum condition, and three instances or 30 minutes for the maximum condition. The capacity (*maxBatchSize*) of a batch cluster is set to three.

If a process instance, e. g., *PI5*, reaches the batch activity, its execution is interrupted by transferring the activity instance into the *disable* state; a disabled activity instance is temporarily deactivated (cf. Section 2.4). Then, the batch activity configuration is evaluated. If the *groupedBy*-parameter is specified, the data view of the process instance is created and it is added to the corresponding batch cluster. In the given example, *OrderingBySameCustomer* is specified as *groupedBy*-parameter leading to the data view `CID12, Anystreet 1` for process instance *PI5*. It is assigned to the batch cluster *BC1* sharing the same data view. If a process instance does not match to any, such as the second arrived instance *PI3*, a new batch cluster is created, and initialized, here *BC3* with `CID12, Teststreet 2` as data view.

Batch cluster life cycle

For batch clusters, we define a life cycle similar to the activity instance life cycle (cf. Section 2.4) to describe the steps of the batch cluster's execution.

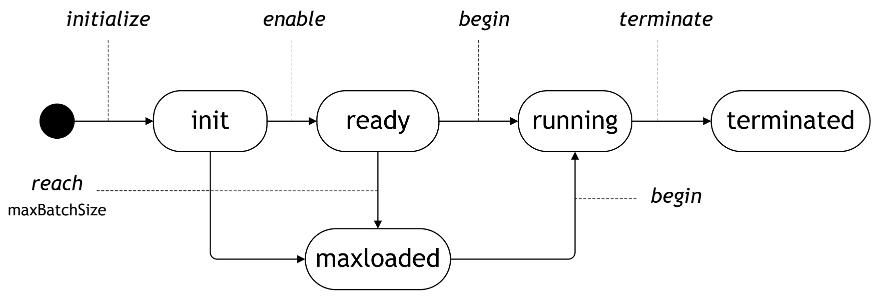


Figure 24: Life cycle state transitions of a batch cluster.

Figure 24 summarizes the states of a batch cluster, and the transitions between them. In the following, these are described in detail, thereby, we start with the explanation for a batch activity being a single task and extend this to batch sub-processes afterwards:

- **init** - Upon arrival of an activity instance requiring a new batch cluster, it gets initialized by transitioning to state *init*, where arriving instances sharing the same data view can now be assigned to the cluster. In the *init* state, the activation rule is checked, e. g., in case of instance addition, or after a certain time duration.

- **ready** - As soon as the predefined activation rule is fulfilled, the batch cluster transfers into the *ready* state. In case no activation rule was defined, the batch cluster transitions immediately from *init* to *ready*. If the batch activity is a user task, the batch cluster is offered as work item to the *task performer*. In case of a service task, the batch cluster aggregates the input to the service for its containing activity instances. It then calls the service with the aggregated input. Newly arriving activity instances can still be added to a batch cluster in state *ready*. Then, the service request is repeated with the new aggregated input.

- **maxloaded** - If a cluster has reached the *maxBatchSize*, the cluster transfers into the *maxloaded* state. In this state, no instance can be added to the cluster anymore. A batch cluster can also change from *init* to *maxloaded*, if already in *init* the maximum capacity of a cluster is achieved. This state transition ensures flexibility for the activation rule definition, because certain rules might not focus on the number of instances, e.g., start at 4pm. Also in this case, the batch cluster is provided to the respective software service or to the task performer.

- **running** - Once the batch cluster is allocated to a service or task performer, they may start with its execution. Then, the batch cluster transitions into the *running* state. Also in this state, no instance can be added anymore. The instances of a cluster can be either executed in parallel, or in sequence depending on what the process designer has defined as *executionOrder*. In case of user task, a *batch work item* is generated for the parallel execution which aggregates the work items of all activity instances of a cluster allowing a joint visualization and execution in one step.

- **terminated** - As soon as the batch work is completed, the cluster changes into the *terminated* state. Then, the resulting data of the *batch work item* or the service call is distributed to each individual activity instance which are terminated, too. In case of sequential execution, the batch cluster terminates as soon as the last instances of the batch was processed. With termination of the cluster, the process instances continue their execution individually for the control flow nodes after the batch activity.

Assignment of instances to batch clusters

After having introduced the life cycle of a batch cluster, next, an algorithm is presented in Algorithm 3 (as an extension of Algorithm 2) for assigning activity instances of a batch activity to batch clusters:

Algorithm 3 takes as input an activity instance ai, the currently available batch clusters Q, and the data view definition X referenced in the *groupedBy*-parameter. First, a boolean variable `foundCluster` is initialized with `false`. Then, if Q is not empty, the data view of each cluster $q \in Q$ is compared to the data view of the process instance i_{ai}, parent of

Algorithm 3 Algorithm for adding an activity instance ai to a batch cluster

```
Input: $ai$, Q, X
  $\mathsf{foundCluster} \leftarrow \mathsf{false}$;
  if Q not empty then
    for each $q \in Q$ do
      if $\mathsf{dataView}(X, i_{ai}) == \mathsf{dataView}(X, \gamma(q))$ then
        if $\mathsf{lifeCycleState}(q) = \mathrm{INIT} \,||\, \mathrm{READY}$ then
          $q \leftarrow \{ai\}$;
          $\mathsf{foundCluster} \leftarrow \mathrm{true}$;
          break;
        end if
      end if
    end for
  end if
  if $\mathsf{foundCluster} == \mathsf{false}$ then
    initialize batch cluster c;
    $c \leftarrow \{ai\}$;
    $Q \leftarrow \{c\}$
  end if
```

the activity instance ai, with the help of the data view function. If they are equal, the state of this cluster has to be checked, because instances can be only added, if the cluster is in state *init* or *ready*. If this is the case, the instance ai is added to the currently selected q, and the loop is interrupted. If no existing cluster could be identified for adding ai, cluster c is initialized, and instance ai is added to c. Finally, c is added to the set of existing clusters Q.

Let us come back to the presented example in Figure 23. With process instances *PI1*, *PI2*, and *PI4* having already arrived at the batch activity, the batch clusters *BC1* and *BC2* are in state *init*. Checking the activation rule for cluster *BC1* reveals that the minimum rule is satisfied but evaluating function `SimilarPI()` shows that there is another instance running with the same data view – *PI5* The maximum condition with three arrived instances is not yet satisfied; thus, the activation rule is not yet fulfilled. The arrival of *PI5* adds this instance to cluster *BC1*. The batch cluster changes into state *ready*, because the maximum condition is now satisfied.

Batch work item

In the *ready* state, the batch cluster is offered – in case of a user task – to the *task performer* of the first task in the batch sub-process. In this example batch configuration, parallel execution was selected; thus the instances of the batch cluster are provided as *batch work item* to the task performer. In case of sequential execution, only the work item of the first instance is given. We propose that a batch is assigned to the same employee for all user tasks within the batch sub-process (i.e., case handling resource pattern [95]) to ensure that the batch is performed uninterruptedly. However, other resource allocation patterns can also be applied.

As the arrival of *PI5* also satisfies the *maxBatchSize* of three, the state is transitioned to *maxloaded*, and no further instance addition is allowed. With termination of the first user task *Pack order*, the results of the activity are distributed to batch cluster's instances, and the activity instances are terminated. However, the batch cluster is not terminated, because the batch activity is in this example a sub-process. The batch cluster creates also for the *Send order*-activity a batch work item for the task performer. Details on the execution of a running batch cluster over several tasks are given in the next subsection. The batch cluster's state changes to *terminated* as soon as all control flow node instances of the batch sub-process instances assigned to the cluster have terminated.

Execution details for batch sub-processes

In this subsection, more execution details starting from the enablement of a batch cluster are given for the case that the batch activity is a sub-process. Additional details on the execution of batch activities being a user task as well as a service task are also provided.

With the configuration parameter *executionOrder*, the process designer selects the type of batch execution. They may choose *parallel* as in the previous example, *sequential per activity*, or *sequential per case*. Their different execution behaviors are illustrated as simplified UML Sequence Diagrams in Figure 25, Figure 26, and Figure 27 using an example where two process instances are synchronized within one batch cluster.

PARALLEL EXECUTION. In *parallel* execution, if a batch cluster is *enabled* (or it is *maxloaded*), then the batch cluster re-enables its assigned instances. This is shown in Figure 25 where the *disabled* instances *PI1.AI1* and *PI2.AI1* of the first activity in the batch sub-process are re-enabled, one of each assigned process instance *PI1* and *PI2*.

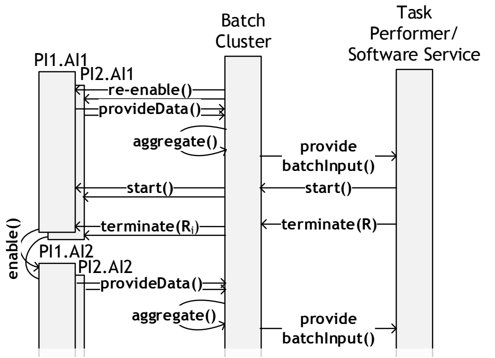


Figure 25: *Parallel* execution of two instances.

With enablement, an activity instance would usually execute directly its activity behavior (i. e., in the case of a user task creating a work item for the task performer, or in case of a service task doing a service call). Here, the batch cluster acts as interface between the activity instances and the task performer (or the service) to organize the batch execution. Thus, each activity instance provides its input data to the batch cluster. The batch cluster aggregates this input data of all instances and provides it as batch input to the task performer (or software service).

To detail this, in case of a user task, the batch cluster aggregates the data input of the activity instances into one *batch work item*, and

Batching of user tasks vs. service tasks

provides it to the task performer for parallel execution. In case of a service task, the data input is aggregated into a format that the service can handle. Here, two cases can be differentiate:

- If the software service can handle *only one input* and batching is used to get the results for several similar requests, then the service is only called with one of the data inputs which should be the same for all activity instances in a cluster.
- If the service can handle *multiple inputs*, and batching is used to reduce the number of calls of a service, then an aggregated input for the service is generated.

As soon as the task performer starts the batch work item or the service starts working on the requests, all activity instances are transitioned into the *running* state by the batch cluster. With termination of the batch work item (or the service), the combined result R is sent to the batch cluster. The cluster, then, provides each activity instance with the individual outputs R_i of the task execution. The activity instance is terminated which results in activation of its outgoing sequence flow leading to the enablement of the subsequent activity instances – *PI1.AI2* and *PI2.AI2* in Figure 25. These instances again provide their data inputs to the batch cluster and the above described steps are repeated until all control flow node instances of the batch activity are terminated. Then, also the batch cluster terminates.

SEQUENTIAL-PER-CASE EXECUTION. Similarly to the *parallel* execution, in the *sequential per activity* execution, shown in Figure 26, the batch cluster re-enables all *disabled* instances of the first node. Again, all data inputs of the activity instances are provided to the batch cluster. This time, they are arranged in a list specifying the order in which the batch cluster provides the work items one after another to the task performer (or in which order the service is called). In Figure 26, first, the work item (or the service input) of activity instance *PI1.AI1* is provided. With its termination, the first activity instance of the next process instance is provided – *PI2.AI1*. When all instances of the first activity are terminated, and the data inputs of the subsequent activity are provided to the batch cluster, a new list is generated specifying the order in which the instances of the second activity are processed. This continues for all activities in the batch activity.

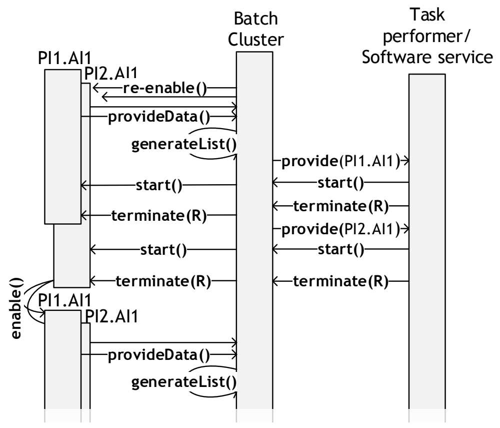


Figure 26: *Sequential per activity* execution.

SEQUENTIAL-PER-ACTIVITY EXECUTION. In the *sequential per case* execution, shown in Figure 27, all nodes in the batch activity are executed for the first process instance assigned to the batch cluster, before the nodes of the second process instance can be started. Thus, only the *disabled* activity instance *PI1.AI1* of the first process instance *PI1* is re-enabled by the batch cluster. Then, the batch cluster provides the work item/service input of *PI1.AI1* to the task performer/software service. If it is finished, the work item of the subsequently enabled activity instance *PI1.AI2* of the same process instance is provided. When all nodes of the first process instance *PI1* are terminated, the *disabled* activity instance *PI2.AI1* of the second process instance *PI2* is enabled by the batch cluster, and all activity instances of this process instance are processed as described above for the first process instance. The batch cluster terminates, if all assigned process instances are processed.

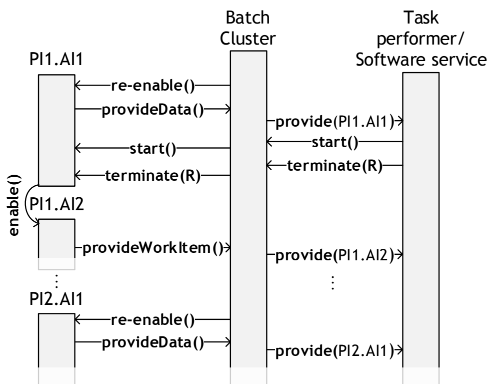


Figure 27: *Sequential per case* execution.

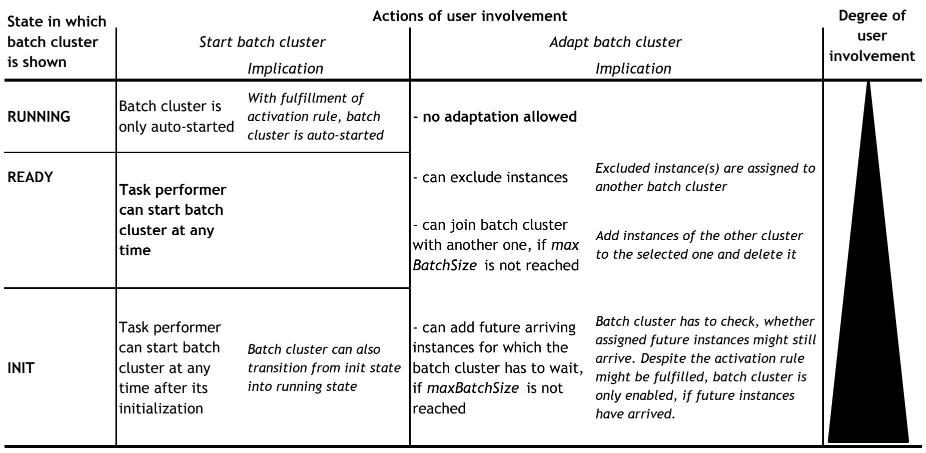
5.4 USER INVOLVEMENT

The previous chapter – the requirements analysis – revealed that user involvement is desired in certain use cases during batch assignment, during the batch activation, or in the batch scheduling. In this section, we want to discuss where and how user involvement can be incorporated in the presented concept.

Currently, task performers get presented, in case of a user task, the batch cluster in the user interface, when it is in state *ready*. Then, they can start it at any time. In case of a service activity, the batch cluster is not visible. In the following, we mainly discuss involvement strategies explicitly for user tasks; however, they can be also applied to service tasks. In case of service tasks, it has to be additionally defined which resource is responsible to monitor the batch clusters, e. g., the process owner.

Different options of user involvement

For different user involvement strategies, the state in which the batch cluster can be accessed by the task performers can be different (i. e., *init*, *ready*, *running*) which has different implications being shown in Figure 28. The earlier a batch cluster is shown to a task performer, the more the user is involved. Further, the actions by task performers on batch clusters can be strengthened or limited. Thereby, we differentiate two basic actions by task performers: they can start the batch execution, or can adapt a batch cluster (cf. Figure 28). Next, the user action possibilities per state, in which a batch cluster can be, are discussed.



State in which batch cluster is shown	Actions of user involvement				Degree of user involvement
	Start batch cluster		*Adapt batch cluster*		
		Implication		*Implication*	
RUNNING	Batch cluster is only auto-started	*With fulfillment of activation rule, batch cluster is auto-started*	**- no adaptation allowed**		
READY	**Task performer can start batch cluster at any time**		- can exclude instances	*Excluded instance(s) are assigned to another batch cluster*	
			- can join batch cluster with another one, if *max BatchSize* is not reached	*Add instances of the other cluster to the selected one and delete it*	
INIT	Task performer can start batch cluster at any time after its initialization	*Batch cluster can also transition from init state into running state*	- can add future arriving instances for which the batch cluster has to wait, if *maxBatchSize* is not reached	*Batch cluster has to check, whether assigned future instances might still arrive. Despite the activation rule might be fulfilled, batch cluster is only enabled, if future instances have arrived.*	

Figure 28: Possibility for user involvement in different states of a batch cluster and their implications - the current configuration is shown in bold.

The latest point in time in which a batch cluster can be presented to a task performer is in its *running* state. In this case, the batch cluster has to be allocated to a resource, and is then, auto-started similar to *Commencement on Allocation* resource patterns described in [95]. The task performer has to immediately start with its execution. Currently, the batch cluster is provided as soon as the activation rule is fulfilled (i. e., in the *ready* state). Here, users can decide for themselves, when to start with the execution. For increasing the user involvement, the batch cluster could be already shown when it is initialized (i. e., in the *init* state). Then, users can decide on their own when the most optimal point in time is to start a batch. Thereby, they are still supported by the BPMS which indicates whether the batch cluster is still *init* or already *ready*.

No matter which state a cluster has, BPMS providers can configure to which extent the task performers are allowed to adapt a batch cluster. Currently, no adaptations are allowed on the batch cluster. Additionally, we differentiate three adaptation possibilities:

- users can exclude instances from a cluster,
- they can join a cluster with another one, or
- they can add process instances which arrive in future at the batch activity.

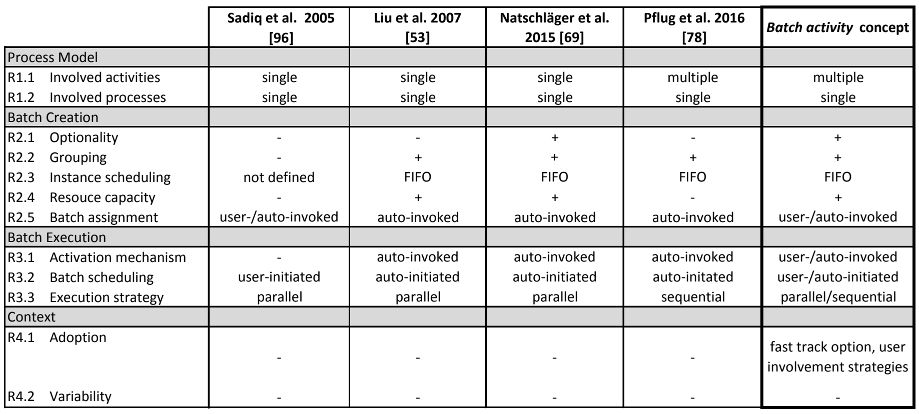
When allowing users to exclude instances from a batch cluster, these instances need to be assigned to a different cluster which is still not *maxloaded*, *running* or *terminated*. If no fitting cluster exists, a new one has to be initialized. For allowing task performers to join different batch clusters, it has to be ensured that the join does not contradict the *maxBatchSize*. For joining the two clusters, instances of one batch cluster are added to the other one and the selected one is deleted. For the last option in which users can add future arriving instances on which the batch cluster has to wait, the batch cluster is obliged to check regularly whether the assigned future instances might still arrive. Future instances can only be assigned, if the *maxBatchSize* is not reached, yet. With adding future instances, the batch cluster, despite the activation rule might be fulfilled, is only enabled, if they have arrived.

In this section, different options for involving users in the batch assignment, activation, and scheduling were presented. However, increasing the user involvement is also connected with certain risks. For instance, batch cluster adaptations could lead to errors or the optimization potential is not fully used anymore. We assume that BPMS providers configure the batch activity in their process engine based on their customer requirements. A possibility might be to provide different kinds of batch activities, such as an *automatic batch activity* with less user involvement, and a *user-involved batch activity* with more adaptation possibilities for the task performers.

5.5 conclusion

This chapter presented the main concept of the thesis – the batch activity. A batch activity is a new process modeling element with several configuration parameters and with an own execution semantics to support its automatic execution in a BPMS. Additionally, *data views* were introduced for characterizing process instances based on their data, and identifying similar ones. Data views were used, in case of the batch activity, to group and cluster process instances in specific batches. The *activationRule* – the configuration parameter to activate a batch for execution – was discussed in detail, because this rule is essential for balancing between the overall cost reduction by batch processing, and the possible increase in cycle time for a single process instance. It is based on the general concept of ECA rules to enable a flexible design. In the course of this chapter, several examples of activation rules were presented, e. g., the threshold rule known from queuing theory, and extensions of it to consider also future instances or a fast track option. At the end of this chapter, different user involvement options during batch creation and execution, and their implications on the concept were presented. Here, it was shown that the batch activity concept supports automatic as well as user-involved batch activities in business processes.

Next, we want to compare the presented *batch activity concept* to related research work based on the requirements framework. The comparison is shown in Figure 29. Thereby, we have focused on the works presented in [53, 69, 78, 96] which aim at improving the process performance by extending it with batch processing.



	Sadiq et al. 2005 [96]	Liu et al. 2007 [53]	Natschläger et al. 2015 [69]	Pflug et al. 2016 [78]	*Batch activity* concept
Process Model					
R1.1 Involved activities	single	single	single	multiple	multiple
R1.2 Involved processes	single	single	single	single	single
Batch Creation					
R2.1 Optionality	-	-	+	-	+
R2.2 Grouping	-	+	+	+	+
R2.3 Instance scheduling	not defined	FIFO	FIFO	FIFO	FIFO
R2.4 Resouce capacity	-	+	+	-	+
R2.5 Batch assignment	user-/auto-invoked	auto-invoked	auto-invoked	auto-invoked	user-/auto-invoked
Batch Execution					
R3.1 Activation mechanism	-	auto-invoked	auto-invoked	auto-invoked	user-/auto-invoked
R3.2 Batch scheduling	user-initiated	auto-initiated	auto-initiated	auto-initated	user-/auto-initiated
R3.3 Execution strategy	parallel	parallel	parallel	sequential	parallel/sequential
Context					
R4.1 Adoption	-	-	-	-	fast track option, user involvement strategies
R4.2 Variability	-	-	-	-	-

Figure 29: Comparison of existing batch processing solutions based on the requirements framework.

With regards to the *Process Model* category, it can be observed that most related research works focused so far on single activities, only Pflug et al. [78] considers sequences of activities. We have extended this to batch processing over a set of connected activities, including also concurrent behavior. However, all existing solutions including the presented one are currently bound to single business processes.

Regarding the *Batch Creation* category, the developed batch activity concept supports all mentioned requirements; thereby, we want to highlight the following aspects: *Optionality* is supported by the *batch activity*, if no activation rule is defined. Then, each arriving instance is compared with other enabled activity instances of the batch activity, and in case of matching partners, those are executed by one cluster. Natschläger et al. [69] considers also possible future instances which might arrive. If future instances assigned to a batch do not arrive anymore, the batch simply stops waiting in their concept. In this chapter, the *MinMaxRule* is more flexible in this regards; it waits for any similar future instance as long as the maximum timeout is not reached. If the future instances are not available anymore, the threshold is lower and can lead to an immediate activation of the respective batch cluster. As given in Figure 29, almost all given solutions support the *grouping* of process instances into specific batches. Whereas in [53, 69] as well as in the work of this thesis the grouping constraints are defined at design time, in [78] instances are dynamically grouped for each batch assignment. Dynamically grouping requires a clustering algorithm which leads to meaningful results for the respective use case. Finding the most useful clustering algorithm and its correct specification is connected with high effort. We decided for a static grouping mechanism, because we observe in the collected use cases presented in Section 4.1 that the grouping parameters are usually known before, e. g., by customer, by country, by responsible employee. With the presented *dataView*-concept, the grouping parameter can be quickly defined by process designers. Regarding the *batch assignment*, the approaches by [53, 69, 78] follow an auto-invoked batch assignment approach, because they want to ensure the efficient generation of batch clusters with the support of technology. Additional to that, this chapter discussed different possibilities for users to adapt batch clusters. In [96], batches are either generated by a system or created by the users whereas we suggest a combined approach where the system proposes batch clusters which users might adapt.

The *Batch Execution* category includes the batch activation mechanism. The activation mechanism is essential for batch processing as it defines when to activate a batch cluster to balance cost reduction and waiting time. For supporting effective batch processing in business processes, almost all solutions provide an auto-invoked batch activation. In the work by [53, 77], batch activation is connected with the batch assignment, because scheduling algorithms are used. As soon as a resource is free, instances are assigned to batches from which a batch is scheduled on the resource, and immediately activated. However, it assumes that resources executing the batch are only involved in the batch activity. In practice, task performers are usually responsible for different activities in several processes. Therefore, in the batch activity concept presented in this chapter, batch clusters are created and activated independently from the resource availability similar to the concept of work items in BPMSs [130]. Thereby, activation rules are based on ECA rules and can be flexibly designed. We assume that the activation rules are defined by the BPMS vendors which have the technical expertise and an overview about different use cases from their customers. Process designers in turn can select between different provided activation rules, and only have to fill out requested inputs without taking care about the technicalities. In contrast, in [69], Natschläger et al. request process designers to specify a complex optimization function and constraints.

Additionally to the auto-invoke mechanism, this work also discusses the possibility that users can start batch clusters on their own, if a use case scenario requests more flexibility in the batch activation. In this presented batch activity concept, batch clusters are allocated after their activation to task performers similar as work items. It allows that batch clusters can be allocated based on different resource allocation patterns [95]. Based on the selected resource allocation pattern, task performers could either individually organize their queue of batch work items, or it can be organized by the BPMS. Therefore, the presented batch activity allows auto-invoked, as well as user-initiated batch scheduling in contrast to the other works. Furthermore, the batch activity supports both batch execution strategies, whereas other solutions either support *parallel* or *sequential* batch execution (cf. Figure 28).

Regarding the *Context* category, existing works have not discussed flexibility aspects in batch processing. In this chapter, first flexibility aspects were introduced. Currently, the adaptation of a batch activity by considering special cases with a *fast track option* in the batch activation and allowing users to adapt batch clusters. Wong et al. present in [135] a technique to monitor batch activities and exceptions in manual process environments, or environments with heterogeneous IT-systems. However, they do not discuss how to react and handle changes in process environment automatically in the batch creation and execution.

Based on the above given discussion of the presented batch activity concept in comparison to other solutions, we can derive two important directions for future research. On the one hand, the enlargement of flexibility aspects during batch processing, and on the other hand, the extension to multi-process batch processing. In the following chapters, extensions of the developed batch activity concept are presented which target these two aspects.

Part III

EXTENDED CONCEPTS

6 FLEXIBLE BATCH CONFIGURATION WITH EVENTS

The previous chapter presented the batch activity concept to enable batch processing in business processes. Several configuration parameters allow the process designer to individually setup the batch execution. However, specifying the rules at design time does not always guarantee optimal batch execution, since changes in the process environment represented by events occurring during process execution might influence it. Reacting on these events and changing the specified configuration parameters is required for increasing the process flexibility. In this chapter, event processing techniques are applied to the batch activity to react on relevant changes. Therefore, batch adjustment rules are introduced and their integration in the BPMS architecture is given. The application of batch adjustments to a healthcare use case in a simulation implicates that they help to compensate losses caused by the exceptional behavior. This chapter is based on the published paper "Flexible Batch Configuration in Business Processes based on Events" [88].

Changes in the process environment might influence batch processing

Events represent "real world happening occurring in a particular point"' [38]. They indicate and inform about changes or exceptions in the business process environment. The sensing and observation of events is important for business processes to allow an adequate reaction on changes [55].

Events may also influence the execution of batch activities. For example, in the online retailer example, if a certain parcel size is not available anymore, the maximum batch size should be reduced. As the rules for batch execution are defined at design time, a batch activity cannot react flexibly on such events. Task performers might observe such changes and might adapt batch clusters, if necessary, based on the user involvement strategies introduced in Chapter 5. This approach is driven manually and highly depends on the amount of time which users have to observe batch clusters. Therefore, this chapter aims at an automatic support. It introduces a concept to apply event processing techniques [29] to batch activities to allow flexible adjustments of the batch configuration based on run time changes. Event types are selected at design time that trigger an adjustment of certain batch clusters at run time.

This chapter provides (i) an overview on how events can change each batch configuration parameter, and (ii) a concept which allows the flexible adaptation of the batch activity configuration triggered by an event occurrence.

The chapter is structured as follows. Section 6.1 presents a motivating example originating from a real world scenario of the healthcare domain. This is followed by an analysis on how events may influence a batch configuration and presents the corresponding requirements. Section 6.2 introduces the concept of flexible adaptation of batch activities based on event processing techniques. In Section 6.3, this concept is validated by applying it to the healthcare scenario in a simulation environment. Section 6.4 concludes the chapter.

6.1 MOTIVATING EXAMPLE AND REQUIREMENTS

In this section, first a healthcare process, a blood testing process is presented to motivate the need for flexible batch processing. Then, the influence of events on each batch activity's configuration parameter is analyzed. Based on this analysis, requirements for flexible batch processing are deduced.

Blood Testing Process

Motivating example

In Figure 30, the blood testing process briefly described in Section 4.1 is captured as Business Process Model and Notation (BPMN) process diagram. If there is a blood test required for a patient at the ward, the process is initiated. First, the blood test order is prepared before a blood sample is taken from the respective patient. Afterwards, a nurse transports the sample and the blood test order 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. Then, the results are accessible by the physicians in the respecting ward. There, they can evaluate the blood test result for a patient and can use it for diagnostics.

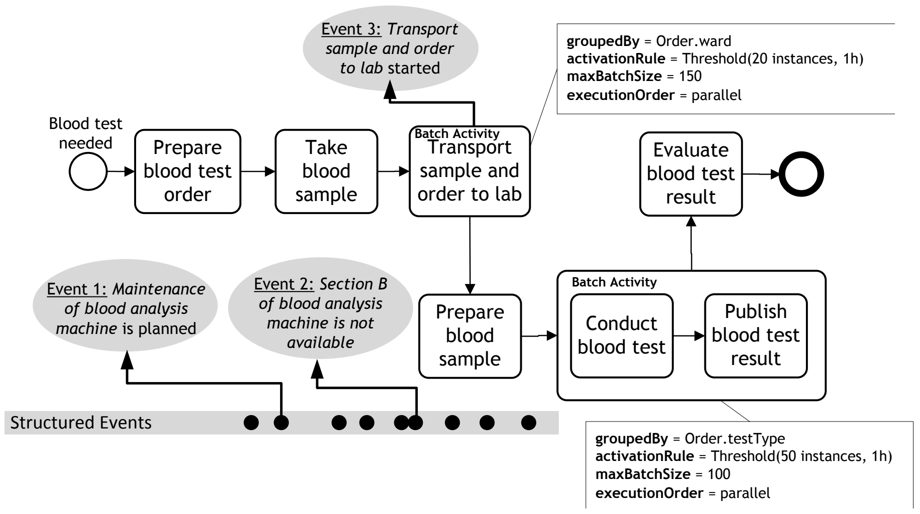

Event 3: Transport sample and order to lab started
groupedBy = Order.ward
activationRule = Threshold(20 instances, 1h)
maxBatchSize = 150
executionOrder = parallel
Blood test needed
Prepare blood test order
Take blood sample
Batch Activity
Transport sample and order to lab
Evaluate blood test result
Event 1: Maintenance of blood analysis machine is planned
Event 2: Section B of blood analysis machine is not available
Prepare blood sample
Batch Activity
Conduct blood test
Publish blood test result
Structured Events
groupedBy = Order.testType
activationRule = Threshold(50 instances, 1h)
maxBatchSize = 100
executionOrder = parallel

Figure 30: Blood testing process with two batch activities.

Within the given process, two batch activities 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 batch activity *Transport sample and order to lab*. The second batch activity is a sub-process which consists of the 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 cannot 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 activities in the blood testing process:

Example events influencing the batching of blood samples

Planned maintenance of a machine: This event indicates that the maintenance of a machine is planned. During the maintenance, the machine is not available to conduct tests of the specific type. Blood samples being part of a batch cluster which is not yet enabled 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.

Partial unavailability of a machine: Let us assume that a blood analysis machine contains four sections to process blood samples among 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: It might happen that 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. In the next subsection, an analysis is performed to set the requirements for flexible batch cluster adjustment.

Events and Batch Activities

As discussed above, it is valuable for organizations to design batch processing in a flexible manner. Thus, created batch clusters can be adjusted according to the changes of the process environment notified by events. Here, adjustments refer to changes on the batch cluster configuration parameters. Therefore, we want to analyze how the configuration of a batch cluster can be adapted and by which events. Further, we want to discuss the validity of events and in which states a batch cluster can be still adjusted.

Possible adjustments of a batch activity's configuration

Table 3 provides an overview how the configuration parameters (1) *groupedBy*, (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*) with corresponding event *examples*.

Configuration parameter	**Type of changes**	**Influence**	**Events indicating**	**Examples**
groupedBy	- aggregate - refine - restructure	- cancel existing batch cluster(s) 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 might have 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 the blood testing 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 the blood testing 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

Table 3: Classification on how batch clusters can be changed and by which events.

In the table, all types of adjustments are considered. It is ensured that for each parameter still a value is given which can also be undefined for the first three parameters. Usually, the configuration of a batch cluster is only adapted as reaction on an event. In case of changing the *groupedBy*-parameter, 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 match the adapted *groupedBy*-parameter. For example, a grouping regarding the *Order.ward* results in batch clusters with data views *General Surgery* and *Endoscopic Surgery*. If the *groupedBy*-parameter 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*. Whereas a change of the activation rule has only influence on the moment when a batch cluster is activated, reducing the maximum batch size may result in batch clusters exceeding the newly set limit. Then, the newest assigned process instances are removed from the corresponding cluster and get assigned to another or a new batch cluster accordingly. Hence, a concept of flexible batch adjustments needs to consider that process instances might need to be reassigned to another cluster.

Validity of events

Events can be relevant for none, one, or a set of existing batch clusters. Further, events can be valid for a certain time frame, such that an event might be relevant for batch cluster created after the occurrence of an event. The validity of an event should be considered during the event correlation.

As described in the previous chapter, 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:

(R-1) Identify event types relevant for adjusting batch clusters of a batch activity,

(R-2) Correlate the events to the respective cluster at run time by considering the validity of events,

(R-3) Adjust batch cluster correspondingly including reassignment of instances being removed from clusters.

6.2 flexible configuration concept

In the following, we describe the basic idea of the batch adjustment concept by referring to the blood testing example introduced in Section 6.1. Afterwards, the newly introduced *batch adjustments* and their *batch adjustment rules* are presented, before we explain an approach for process instance reassignment and describe an architecture for realizing the presented batch adjustment concept.

Basic Idea

We assume that events are centrally observed and stored by an Event Processing Platform (EPP). If a relevant event is observed, the corresponding batch cluster gets adjusted accordingly, cf. Figure 31.

The batch adjustment concept builds on structured events. As described in Section 2.5, a structured event is a derivation of an event object consisting of an identifier, a time stamp, and some structured event content, e. g., a set of key-value-pairs or a tree-structure expressed in

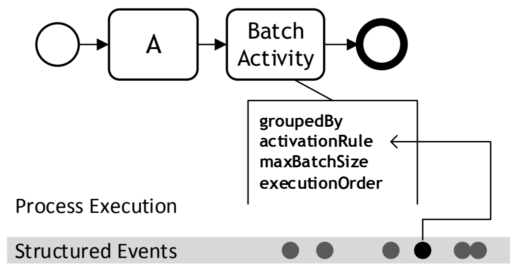


Figure 31: Events influencing the properties of a batch cluster during run time.

the Extensible Markup Language (XML). Structured events are usually the result of one or several normalized events processed by the EPP. A structured event type describes a class of structured events that have the same format.

Flexibility of batch configurations

We propose a concept 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 might need to be applied are defined in the *batch adjustment rule*. The structure of a batch adjustment rule follows Event-Condition-Action (ECA) rules originating from the database domain [19]. Each ECA rule consists of events E triggering the rule, a condition C which has to be satisfied, and an action A being executed in case of fulfillment of the condition. 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 are interested in the event. The described action specifies the particular adjustment of a batch cluster, e. g., the immediate execution of a batch cluster.

The connection of events and the batch activity concept is illustrated in the class diagram of Figure 32. One batch model can have an arbitrary set of batch adjustment rules which are provided by the process designer. They extend the set of configuration parameters of batch activities. A batch adjustment rule refers to at least one structured 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 an 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. Batch adjustment rules, and the resulting batch adjustments are explained in detail using an example in the next subsection.

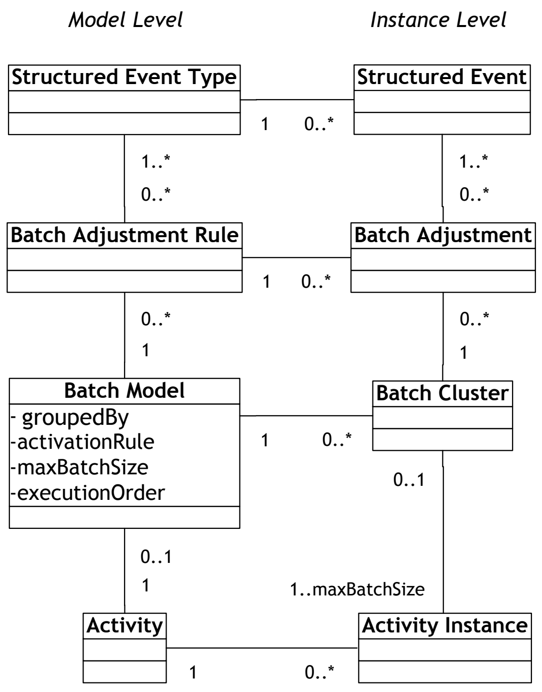


Figure 32: Conceptual model for integrating batch activity and event processing concepts represented by a UML Class Diagram [72]. The model level shows the design time concepts and the instance level shows their run time artifacts.

Batch Adjustment Rule and Batch Adjustments

A *batch adjustment rule*, following ECA rules, describes how and under which conditions a batch cluster needs to be adjusted during run time. In this subsection, we first give an example of a *structured event*, describe then the structure of a *batch adjustment rule* followed by the structure of a *batch adjustment*.

STRUCTURED EVENT. The events that need to be considered for an adjustment of a batch cluster are described by their event type. For example, an event type describes events that indicate a planned maintenance of the blood analysis machine, cf. Listing 6. The event should be provided some time before the maintenance starts, such that batch clusters, which has not be enabled yet, can be activated and finished before the start of the maintenance. This information is composed of fine-grained information of normalized events indicating the maintenance need and the schedule of the service technician.

The 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 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.

```
machineMaintancePlanned_b.extraction =
{machineMaintancePlanned_b.id = getGuid();
 machineMaintancePlanned_b.timeStamp = getTime(now);
 SELECT
   machineStatus_n.name,
 FROM
   machineStatus_n,
   technicianSchedule_n
 INTO
   machineMaintancePlanned_b.MachineName
 WHERE
   machineStatus_n.name =
   technicianSchedule_n.machineID AND
   machineStatus_n.status = "MaintenanceNeeded" AND
   technicianSchedule_n.state = "planned" AND
   technicianSchedule_n.time − getTime(now) <= machine(name).getRuntime()
 }
```

Listing 6: Definition of the structured 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$_n$).

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. The event shall be created exactly one machine run before the maintenance 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*).

BATCH ADJUSTMENT RULE. The event type *machineMaintancePlanned*$_b$ can be used as trigger for a batch adjustment rule which adapts the activation rule of batch clusters to avoid expired blood samples in case of a maintenance. The proposed batch adjustment rule is shown in Listing 7, illustrating its basic structure. The condition part of the batch adjustment rule ensures that batch adjustments are only created for batch clusters for which the event is of relevance. In our example, the events of type *machineMaintancePlanned*$_b$ are relevant for all batch clusters which have the same blood testing type as the machine to be maintained and are not yet enabled for execution (i. e., in state *init*). Those should be enabled before the maintenance takes place.

The instances of the *blood testing* batch activity are grouped based on their blood test type (cf. Figure 30) with groupedBy = *Order.bloodTestType*. Thus, the batch cluster's data view provides information which blood test type its assigned process instances require, e. g., *BC1(BloodTestA)*. The data view of the batch cluster can be used for the condition, cf. Listing 7 line 2 and 3.

Structure of a batch adjustment rule

```
ON EVENT
   (machineMaintancePlanned_b)
IF CONDITION
   (batchCluster.dataView == machineMaintancePlanned_b.name AND
            batchCluster.state == "INIT")
ACTION
   batchcluster.activationRule=Threshold (50,0h)
```

Listing 7: 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 its characteristics, i. e.,

1. data view,
2. current state,
3. number of instances contained in a cluster, and
4. characteristics of the cluster's 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 has occurred 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 should enable the batch execution before maintenance, cf. Listing 7 line 4. With this action, the activation rule of the cluster is adjusted, in this case a threshold rule which was introduced in this thesis in Section 5.2. It is adapted, such 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. Batch adjustment rules are used to create batch adjustments for batch clusters. 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 8 is generated for batch cluster 1234.

Example of a batch adjustment

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

Listing 8: 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 performed. In case 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 subsection discusses the special case where a batch cluster is not only adapted, but a reassignment of process instances is necessary.

Reassignment of Process Instances - Adapting execution semantics of the batch activity

A batch adjustment usually results in the adaptation of the configuration of one batch cluster. Sometimes, it can also trigger (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 *groupedBy*-parameter. The extended lifecycle of batch clusters with the *canceled* state is shown in Figure 33; a cancellation is only possible from states *init*, *ready*, and *maxloaded*. In all cases, process instances have to be reassigned to other or new batch clusters.

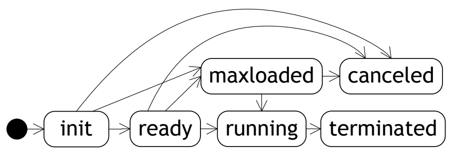


Figure 33: Lifecycle of batch cluster extended by *canceled* state.

In general, process instances that arrive at a batch activity are temporarily deactivated and assigned to the queue of the batch activity in the order of their arrival time (first-come-first-served (FCFS)). The batch activity organizes the assignment of process instances to batch clusters and, if necessary, initializes new batch clusters.

Procedure of reassigning instances

If a process instance, in case of an adjustment, is reassigned, it should be prioritized, because it already experiences a longer waiting time than newly arriving instances at the batch activity. Thus, the reassigned process instance is placed in the front of the queue based on its arrival time at the batch activity. Then, it is assigned to an existing, or new batch cluster. In the example of Figure 34, the number of instances of the batch cluster BC1 have to be reduced because an event indicated that an error of a machine section has occurred. 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 activities 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 Figure 34, the threshold

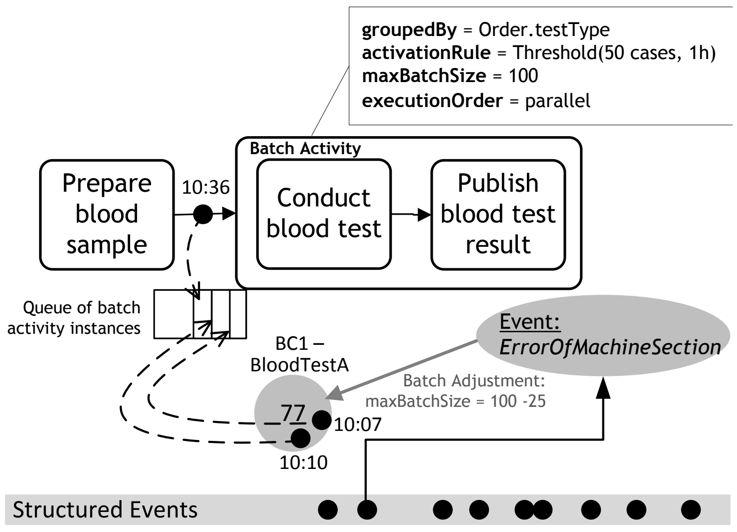

groupedBy = Order.testType
activationRule = Threshold(50 cases, 1h)
maxBatchSize = 100
executionOrder = parallel
Batch Activity
Prepare blood sample
10:36
Conduct blood test
Publish blood test result
Queue of batch activity instances
BC1 – BloodTestA
77
10:07
10:10
Event: ErrorOfMachineSection
Batch Adjustment: maxBatchSize = 100 -25
Structured Events

Figure 34: Reassignment of process instances in case of a reduced *maxBatchSize*.

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 activity 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 activity.

Architecture

Next, an architecture is proposed for the technical implementation which supports the flexibly adaption of batch cluster configurations. Figure 35 presents the main components and their interactions as Fundamental Modeling Concepts (FMC) block diagram [47]. The architecture is structured into three parts: *event producer*, *EPP*, and *BPMS*. The *process engine* of the BPMS, which controls process execution and batch handling, is an event producer and event consumer at the same time. It consumes events provided by the EPP by being connected to the *Event Consumer Interface*.

Several event producers (*event sources*) can be connected via an appropriate *event adapter* to the EPP. The EPP itself normalizes the received raw events and creates structured events based on defined rules. Event consumers are connected by an *event consumer interface*.

Extended BPMS architecture to integrate batch adjustments

The *BPMS* comprises the process engine and the modeling environment (cf. in Section 5.3). With the latter, process models can be created for being executed within the process engine which are stored in the

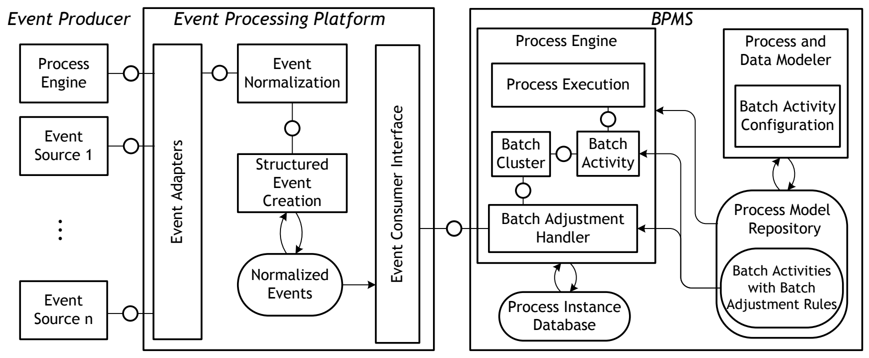


Figure 35: System architecture to realize batch adjustments during process execution.

process model repository. While modeling a process, batch activities can be designed with the batch activity configurator. Thereby, the process designer can define batch adjustment rules used at run time to adapt batch activities. Those are saved together with the process model in the repository on which the process engine has access during run time. As soon as a process model with a batch activity is deployed on the process engine, a *batch activity* object is initialized in the engine. This *batch activity* object is responsible to assign process instances to batch clusters. During process deployment, the *batch adjustment handler* registers for events at the *event consumer interface* that are specified in the *batch adjustment rules* of a batch activity. If the handler receives a registered event from the EPP, then the event is evaluated and the batch adjustments for the relevant batch clusters are created which trigger the corresponding actions. 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. Additionally, the event is stored for its validity period, if any, to allow later application to new batch clusters.

6.3 VALIDATION OF FLEXIBLE BATCH ACTIVITY CONFIGURATIONS

In this section, we present an application of the introduced batch adjustment concept to the blood testing process described in Section 6.1 to study its effect. For this, we conducted a single-case mechanism experiment [133]. Single-case mechanism experiments are used in design science to apply an artifact to its context in a laboratory environment to predict its influence and its contributions. Therefore, we applied the concept in a simulation of the business process to compare the effect of batch adjustments to the situation without them (i. e., normal batch execution).

As described, the laboratory uses a batch activity to synchronize several blood samples for the blood analysis to save machine costs. The blood analysis machine needs to be maintained regularly 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, because the blood structure changes. Then, the blood sample is not useful for medical analysis anymore. An often conducted blood test is the coagulation test which has to be undertaken the latest after 4 hours according to guidelines [80]. However, the guidelines do not provide statistics of expiration of blood tests. By interviewing process experts, we assumed an expiration time after 120 minutes. Each expired blood sample causes costs of taking a new one.

Simulation setup

For the evaluation, the blood testing process is simulated 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 chapter. Therefore, the laboratory part of the blood testing process was implemented as simulation[1] with DESMO-J [33, 111], 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. On 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 one blood analysis machine exists. One run of the machine for analyzing blood samples takes 25 minutes. The machine can handle maximum 100 blood samples in one analysis.

For the simulation, the laboratory selected *ThresholdRule(50 instances, 1h)* as activation rule requiring 50 instances or a timeout after one hour to enable a batch cluster (cf. Figure 30). 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. Thereby, different durations of the maintenance are simulated: 30, 45, and 60 minutes, for testing the effect of the batch configuration adaptation for different settings of the context. Additionally, an simulation experiment without any maintenance is executed to compare the results of it to the flexible batch handling.

For the flexible batch handling, some time before the technician arrives, an event regarding the maintenance is provided. For studying how much before an event should be provided for having a positive impact, different simulation experiments are conducted where the event is sent 1, 1.5, or 2 times the blood analysis run, i. e., 25, 37.5, or 50 min-

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

utes respectively, before the technician arrives. When the technician arrives, they are prioritized, but a current analysis on the machine is not interrupted.

Results

Several simulation runs with different settings to observe the impact of flexible batch adjustments were conducted. Figure 36 summarizes the results of the simulation runs over a period of two years. It shows the results for a maintenance time of 30 minutes, 45 minutes and 60 minutes (intercept 2, 3 and 4) 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 25, 37.5, or 50 minutes before the technician arrives.

If no maintenance is conducted, 1,738 samples would expire due to exponential arrival of these blood samples, and resulting waiting times at the machine. If the maintenance is conducted at average twice a week as indicated above, the number of expired blood samples increases by 0.7%, 14%, and 29% for 30, 45 and 60 minutes maintenance duration, respectively (cf. black bars in Figure 36). It can be observed that the number of expired blood samples increase with longer maintenance durations. Whereas for 30 minutes no difference can be observed, in case of 60 minutes, already 29% more blood samples would become unusable.

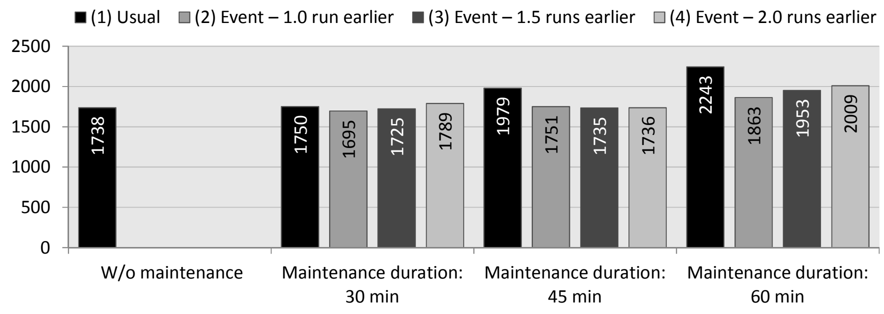


Figure 36: Number of expired blood samples in two years for different simulation runs.

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 7). 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 case of 30 minutes maintenance time, only small reductions, and even an increase can be observed if the event is published 50 minutes before the maintenance. For the others, we observe significant improvements. The improvement is at 13% and 20% for 45 and 60 minutes maintenance time, respectively.

With these numbers, the maintenance was almost compensated. The highest improvements for the different settings are mostly observed for the middle gray bar ((2) Event – 1.0 run earlier). It indicates that it is beneficial to inform about the maintenance one analysis run before the start of the maintenance.

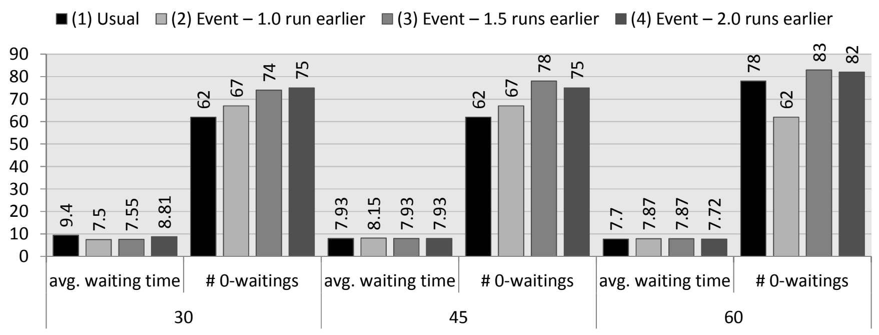


Figure 37: Average waiting time and zero-waitings for the technician in the different settings.

The simulation results indicate that the waiting time for the technician slightly increases, on average less than a minute as shown in Figure 37. 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. Nevertheless, the cost savings due to reductions in expired blood samples will be higher than the technician costs due to small increases in the waiting time.

Summarizing above observations, flexible batch adjustments are useful in cases where certain external events have a strong influence on the batch processing, as in case of maintenance with a duration of 45 or 60 minutes. Further, the improvements of the batch adjustments are also dependent on the time of an event being published. This should be analyzed and validated in advanced.

6.4 conclusion

In this chapter, we showed the necessity to adjust batch executions flexibly due to run time changes. A concept was introduced to apply event processing techniques to batch activities enabling flexible adjustments of batch configuration parameters based on events representing run time changes. Grounded in 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. The concept also includes the possibility to reassign process instances of an adjusted batch cluster to avoid inconsistencies. In future, it should be evaluated whether further techniques are necessary to ensure that batch adjustments does not lead to inconsistencies .

An architecture was presented showing details about a technical implementation and the components that are necessary to apply the concept with a BPMS in conjunction with a EPP. We showed the applicability of the introduced concept of batch adjustments on a healthcare use case in a simulated environment. It implies that batch adjustments can compensate losses caused by the exceptional behavior, if the external events have a strong influence on the batch processing. Further, it also indicates that the time when event is published is relevant for the success of the adaptation.

By integrating more information about the process environment, e. g., the availability of resources, the presented concept can be extended. Currently, batch adjustment rules are a technical approach, such that the process designer might need support by an IT specialist for the definition of batch adjustment rules. In future, the concept can be enhanced with a more user-friendly interface for specifying the rules.

7 BATCH PROCESSING ACROSS MULTIPLE BUSINESS PROCESSES

As the developed batch activity concept focuses on instances of a single process, this chapter introduces a batch processing concept across multiple business processes. Organizations being active in Business Process Management (BPM) often manage large collections of process models where similar activities (or even process fragments) can be found. This offers the opportunity to save execution cost over activity instances of different business processes. In this chapter, we first set the design objectives based on a motivating example, and then, present a concept which uses data object life cycles to centrally define the batch requirements ending with a discussion of it. This chapter is based on the published paper "Batch Processing across multiple Business Processes based on Object Life Cycles" [84].

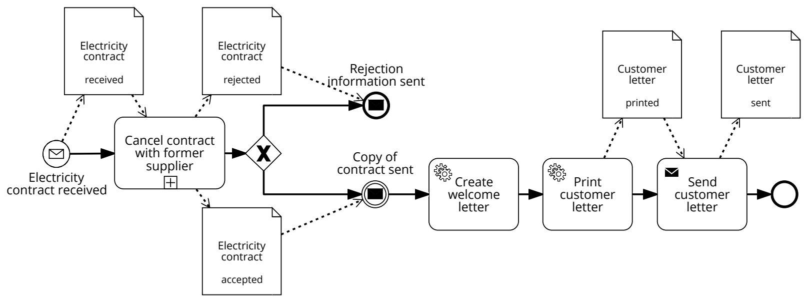
Large process model repositories offers opportunity for multi-process batch processing

Existing solutions discussed in Chapter 3 [53, 69, 77, 96] which enable batch processing in business processes are limited to process instances of a single process model. Also, the introduced batch activity in Chapter 5 focuses on single process models. However, we can observe that organizations, which document their business processes, have to manage large repositories of hundred, or even thousand of process models [28, 136]. In those, certain process activities, or even process fragments, might be reused in multiple process models. For instance, in finance, sending notifications to the customer is required in multiple processes, such as account opening, credit card issue, and loan approval. Although customer notifications are created in different processes, they can be batched for sending only one letter to the customer instead of several ones for saving cost. Despite different techniques for clone detection, e.g. by Dumas et al. [25], still not all repeating activities are outsourced into shared sub-processes, especially small process fragments consisting of one or two activities.

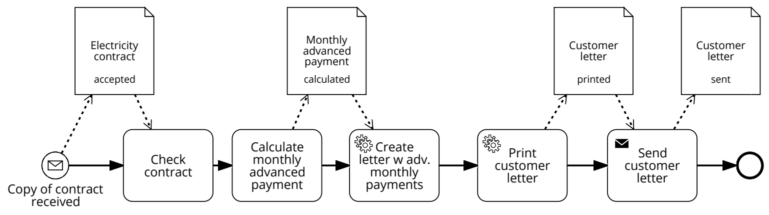
Based on these considerations, this chapter introduces a concept to allow batch processing across the process model boundaries. In a large process model repository, process models are frequently added, deleted, and changed. Therefore, we aim at a centralized batch configuration which is valid for multiple process models and is independent of such updates of the repository. Here, object life cycles (OLCs) [45] describing the allowed manipulations on a data object (cf. Section 2.3) are used. With OLCs, identical behavior of business processes on their data objects across the process model boundaries can be identified. They are centrally given and accompany process models.

In following section, a motivating example is discussed in detail from which specific requirements of batch processing across multiple processes are deduced in Section 7.2. Further, the design objectives are set. Section 7.3 introduces the concept of multi-process batch processing by considering OLCs. This basic concept for single data state transitions is then extended to multiple, connected data state transitions in Section 7.4 to generalize it. Section 7.5 concludes the chapter. The prototypical implementation of the concept is discussed in the evaluation part in Chapter 8.

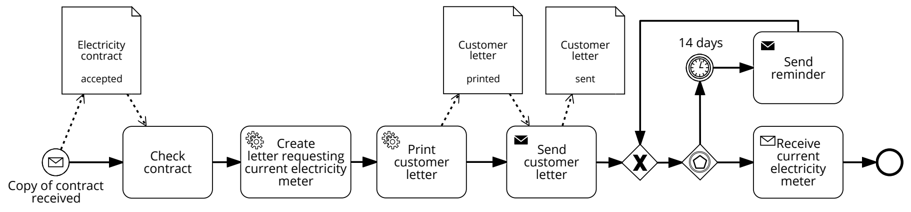
7.1 MOTIVATING EXAMPLE


Electricity contract received
Electricity contract rejected
Rejection information sent
Customer letter printed
Customer letter sent
Electricity contract received
Cancel contract with former supplier
Copy of contract sent
Create welcome letter
Print customer letter
Send customer letter
Electricity contract accepted

(a) Contract acceptance process.


Electricity contract accepted
Monthly advanced payment calculated
Customer letter printed
Customer letter sent
Copy of contract received
Check contract
Calculate monthly advanced payment
Create letter w adv. monthly payments
Print customer letter
Send customer letter

(b) Advanced payment process.


Electricity contract accepted
Customer letter printed
Customer letter sent
14 days
Send reminder
Copy of contract received
Check contract
Create letter requesting current electricity meter
Print customer letter
Send customer letter
Receive current electricity meter

(c) Meter reading process.

Figure 38: Business process models of an electricity supplier for handling a new customer with similar activities.

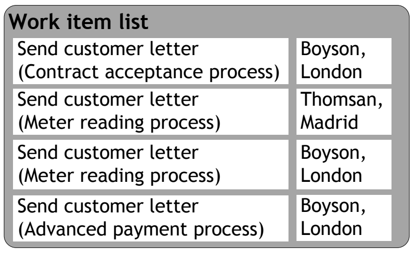
This section discusses the motivation based on the example of an electricity supplier. The supplier manages a repository to document its business processes. Figure 38 represents three business processes of this repository upon which we motivate our work and the requirements.

The model in Figure 38a shows the process of getting a new electricity contract by a customer which is either accepted or rejected. If the electricity contract is *rejected*, the customer is informed about that. In case of acceptance, a welcome letter is created which is then printed and sent to the customer. As soon as an electricity contract is accepted, a copy is also sent to other departments where the *Advanced payment process*, shown in Figure 38b, and the *Meter reading process*, shown in Figure 38c, are initiated. In the *Advanced payment process*, the monthly advanced payment rates, which the customer has to pay, are calculated based on the contract information. The rates are included in a customer letter which is printed and sent to the customer. In the *Meter reading process*, the customer is requested to provide the current electricity meter. Therefore, a letter is created that is then printed as customer letter and sent to the customer.

In these three processes, customer letters are printed and sent to the customer. These repeating activities allow to batch activity instances from multiple process models. Let us discuss this option on the *Send customer letter* activity.

Advantages of consolidated handling of the customer letters

Executing these processes in a business process management system (BPMS) may lead to a situation where the employee John has four assigned work items as depicted in Figure 39. It can be observed that four instances of activity *Send customer letter* are assigned to John; three of them require a sending to customer *Boyson* in London – one for each of the three presented processes – and one of them requires a sending to customer *Thomsan* in Madrid. Although handled by three different processes, the three customer letters to be sent to *Boyson* could be enveloped and sent as one mail.



Work item list

Send customer letter (Contract acceptance process)	Boyson, London
Send customer letter (Meter reading process)	Thomsan, Madrid
Send customer letter (Meter reading process)	Boyson, London
Send customer letter (Advanced payment process)	Boyson, London

Figure 39: Work item list of employee *John* showing recently assigned activity instances.

Consolidating the handling of these three customer letters by putting them in one envelop has the advantages that (1) cost for sending mails can be reduced, and (2) the customer receives only once one envelop with all important information. If we assume mailing costs of 0.50 EUR per letter, the company can already save for this example 1.00 EUR. If we assume around 500 new customers per month, the electricity supplier can save up to 6000 EUR per year. Based on these considerations, this chapter introduces a concept to allow batch processing across different process models. Involving several process models in the batch execution leads to specific requirements which we want to discuss in the following section.

7.2 REQUIREMENTS AND DESIGN OBJECTIVES

This section discusses requirements specific to batch processing across multiple process models. Further, it describes how we want to approach the given requirements by discussing our design objectives.

Requirements in a Multi-Process Setting

Based on the introduced example, four requirements are identified which are presented in detail.

R1-CENTRALLY DEFINED BATCH SPECIFICATION. At design time, the batch processing specification has to be defined by the process designer. Process models, also the ones given by the electricity supplier, are constantly adapted to changing environments; they are added, changed, and deleted from process repositories [136]. Therefore, the batch configuration should be given independently from individual process models, centrally defined and accessible by all targeted process models.

R2-RESOURCE AUTHORIZATION. If instances of different processes are batched, the question arises whether all resources being involved in the different processes are allowed to execute the batch. In our example, the first two processes are executed by the *Customer Service*, whereas the meter reading process is in responsibility of the *Meter Reading Data Management*. The batch of *Send customer letter*(s) could be executed by employees of both departments. However, it might be required that only employees of the *Customer Service* are allowed to handle such batches. This requires a concept for authorizing resources to execute batches over multiple processes.

R3-OPTIONAL BATCH PROCESSING. Most of the existing batch processing solutions for business processes, except [69], have a compulsory approach and wait for an explicit number of activity instances to be executed as batch. If, for example, two instances are required to start a batch execution, the activity instance of *Send customer letter* for customer *Thomsan, Madrid* (see Figure 39) can only be started when a second letter for this customer is available. This means that the advanced payment letter is sent later than intended leading to a delay in payment. In the given case, the cost savings for sending only two letters together may not compensate the costs for waiting for the second letter. Thus, an optional batch processing approach seems to be beneficial, if more than one process model are involved.

R4-USER APPROVAL FOR BATCH ASSIGNMENT. However, neither run time information, nor design time specifications might cover all information required to decide whether multiple activity instances of different models shall be batched. For example, if the welcome letter to customer *Boyson* (see Figure 39) has to sent individually due to a special marketing campaign, then this letter shall not be batch together with the other two letters. Thus, the user-involvement should be increased in a multi-process setting in which the task performer approves firstly the

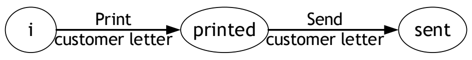


Figure 40: Object life cycle of the data class *Customer Letter*.

proposed batches – whether all, some or no process instance is executed as batch.

Design Objectives and Assumptions

In the following, it is described how we want to approach the given requirements. Let us start with requirement R1, the centrally defined batch specification.

Using OLCs for a central batch configuration

As described in Section 2.3, business processes act on data; process activities read data and update them. The allowed data manipulations by process activities for one class of data objects are given in a OLC. OLCs are centrally given and accompany process models.

In Figure 40, the OLC for the objects of type *Customer Letter* is given, derived from the three given process models in Figure 38. Actions used for transitioning from one state to another directly map to activities in the corresponding process models, e. g., to reach state *sent* from state *printed*, activity *Send customer letter* must be executed. The transitions of an OLC can be used to enrich them with batch information which then applies to all process models using the corresponding action.

Requirement R2, the resource authorization, will be not targeted by the introduced concept in this chapter. It is assumed that the same activity in different process models has the same resource assignment and that all resources are allowed to execute the created batch clusters.

If batch processing is optional, then an instance is only processed in a batch, when other instances are available to which it can be grouped. In the BPMS architecture given in Section 2.4, it was shown that a BPMS has access to a storage containing all data objects created by its process instances. These data objects will be used to tackle Requirement R3, the optional batch processing, to identify whether instances can be executed as a batch. Data query language provide fast means to identify similar data objects.

For requirement R4, the final user approval of the batch assignment, we aim at a concept where the system calculates potential batches and proposes these candidates to the task performer. The task performer, then, finally decides which identified activity instances can be handled as a batch and which not. This leads to a composition of a user-invoked and auto-invoked batch assignment where first the system proposes an assignment which can be adapted by the user.

7.3 BATCH SPECIFICATION IN OBJECT LIFE CYCLES

In this section, we introduce the concept for specifying the batch execution in OLCs. We distribute the discussion into two parts. First, the design of batch transitions in OLCs is formally defined followed by the corresponding execution semantics.

Design

Following the preliminaries in Chapter 2, one OLC per data class is centrally defined in an organization. A state transition can be used in different process models. For our purpose, a new type of state transition: the *batch transition* is introduced. A batch transition carries information to create, and subsequently to execute batch clusters.

Batch Transition Definition

Definition 7.1 (Batch Transition).
Let $t \in T_{olc}$ be a transition of an object life cycle olc. Then, function $\beta : T_{olc} \rightarrow \{true, false\}$ returns true, if t is a *batch transition*, and returns false otherwise. $T'_{olc} \subseteq T_{olc}$ denotes the set of all batch transitions of olc. Each batch transition gets assigned a batch configuration function $\gamma : T' \rightarrow DVD$ where DVD is the set of all data view definitions. A data view definition $dvd \in DVD$ is a list of data attributes $X = [x_1, x_2, \ldots, x_k]$ where for each $x \in X$ holds that $x = c.j$, such that $c = lc_{-1}(olc) \wedge j \in J_c$. ◀

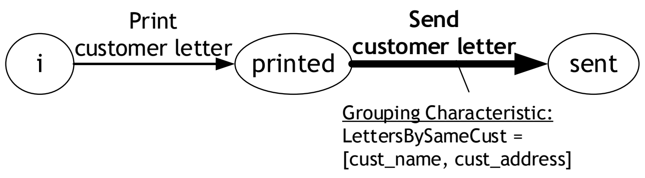


Figure 41: Object life cycle of data class *Customer Letter* with the batch transition *Send customer letter*.

Figure 41 shows an example OLC for data class *Customer Letter* of the electricity supplier scenario. It contains a *batch transition* – the one referring to action *Send customer letter* – with a specific grouping characteristic. We visualize batch transitions by bold edges and connected with a batch configuration, a textual annotation including the grouping characteristic. All process activities using this batch transition are candidates for batch processing. The grouping characteristic helps to identify which type of instances can be processed together. It references a data view definition which we have introduced in Section 5.1. In the concept presented in this chapter, the data view definition DVD can only contain data attributes of the data class $lc_{-1}(olc) = c$ to which the object life cycle olc with the batch transition t belongs. In the given example, the data view definition consists of data attributes *cust_name & cust_address*; hence, only activity instances for which the customer letter objects are addressed to the same customer may be executed as a batch.

An activity of a process model can have several output data nodes referring to different data classes. Thus, several object life cycle transitions may exist for an activity. Following the concept of case objects from business artifacts [16], one class of objects drives the execution. Thus, we require only one of the transitions being for a *batch transition*. The related data manipulation is in the focus of the batch processing, and the remaining outputs are byproducts.

Execution Semantics

Algorithm of creating batch work items across multiple processes

Based on the design time batch specifications, we present the algorithm on processing batches of activity instances, first focusing on user tasks and later discussing the generalization. The algorithm consists of five subsequent steps as visualized in Figure 42. First it is checked whether

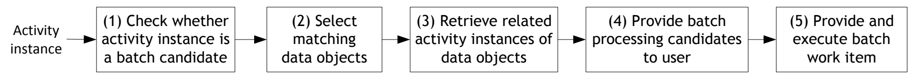

Activity instance
(1) Check whether activity instance is a batch candidate
(2) Select matching data objects
(3) Retrieve related activity instances of data objects
(4) Provide batch processing candidates to user
(5) Provide and execute batch work item

Figure 42: Main steps of the algorithm to batch instances of multiple processes.

an activity instance is a batch candidate (1), then all relevant data objects for batch processing are selected (2) to which the related activity instances are retrieved (3), the batch processing candidates are proposed to the user (4) and finally, if the user has selected more than one instance, the batch work item is created and provided (5). The steps for the running example are visualized in Figure 43 and explained in detail in the following.

(1) CHECK WHETHER ACTIVITY INSTANCE IS A BATCH CANDIDATE. During process execution, activity instances can be in states *init*, *ready*, *running*, *terminated*, and *disabled* (cf. Section 2.4). If an activity instance is *ready* for execution, all matching OLC transitions $\mathcal{U}$ of the corresponding activity $a \in A$ are retrieved by function $\epsilon : A \rightarrow \mathcal{P}(\mathcal{U})$ introduced in the preliminaries in Section 2.3. The resulting set $\mathcal{U}$ of transitions is checked for the existence of a batch transition, i. e., each transition $t \in \mathcal{U}$ is checked whether $\beta(t) = \mathsf{true}$. As mentioned above, we require that β returns *true* for at most one transition per activity. If a batch transition $\widehat{t}$ was found, the activity instance $\widehat{ai}$ is a candidate for batch processing, and the corresponding data object $\widehat{o}$ is retrieved. Therefore, an auxiliary function $\theta : I \times C \rightarrow O$ is needed which returns the data object o belonging to the given process instance $i \in I$ for the given data class $c \in C$. Hence, the batch candidate object $\widehat{o}$ is retrieved with $\theta(i_{\widehat{ai}}, lc_{-1}(olc)) = \widehat{o}$ where the process instance $i_{\widehat{ai}}$ is given by the currently checked activity instance $\widehat{ai}$. The data class is given by using the inverse function $lc_{-1}(olc) = c$ returning the data class c for the ob-

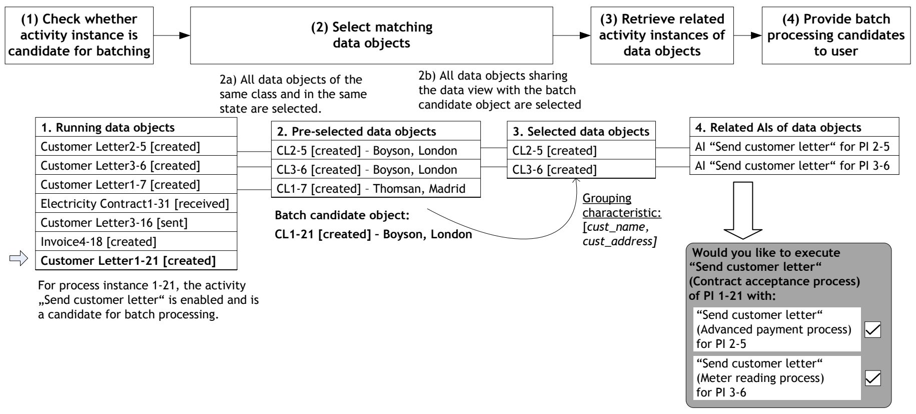

(1) Check whether activity instance is candidate for batching
(2) Select matching data objects
(3) Retrieve related activity instances of data objects
(4) Provide batch processing candidates to user
2a) All data objects of the same class and in the same state are selected.
2b) All data objects sharing the data view with the batch candidate object are selected
1. Running data objects
Customer Letter2-5 [created]
Customer Letter3-6 [created]
Customer Letter1-7 [created]
Electricity Contract1-31 [received]
Customer Letter3-16 [sent]
Invoice4-18 [created]
Customer Letter1-21 [created]
For process instance 1-21, the activity „Send customer letter“ is enabled and is a candidate for batch processing.
2. Pre-selected data objects
CL2-5 [created] - Boyson, London
CL3-6 [created] - Boyson, London
CL1-7 [created] - Thomsan, Madrid
Batch candidate object:
CL1-21 [created] - Boyson, London
3. Selected data objects
CL2-5 [created]
CL3-6 [created]
Grouping characteristic: [cust_name, cust_address]
4. Related AIs of data objects
AI "Send customer letter" for PI 2-5
AI "Send customer letter" for PI 3-6
Would you like to execute "Send customer letter" (Contract acceptance process) of PI 1-21 with:
"Send customer letter" (Advanced payment process) for PI 2-5 ✓
"Send customer letter" (Meter reading process) for PI 3-6 ✓

Figure 43: Illustration of algorithm to process activity instances of multiple process models in a batch.

ject life cycle olc to which the batch transition $\hat{t}$ belongs. For example, activity instance *1-21* is identified as batch candidate. Since, the identified batch transition belongs to the *Customer Letter* object life cycle, the *Customer Letter* object in state *printed* and with id *1-21* is retrieved (see last row of most-left table in Figure 43).

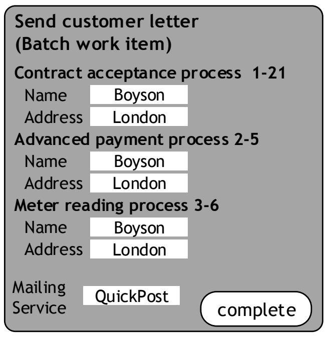
(2) SELECT MATCHING DATA OBJECTS. Next, all data objects sharing the data class and data state (i. e., the value of the data attribute j_{state} representing the OLC state) with the in (1) identified data object $\hat{o}$ are pre-selected. In Figure 43, the most-left table shows all currently processed data objects O of the electricity supplier. The *Customer Letter* in state *printed* referring to the process instance with id *1-21* to be processed by the current activity instance is the candidate for batch processing (shown in bold font). The second table of Figure 43 visualizes the projection O' on the set of data objects O such that all objects refer to data class *Customer Letter* and are in state *printed*. In the given example, this are three data objects in total. From this set, all objects sharing the data view with $\hat{o}$ are finally selected. In our example, the grouping characteristic consists of data attributes *cust_name* and *cust_address* of data class *Customer Letter*. For the current processed data object $\hat{o}$, we find the values *Boyson* and *London*. All data objects $O'' \subseteq O'$ having identical values (i. e., the same data view) are selected as shown by the third table in Figure 43. If the data view definition is empty or not defined, then $O'' = O'$ holds. These two described steps can be also conducted by one data base query.

(3) RETRIEVE RELATED ACTIVITY INSTANCES OF DATA OBJECTS. Batch processing works on activity instance level. Thus, next, the activity instances of the selected data objects O'' must be determined. As given in Definition 2.6, each object contains its corresponding process instance. For activity instance determination, the auxiliary function $\eta : O \rightarrow AI$ is used. AI denotes the set of all currently existing

activity instances. The η-function retrieves the corresponding related activity instance which is currently in state *ready*, and refers to an activity a reading a data node d, where the state matches the one from $\hat{o}$ $((d, a) \in DF)$. The most-right table in Figure 43 shows the resulting activity instances for the data objects O''. Identified activity instances may refer to different process models. In our example, one identified activity instance refers to the *Advanced payment process*, and the other to the *Meter reading process* while the batch candidate refers to the *Contract acceptance process*.

(4) PROVIDE BATCH PROCESSING CANDIDATES TO USERS. The identified activity instances are transferred from state *ready* into state *disabled*; a disabled activity instance is temporarily deactivated in its processing. Then, the user is asked which of the identified activity instances shall be processed in a batch with the batch candidate. In the example, activity instances for activity *Send customer letter* of the process instances *2-5* and *3-6* may be grouped in a batch with the activity instance of *1-21*. The gray-colored box on the bottom-right in Figure 43 visualizes this. One of the responsible task performers decides whether all additionally identified activity instances, a subset of them, or none are joined into one *batch work item*. Equally to the batch activity, also here a batch work item aggregates the work items of all selected activity instances allowing joint visualization, and execution in one step. The task performer is then responsible for executing the batch work item, if at least some activity instances are joined. Otherwise, the batch candidate is handled as a single activity instance. Finally, those activity instances that are not selected for batch processing are transferred back into state *ready*.

(5) PROVIDE AND EXECUTE BATCH WORK ITEM. All selected activity instances, and the batch candidate that are jointly realizing the *batch work item* are added to a batch cluster. A batch cluster is a container collecting activity instances that can be executed together. It may pass multiple states during its lifetime as described in the batch activity concept in Section 5.3. First, a batch cluster is *init*ialized. In the next state, the *ready* state, it aggregates the data input of all contained activity instances into a single work item – the batch work item. Further activity instances can be added still in this state by expanding the batch work item accordingly. Ad-


Send customer letter
(Batch work item)

Contract acceptance process 1-21

Name	Boyson
Address	London

Advanced payment process 2-5

Name	Boyson
Address	London

Meter reading process 3-6

Name	Boyson
Address	London

Mailing Service	QuickPost

Figure 44: Resulting batch work item of Figure 43 if all activity instances are selected.

ditions are allowed until the task performer starts its execution (i. e., the batch cluster transitions into state *running*). Figure 44 visualizes the batch work item for the example under the assumption all identified activity instances were selected. Upon completion of the batch work item, the output data is stored by the batch cluster for each activity instance individually. Then, all activity instances are terminated. Subsequently, the batch cluster terminates as well, and the succeeding activity instances are handled again separately.

Approach for service tasks

As introduced in the preliminaries, activities are distinguished into user, service, or unspecified tasks. Handling of user tasks is discussed above. Targeting, service tasks, step (4) needs to be automated by adding all batch candidates to the batch. Alternatively, a responsible role can be specified to handle this step. Step (5) remains conceptually unchanged. Instead of creating the batch work item, the batch cluster aggregates the input data (i. e., into a format which can be handled by the referenced service) and provides it to software service. Unspecified task can be not executed in a BPMS, such that it needs to be transformed into a user or service task.

The presented concept is evaluated as proof-of-concept in a prototypical implementation – an extension of an existing open-source BPMS – in Chapter 8. In the next section, an extension of this basic concept is discussed to several connected transition to allow, on the one hand, batch processing over multiple connected activities (or data transitions) and, on the other hand, batch processing over activities which have different inputs but the same output.

7.4 EXTENSION TO CONNECTED BATCH TRANSITIONS

This section discusses a generalization of the concept being currently focus on single batch transitions to allow also the synchronization over several connected transitions. First, the need for connected batch transitions is motivated followed by the definition of the batch configuration, which is valid for several batch transitions and the description of its execution semantics.

Extended requirements

Necessity for batch processing over multiple subsequent data transitions

In practice, the necessity for batch processing over multiple subsequent data transitions can be observed. Let us take the example of the

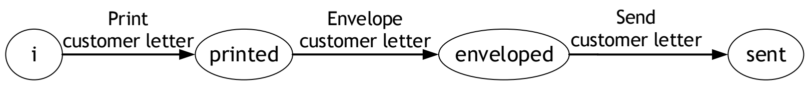


Figure 45: Extended OLC of the data class *Customer Letter* with the new state *enveloped* for automatic enveloping of letters

electricity supplier: Usually in an automated mail print environment, a letter is printed, then it is immediately automatically put in an envelope, and finally it is sent. The object life cycle is adapted to met this data flow changes shown in Figure 45. An additional state *enveloped* is added between the states *printed* and *sent*, and an additional data transition *Envelope customer letter* is supplemented which allows the transition between the *printed* to *enveloped*. Besides the *Send customer letter* transition, batch processing should include the printing and enveloping of customer letters as well. It ensures that the customer letters which should be sent together are also put in one envelope. Thus, the batch execution should start in this example with the *Print customer letter*, and end with the *Send customer letter* transition. This example shows that also in multi-process environments batch processing is needed for a set of connected activities (or transitions, respectively) similar to the batch activity.

Necessity for batch processing over multiple similar data transitions

Further, it can be observed in existing business processes that activities providing the same result, e. g., sending a letter, might have different inputs. These activities are similar in this regard that they handle data objects of the same class, but the input states of these objects are different. For instance, an activity in a process sends unsigned letters whereas an activity in another process sends only signed letters. They would map to different object transitions, despite these two activities conducting the same action on the data objects. The algorithm presented in the previous section cannot handle these cases properly. Only letters being in the same data state before sending can be executed as batch. Despite of different input states, customer letters can still be grouped as batch and sent within one envelope, if they share the same addressee.

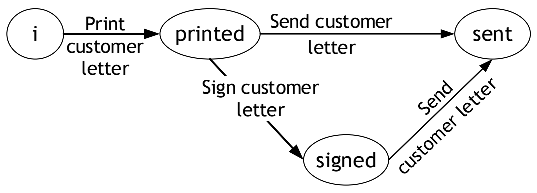


Figure 46: Extended OLC of the data class *Customer Letter* with the new state *signed* to allow the signing of letters

Assume, the *Contract acceptance process* in Figure 38a requires the *Customer Letter* to be signed before it is sent, a corresponding activity is added between activities *Print customer letter* and *Send customer letter*. The data flow is adapted accordingly. The OLC is changed as well to match these data flow changes which is visualized in Figure 46. The additional data state *signed* is added as successor of state *printed* and as predecessor of state *sent*. The corresponding data state transitions get associated to the actions *Sign customer letter* and *Send customer letter* respectively.

Design

For serving the discussed needs, we want to introduce connected batch transitions. Connected batch transitions allow, on the one hand, batch processing over several succeeding activities (or transitions), and on the other hand, batch processing over similar transitions having different input states, but the same output state. Therefore, the batch configuration function is extended and gets independent from single batch transitions for being valid for a set of batch transitions. We define a batch configuration as follows:

Batch Configuration Definition

Definition 7.2 (Batch Configuration)**.**
Given an object life cycle olc with a set of T_{olc} transitions, the batch configuration assigns a data view definition $dvd \in DVD$ to a subset $T'_{olc} \in \mathcal{P}(T_{olc})$ by the function $\gamma : BT \rightarrow DVD$, where $BT \subseteq \mathcal{P}(T_{olc})$. $\forall T_{bt} \in BT \rightarrow \forall t \in T_{bt} : \beta(t) = true \wedge \forall T'_{bt} \in BT : T_{bt} \cap T'_{bt} = \emptyset$. For $t_1 = (s_n, s_m), t_2 = (s_x, s_y) \in T_{bt}, t_1 \neq t_2$, it also holds:

- $(s_n, s_m), (s_x, s_y) \in T_{bt} \Rightarrow (((s_n, s_m), (s_x, s_y) \in (v_0, ..., v_n)$, where $(v_i, v_{i+1}) \in T_{bt})$ (sequentially connected transitions) $(\vee(s_m = s_y))$ (transitions with different input state, but the same output state)

◀

A *batch configuration* describes the batch processing requirements for a set of connected transitions which are all batch transitions ($\beta(t) = true$). Each batch transition takes only part in one subset. The batch transitions of such a set can be connected in two different ways: Either the transitions create a path through the object life cycle, or all batch transitions share the same output state.

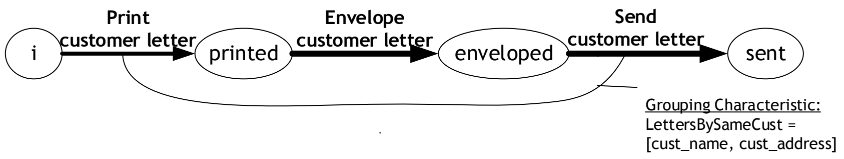


Figure 47: OLC of the *Customer Letter* with the sequential connected batch transitions.

Visually, the batch configuration is represented by associations between the corresponding batch transitions. We assume that the object life cycle designer decides based on the business context which transitions can be connected. In case of sequential connected transitions as shown in Figure 47, the association line starts at the first transition of the sequence and ends with the last one. The grouping characteristic is connected to the association line. Similar, it is visualized for connected batch transitions with the same output.

As shown in Figure 48, an association line connects similar batch transitions to which also the grouping characteristic is attached. Based on the presented design adaptations, the execution semantics needs to be adapted as well. These adaptations are discussed in the next subsection.

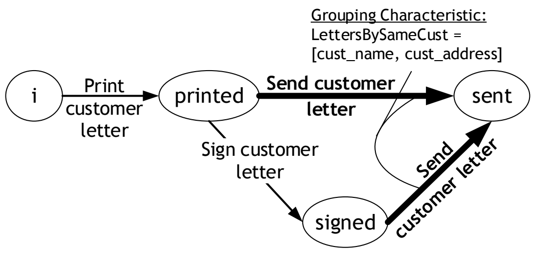


Figure 48: OLC of the *Customer Letter* with connected batch transitions between similar transitions with different inputs.

Extension of the Execution Semantics

Extension for sequential connected batch transitions

For sequentially connected batch transitions, the execution semantics does not change for the first batch transition of the sequence. However, after its execution, the cluster is not terminated. Additionally, two changes are done on the existing execution semantics, such that no new cluster has to be created for the succeeding batch transitions, in specifically in step (1) and step (5).

- Step (1): After it was identified that an activity references a batch transition, it has to be additionally checked whether the process instances of the just enabled activity instance is part of a batch cluster. If this is true, the activity instance gets *disabled*. All steps are skipped until step 5, because the cluster already exists. As soon as all instances of the cluster are *disabled* for currently running activity, the cluster aggregates the activity execution information of all contained activity instances and provides the batch work item to the task performer. As the cluster is already in the *running* state, no instances can be added anymore.
- Step (5): This step has to be extended with a check after the termination of the batch work item whether the cluster can be terminated. Thereby, the cluster checks whether all defined batch transition of a batch configuration are already completed. If this is the case, the cluster transfers into the *terminate* state and deletes all references to it.

Extension for similar batch transitions

For supporting the selection of activity instances referring to different batch transitions with a different input, but having the same output state, step (2) has to be extended.

- After checking whether the currently enabled activity instance is a candidate for batch processing, the set O' of pre-selected data objects is expanded. All data objects sharing the same data class and being in one of the data states that are source states of the connected batch transitions are added. Thereby, alignment of data objects to processes must be considered since the same data state may be utilized differently in process models (cf. state *printed* of data class *Customer Letter*). In the aforementioned example, this includes all data objects of class *Customer Letter* being in state *printed* – if they refer to the *Advanced payment process* or *Meter reading process* –, or *signed* – if they refer to the *Contract acceptance process*.

The remaining steps are applied as described in the previous section.

7.5 CONCLUSION

This chapter presented a concept to process a group of activity instances of different processes as batch in order to improve process performance in large process collections with repeating activities. The requirements for batch processing are centrally defined in OLCs which describe the allowed data manipulations of data objects used in the business processes. The presented concept is an optional batch processing approach that is started as soon as matching partners for an instance can be identified based on run time information, the currently existing data objects. Further, identified batch processing candidates are proposed to a task performer who selects the proper candidates. This ensures a correct batch assignment in a multi-process setting. We consider the presented concept light-weight, since it only uses few additional concepts (most prominently the batch transitions) which are well embedded in existing research works.

Currently, the multi-process batch processing concept allows all task performers who are permitted to execute one of the activities referencing the batch transition to perform the batch. In future, the current work could be enhanced by a user authorization concept.

In this chapter, activity instances are not delayed to avoid unnecessary waiting times. Further cost saving potentials by grouping future activity instances also into a batch are not considered yet. Considering this, the current batch configuration of the batch transition can be easily extended by the configuration parameters of the batch activity, for instance, with the activation rule. Thereby, activation rules specific for the multi-process setting should be developed.

Part IV

EVALUATION AND CONCLUSIONS

8 PROOF-OF-CONCEPT IMPLEMENTATION

After having introduced the batch activity concept and extending this to batch processing across multiple business processes, this chapter presents the prototypical implementation of the concepts to evaluate their feasibility by extending an existing open-source BPMS. For the implementation, the Camunda BPM platform, a Java-based, lightweight BPMN process engine was used. The implementation shows that with a few extensions, batch processing is enabled. It also demonstrates that the consolidated view of several work items in one user form leads to an improved work efficiency for users. This chapter is partly based on the published paper "Enabling Batch Processing in BPMN Processes." [85].

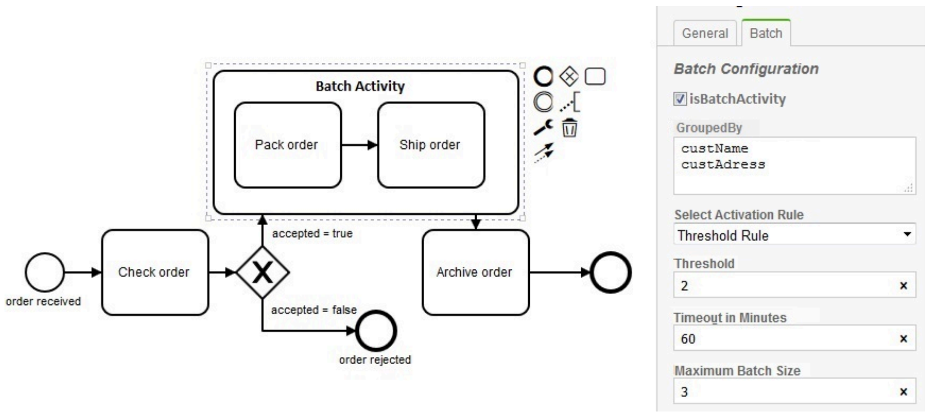
The batch activity concept presented in Chapter 5, and the batch processing across multiple business processes presented in Chapter 7 is realized by extending a Java-based, open-source BPMN process engine, the Camunda BPM platform [13]. This business process management system (BPMS) enacts process models imported in the BPMN XML format. Additionally, Camunda provides an open-source BPMN process modeler called `bpmn.io`. This modeler is used to enable the modeling, and configuration of batch activities. The implementation and the screen casts are available at `http://bpt.hpi.uni-potsdam.de/Public/BatchProcessing`.

First, the prototypical implementation of the batch activity concept is presented in Section 8.1. It is followed by the demonstration of the prototypical implementation for batch processing across multiple business processes in Section 8.2. A summary of the results is given in Section 8.3.

8.1 IMPLEMENTATION OF THE BATCH ACTIVITY

For the implementation, the *Camunda Modeler* `bpmn.io` was adapted to enable a quick design of batch activities. More precisely, the sub-process element was extended as shown in Figure 49. Besides the general sub-process attributes, batch configuration parameters, e. g., the *groupedBy*-parameter, the activation rule and the maximum batch size can be defined by the process designer in the extended *Camunda Modeler* (cf. Figure 49). In the prototypical implementation, currently *parallel* batch execution is supported.

As a result, the *Camunda Modeler* can produce a BPMN XML file based on a designed process model, which can be then deployed in the Camunda BPM platform for process execution. If a batch activity is added in a process model, then the extended *Camunda Modeler* adds



Batch Configuration

☑ isBatchActivity

GroupedBy
custName
custAdress

Select Activation Rule: Threshold Rule

Threshold: 2

Timeout in Minutes: 60

Maximum Batch Size: 3

Figure 49: `bpmn.io` extension to configure batch activities on the example of the online retailer process with a batch activity to save shipping costs.

extension elements to the sub-process for adding the batch configuration as described in Section 5.2.

Figure 50 depicts a high-level architecture of the Camunda BPM platform as FMC block diagram[1] with the extensions for enabling batch processing. Thereby, adapted components are represented in yellow and new added components are shown in orange. As already described, the *Camunda Modeler* was extended, such that batch activities with its configuration parameters can be defined. As shown in Figure 50, a process designer can design a process with it, and then, store the resulting model as BPMN XML-file in a process model repository accessed by the *Camunda Engine*. Additionally, the process designer can create HTML forms for each specified user task in the process model. They also have to be stored, such that they are accessible by the engine.

Extended Camunda architecture with batch processing capabilities

The *Camunda Engine* itself was extended by four additional classes and the *BPMN Parser* was adapted. The added classes, and the adaptions are described in detail in the following.

The *BatchActivity* class stores the configurations of each identified batch activity in a BPMN XML-file and manages the assignment of process instances to batch clusters. The *BatchCluster* class governs the batch execution for its assigned group of process instances. The *BatchBehavior* class includes the internal behavior of a batch activity, or of each activity being part of a batch activity sub-process, respectively. In the *Camunda Engine*, every process activity gets a task behavior (e. g., user task, service task) assigned, describing the internal activity behavior. In order to reuse these behaviors and limit the engine extension, the *BatchBehavior* is an extension of the normal task behavior and includes additional methods for the batch execution defined by an interface. This is driven by the idea that one of the cluster instances leads the batch execution by first merging the data of all cluster instances, and then, executing the usual task behavior. Currently, this is implemented for user tasks

1 A short introduction into FMC block diagrams can be found in Section 2.4

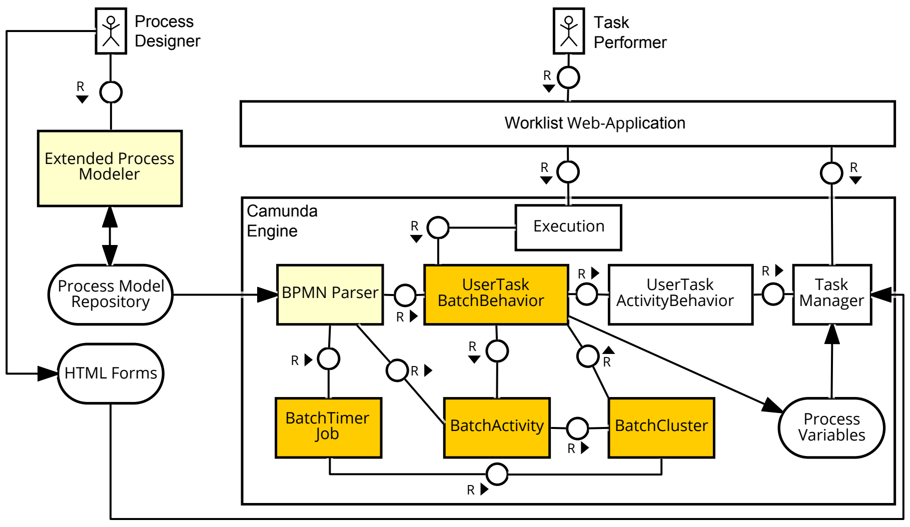


Figure 50: Architecture of the extended Camunda engine with batch processing capabilities (as FMC block diagram [47]): adapted components are shown in yellow and new components are shown in orange.

(cf. *UserTaskBatchBehavior* in Figure 50), but can be applied in a similar way to service and script tasks which is explained later in this section. A *BatchTimerJob* class was added to enable the time-out defined in the threshold rule. Additionally, the BPMN parser was adapted to read batch activities' specifications, and to instantiate a *BatchActivity* object and a *BatchTimerJob* object for each defined batch activity in a process, as well as to instantiate a *BatchBehavior* object for each activity of a batch activity sub-process.

Returning to the overall architecture presented in Figure 50, process participants, the so-called *Task Performers*, can access the *Camunda Engine* via the *Worklist Web-Application* in which they can start new process instances of deployed process models, and can perform user tasks. For each started process instance, Camunda creates an *Execution* object representing the state of the corresponding process instance. An *Execution* can access the activity behavior of the currently enabled nodes of a process instance. In Figure 51, the interaction of an *Execution* with the *BatchBehavior*, the *BatchActivity*, and the *BatchCluster* class is shown on the example of a user task.

As soon as an *Execution* object enables an activity with a batch behavior, it is added by the *BatchActivity* to a cluster. If no batch cluster is currently available, a cluster is first created, and then, the `add()`-method of the cluster is called in which also the activation rule is checked. Currently, our implementation supports the threshold rule. If a batch cluster fulfills a batch activation rule, the cluster calls the `composite()`-method of the *BatchBehavior* merging the data of all instances. In case of the user task, a JSON variable with all instance data is created and stored as process variable. This can be later reused during the user form design. In case of a service or script task, the Camunda engine

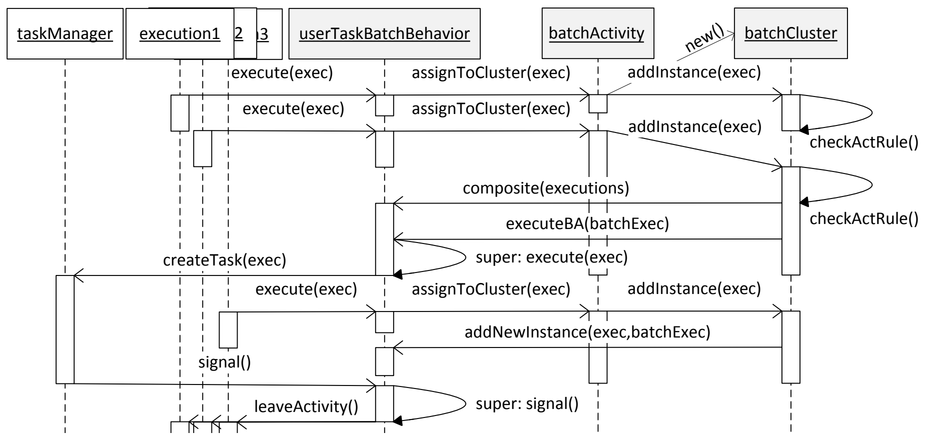


Figure 51: Sequence diagram visualizing the interaction among added classes *BatchBehavior*, *BatchActivity*, and *BatchCluster* to the *Camunda Engine*.

calls either a specified Java class or a script defined by a process developer. Thereby, the JSON variable can be also used to execute code on the aggregated instance data, for example, calling an external service with aggregated input. In this case, the Java service or the script has to be designed in a way that it is able to handle the JSON variable.

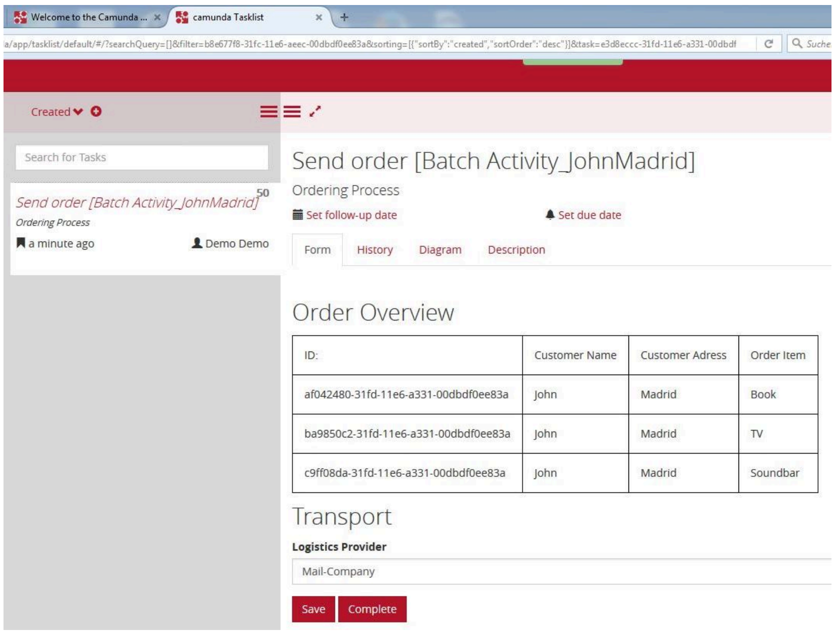
After the `composite()`-method has prepared the JSON variable, the `executeBA()`-method is called with one representative instance of the cluster, the *batchExecution*. With the `executeBA()`-method, the *BatchBehavior* calls the `execute()`-method of its super class. Now, the normal *UserTaskActivityBehavior* is executed in which a work item for the task performer is prepared. As soon as the task performer wants to access the batch work item in the *Worklist Web-Application*, the *TaskManager* takes the created *HTML Forms* by the process designer, and the stored JSON variable (cf. Figure 50) to create the content of the batch work item.

Visualization of a batch work item

Figure 52 shows the batch work item for the *Ship order* activity of the retailer example. The JSON variable was used to visualize all orders of one customer in a table. The task performer can easily inspect all orders and has to enter the value for the logistics provider only once, as it is valid for all orders. Instead of three work items, the task performer has to process only one, which can lead to time savings.

Our implementation provides also the feature to add new instances, while the first activity in a batch activity sub-process is not completed yet. The corresponding `addNewInstances()`-method simply adapts the JSON variable.

With completion of a batch work item, the *TaskManager* calls the `signal()`-method of the *BatchBehavior* distributing newly added data (e. g., the logistics provider in Figure 52) to all other cluster instances. Finally, with the last batch work item, also the batch cluster is terminated. The implementation shows that a small number of extensions are necessary in a BPMS to enable batch processing, which have no


Welcome to the Camunda ...
camunda Tasklist
a/app/tasklist/default/#/?searchQuery=[]&filter=b8e677f8-31fc-11e6-aeec-00dbdf0ee83a&sorting=[{"sortBy":"created","sortOrder":"desc"}]&task=e3d8eccc-31fd-11e6-a331-00dbdf
Suche
Created
Search for Tasks

Send order [Batch Activity_JohnMadrid] 50
Ordering Process
a minute ago Demo Demo

Send order [Batch Activity_JohnMadrid]
Ordering Process
Set follow-up date
Set due date

Form
History
Diagram
Description

Order Overview

ID:	Customer Name	Customer Adress	Order Item
af042480-31fd-11e6-a331-00dbdf0ee83a	John	Madrid	Book
ba9850c2-31fd-11e6-a331-00dbdf0ee83a	John	Madrid	TV
c9ff08da-31fd-11e6-a331-00dbdf0ee83a	John	Madrid	Soundbar

Transport
Logistics Provider
Mail-Company

Figure 52: Batch work item for the *Ship order* activity of the online retailer example visualized in the Camunda *Worklist Web-Application*.

significant influence on the engine performance. In case of user tasks, the implementation shows that batch activities have the advantage that task performers have several items consolidated in one work item improving their work efficiency, because they do not need to open several ones.

8.2 implementation of multi-process batching

Batch processing across multiple business processes, which was presented in Chapter 7, relies on annotated data in business processes and their object life cycles. Current standard BPMS do not support the handling of data objects during process execution [67]. Instead, process data is stored in some engine-internal data structures or process variables. The Camunda BPM platform is no exception in this regard. For the prototypical implementation, we use it to show that the concept introduced in Chapter 7 can be implemented in a standard BPMS without direct support for data objects. The architecture of the resulting extended *Camunda Engine* is shown in Figure 53.

Extended Camunda architecture with multi-process batch processing capabilities

While process models can be created using the *Camunda Process Modeler*, the object life cycles (OLCs) can be created in an extension of the so-called *Process Editor*[2] (cf. Figure 53). With this editor, it is possible to design OLCs, as well as *batch transitions* with additional data state transition information (i. e., a Boolean value specifying whether it is a batch transition, a set of Strings specifying the grouping character-

2 https://github.com/BP2014w1/processeditor

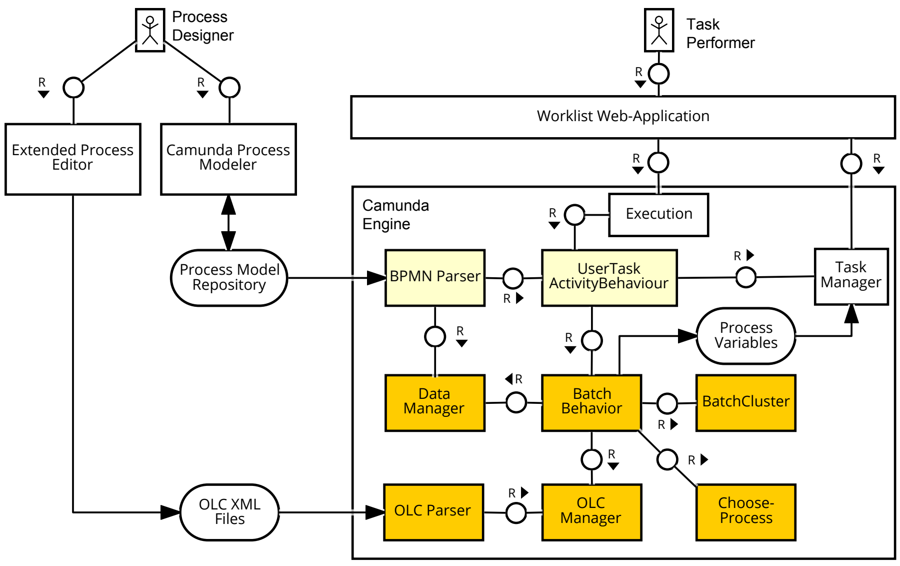


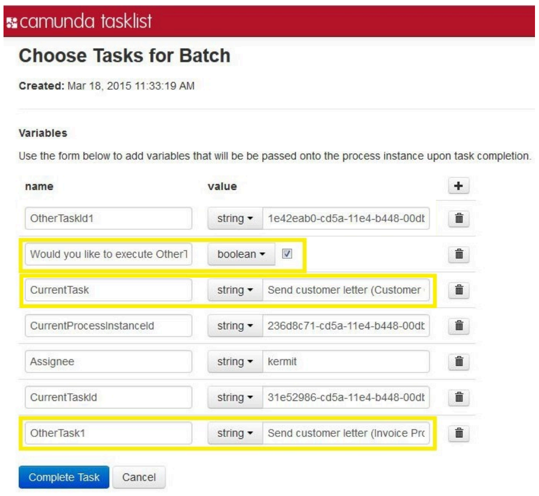
Figure 53: Architecture of the extended Camunda engine for enabling batch processing across multiple business processes (as FMC block diagram [47]): adapted components are shown in yellow and new components are shown in orange.

istic referencing process variables, and a set of connected transitions). The resulting OLC model can then be stored as XML file by the extended process editor. We added to the *Camunda Engine* an *OLCParser* to retrieve the OLC information from the generated XML files, and an *OLCManager* to access an OLC for a given data type.

Additionally to OLCs, the process designer can store multiple process models in a process model repository accessible by the *Camunda Engine*. We assume that the process models which should be involved in the batching are available in this repository. The *Camunda Engine* does not handle data nodes annotated in process models out of the box. Thus, the BPMN parser was extended to consider annotated data nodes and their data associations. A *DataManager* was added which allows to retrieve the input and output data nodes for a given process activity. In the remainder of this section, we detail our implementation on the example of user tasks; service and script tasks can be handled similarly with the assumption that the Java service or the script referenced by such tasks can handle aggregated input.

The *UserTaskActivityBehavior* class responsible for executing the behavior of a user task is extended in the following way: When a user task gets enabled, the central *BatchBehavior* class (cf. Figure 53) is called. A method of this class checks by using the *DataManager*, and the *OLCManager* the related data state transitions of the user task whether one of them is a batch transition. If yes, all currently enabled user task instances are retrieved. Based on the input and output data nodes of their related activities, it is checked which user tasks reference the same batch transition as the newly enabled one, or a connected batch transition. All positively identified user task instances are pre-selected. Next, the grouping characteristic of them is checked. Therefore, the values of the given process variables in the grouping characteristic are retrieved for the current user task. These are compared to the corresponding values for the pre-selected ones. A matching data view adds the corresponding user task to the set of selected instances. If some matching user task instances were found, the *BatchBehavior* calls another process deployed in the engine, the so-called *Choose-Process*. With this, it is possible to provide the identified similar user task instances to the task performer in a *Choose Tasks*-form as shown in Figure 54a. A *Choose Tasks*-form presents the just enabled user task (cf. *CurrentTask* in Figure 54a) and all selected ones (cf. *OtherTask* in Figure 54a) and asks which of the identified instances can be consolidated in a batch work item. It is shown to all task performers which are allowed to execute the just enabled user task.

Visualization of the user approval and the batch work item


camunda tasklist

Choose Tasks for Batch

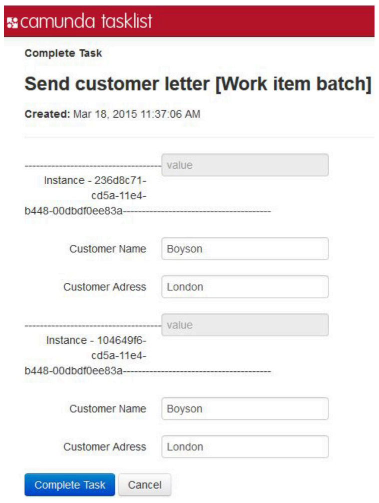
Created: Mar 18, 2015 11:33:19 AM

Variables

Use the form below to add variables that will be be passed onto the process instance upon task completion.

name		value
OtherTaskId1	string	1e42eab0-cd5a-11e4-b448-00dt
Would you like to execute Other1	boolean	☑
CurrentTask	string	Send customer letter (Customer
CurrentProcessInstanceId	string	236d8c71-cd5a-11e4-b448-00dt
Assignee	string	kermit
CurrentTaskId	string	31e52986-cd5a-11e4-b448-00dt
OtherTask1	string	Send customer letter (Invoice Prc

Complete Task Cancel


camunda tasklist

Complete Task

Send customer letter [Work item batch]

Created: Mar 18, 2015 11:37:06 AM

---------- Instance - 236d8c71-cd5a-11e4-b448-00dbdf0ee83a ----------	value
Customer Name	Boyson
Customer Adress	London
---------- Instance - 104649f6-cd5a-11e4-b448-00dbdf0ee83a ----------	value
Customer Name	Boyson
Customer Adress	London

Complete Task Cancel

(a) Choose-Tasks-form provided to the task performer to select activity instances for batch processing.

(b) Batch work item with the forms of all selected activity instances for user task *Send customer letter*.

Figure 54: Choose-Task-form and batch work item visualized in Camunda *Worklist Web-Application*.

If one or more user tasks are selected by one of the task performers, a work item batch is created. Thereby, the *BatchBehavior* disables the execution of all selected user task instances first. Then, it aggregates the data of all instances and continues the execution of the *UserTaskActivityBehavior* for the just enabled user task with the aggregated data. In this case, the *UserTaskActivityBehavior* provides a batch work item to the task performers in which the form variables of all instances are shown in one view (cf. Figure 54b) It can be terminated by one click on *Complete Task*. As soon as the batch work item is terminated, the

BatchBehavior terminates the other user task instances, and if necessary, adds updated data to them.

Summarized, we added a new OLC parser to the *Camunda Engine* and made small adaptation on the BPMN parser for parsing data objects to make the engine data-aware. With this, each activity can be checked whether it is a batch candidate. A central *BatchBehavior* class is responsible for this check, for the user approval, and for batch execution which can be reused for other activity types. Our implementation shows that batch work items can be created and executed, and several bones can also run in parallel. Further, it shows that if the Choose-Tasks-form is not yet executed, new user tasks can be added dynamically. Further, an existing batch work item can also be combined with a new work item, if a task performer approves it.

8.3 CONCLUSION

The proof-of-concept implementation showed that both concepts could be integrated in an existing BPMS with small extensions on it. Only a small set of components had to be adapted, e. g., the parser and a small set of constructs, e. g., the batch activity and the batch cluster had to be added. Although the BPMS does not consider data objects in business processes, the multi-process batching concept on OLC could be enabled. In case of user tasks, both implementations show that the visualization of several work items in one user view improves the work efficiency, because the task performer do not have to open each work item individually. In future, both prototypes could be integrated in one system.

9 APPLICATION TO USE CASES

This chapter presents the validation of the introduced batch activity concept. *Therefore, single-case mechanism experiments are used to study the effect of batch activities on two different use cases with the help of a simulation. Business process simulation (BPS) is an important mean for the quantitative process analysis. Based on this, we can predict how the batch activity concept can support business processes in future, and its influence on the process performance. For the BPS, an extensible BPMN process simulator is developed and extended by the functionality to simulate batch activities. The simulation results implicate cost reductions with a slightly positive influence on the cycle time, if a suitable batch activity is applied. The simulator can be also used by process designers in future to simulate their processes and to validate selected batch configurations.*

Single-case mechanism experiments

In this chapter, the *batch activity concept* is validated by applying it to use cases in single-case mechanism experiments. In design science, it is essential to predict the influence of an artifact and to justify its contribution to the stakeholder goals [133]. Therefore, single-case mechanism experiments are used to study the effects of the mechanisms of an artifact in interaction with its context in a laboratory environment [133]. For such experiments, a validation model is created with a model of the artifact and a model of the context which are fed with test scenarios to observe the results [133]. With such a validation model (1) possible effects of an artifact on its context, (2) the trade-off (i. e., comparison to alternative artifacts), and (3) the sensitivity of artifacts (i. e., comparison to an alternative context) can be studied [133].

In this thesis, single-case mechanism experiments are used to apply the batch activity concept to different use cases in a simulated environment to predict its effect on the real world. The following scenarios are used for the experiments:

- **An administration use case**: The process of this use case is about the relabeling of the nameplates of office rooms in an organization, e. g., in the case of new arriving employees. In this process, the clerk can combine the relabeling of several office rooms in order to save working time, thus labor costs. This scenario is used to validate the trade-off between having a batch activity versus no batch processing.
- **An online retailer use case**: This process is our running example introduced in Section 2.2. In this use case, batch processing can be used to combine several orders of the same customer during

packing and sending the parcel to save shipment costs. This scenario is used to compare the effect of different batch activation rules.

The main goal of batch activities is lower the process cost as well as the processing time as several instances are served in one batch. Nevertheless, batch activities can also increase the process cycle time as it requires certain time to fill the batch. Thus, we want to conduct a quantitative analysis of the selected business processes in order to compare the process cost and the cycle time with and without batch activities.

Queuing systems vs. simulation

In a previous work in [87], we studied the performance of a batch activity in comparison to a simple activity with the help of techniques from queuing theory. By using queuing systems, such as *M/M/1*[1] for simple activities and *M/M(a,b)/1*[2] for batch activities [63], the average cost and the average waiting time can be calculated. However, there are certain limitations with regards to queuing systems [87]. If alternative batch activation rules to the threshold rule (cf. Section 5.2) should be evaluated, the equations of the queuing system becomes more complicated, hardly applicable in practice. A fundamental limitation of queuing systems is that they focus on the evaluation of one activity at a time and not on the whole process. However, a batch activity can influence, for example, the instance-arrival at the subsequent activities which should be considered in the evaluation. Queuing networks can be utilized to extend the analysis to the whole process. However, those become quite complex [26]. An alternative method for measuring of the performance of business processes is the BPS.

Business process simulation (BPS) is a cost-efficient mean for quantitative analysis of business processes [120] which provides insights into throughput times, resource utilization, and process costs of different process alternatives [26]. Existing BPS tools do not support the batching of instances as identified in [120]; thus, an open-source BPMN process simulator was developed by Pufahl et al. [90]. This BPMN process simulator provides, in comparison to existing ones, an extensibility mechanism to enlarge the functionality of the simulator to new features by adding plug-ins.

In the remainder, this chapter is structured as follows: In Section 9.1, the extensible BPMN simulator is shortly presented including the *batch activity plug-in* which is used later to conduct different simulation experiments. Then, measures for validating and evaluating the performance of a batch activity are presented in Section 9.2, which are used

1 Queue system is given in the Kendall's notation [46] has a single task performer, where arrivals are determined by a Poisson process and service time has an exponential distribution.

2 Queue system given in the Kendall's notation [46] has a single task performer, where arrivals are determined by a Poisson process. With that, the service commences in batches, when the queue size reaches or exceeds a threshold value a, and the capacity of the service is b, so that $1 \leqslant a \leqslant b$. As well, the service time is independent of the batch size and has an exponential distribution.

for the single-case mechanism experiments. In Section 9.3, the single-case mechanism experiment with the administrative use case is presented and the simulation results with regards to no batching and with batching are presented and discussed. This is followed by the retailer use case in Section 9.3 to which different activation rules are applied. Finally, a summary of the results is given and discussed in Section 9.5.

9.1 EXTENSIBLE BPMN PROCESS SIMULATOR

BPS is an important means for quantitative analysis of business processes [120], but it is also used by researchers to evaluate new process modeling artifacts.

With the BPMN standard being the state-of-the-art language for the graphical representation of business processes, BPMS vendors already provide BPMN simulators (e. g., Bizagi Modeler, Trisotech Modeler, BonitaSoft, and Visual Paradigm [31]). With these simulators, a translation of a BPMN process diagram in a specific simulation language is not necessary anymore. However, the commercial as well as academic products, such as BIMP [1], are proprietary and do not support the extensibility by new BPMN constructs. Thus, CPN Tools [91] relying on colored petri nets (CPNs) is mainly applied for process simulation in research [44]. But CPN Tools requires a manual translation of the BPMN process diagrams into CPNs and expert knowledge to work with the tool. Therefore, we developed an extensible BPMN process simulator [90], called Scylla[3], which is used in this thesis to validate the batch activity concept.

Design of the extensible BPMN process simulator

The extensible BPMN process simulator builds on discrete event simulation (DES). With DES, a real-world process is captured as a finite set of events in time, i. e., each event occurs at a certain point in time and marks the change of the process state [4]. As no changes occur between events, the simulation jumps from one event to another, allowing DES to run fast and independently from real process time in contrast to continuous systems. Tumay [109] describes DES as the "most powerful and realistic tool for analyzing the performance of business processes", which provides "statistical input and output capabilities and advanced modeling elements [...]." The BPMN process simulator uses DESMO-J [33, 111] as DES framework which has not only gained wide acceptance in the academic community; it is also the foundation of many commercial simulation software. It provides blueprints for simulation models in which discrete events and entities can be defined. In the following, the plug-in structure as extensibility mechanism of the business process simulator is shortly presented followed by a description of the *batch activity plug-in*.

3 Its source code is available at https://github.com/bptlab/scylla.

PLUG-IN STRUCTURE. The simulator provides a designed plug-in structure consisting of several abstract classes which defines entry points into the simulation environment.The plug-in structure is shown in Figure 55 in which the so-called *pluggable*-classes are categorized into the different stages of simulation: parsing, initialization, execution, and reporting.

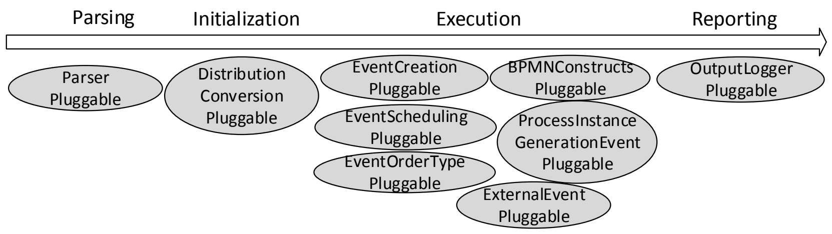


Figure 55: *Pluggable*-classes of the BPMN process simulator as entry points for writing plug-ins categorized into the different steps of a simulation.

The `ParserPluggable` offers an entry point to the input parsers, such that new simulation input, e. g., a new BPMN element, can be parsed. During initialization of the discrete simulation model, the DESMO-J distribution for the arrival rate and activity distributions are set. The entry point `DistributionConversionPluggable` allows the initialization of additional distributions. While executing a simulation experiment, events are generated, stored in queues, and if an event occurs, their event routines are executed which usually results in new events. Different entry points are available to influence DES events: `EventCreationPluggable` to generate new type of events, `EventSchedulingPluggable` to influence the scheduling of events, and `EventOrderTypePluggable` to adapt the priorities of events and to change their order in the queues. For influencing the implemented BPMN behavior of the simulator, two entry points exist: one on the process instance level – `ProcessInstanceGenerationEventPluggable`– and one on the BPMN events level – `BPMNConstructsPluggable`. The latter ones includes several sub-classes to influence the behavior of the minimum set of BPMN elements supported by the basic simulator, for instance, the `BPMNStartEventPluggable` or the `TaskEnableEventPluggable`. The `ExternalEventPluggable` offers the opportunity to add behavior which is not strictly related to a single process instance, but to the general behavior of business process simulation. Finally, a plug-in entry point is available to extend the simulator logs and output reports, the `OutputLoggerPluggable`.

BATCH ACTIVITY PLUG-IN. The extensible BPMN process simulator provides a plug-in for batch activities. For realizing a batch activity, five entry points were used. First of all, the `BatchParserPlugIn` extending the `ParserPluggable` is created to parse the extension elements used for a batch activity consisting of different batch configuration parameters, such as the batch activation rule. For a batch activity, the challenge is that the normal activity behavior has to be adapted. Activity instances are interrupted in their execution and only if a certain condition is fulfilled, a batch of instances is executed. Therefore, the `BatchTaskEnablePlugIn` extending the `TaskEnablePluggable` ensures that enabled batch activity instances are assigned to a batch cluster after activity enablement. In case a new batch cluster has to be created, a new type of event – a `BatchClusterStartEvent` – is prepared. It implements the blueprint of a DESMO-J discrete event with the batch cluster as its entity. Its event routine schedules all *task-begin-event* objects of the instances in the batch cluster. Additionally, it selects one process instance representing the batch execution. This representative process instance is executed as usual. For the remaining instances, the `BatchTaskBeginPlugIn` ensures that the *task-begin-event* routine is not executed, instead their *task-terminate-events* are scheduled. After the representative instance was executed, the `BatchTaskTerminatePlugIn` allows that the *task-terminate-events* of the others are activated and that resulted logging data of the representative ones are taken over by the others. Finally, the `BatchLogger` extending the `OutputLoggerPluggable` creates a report with batch-specific performance indicators, such as maximum and average waiting time of the batch clusters.

9.2 PERFORMANCE MEASURES FOR BATCH ACTIVITIES

Before presenting the single-case mechanism experiments, we want to define performance measures based on which the success of a batch activity can be measured. With regards to Dumas et al. [26], the performance measures for business processes are time, cost, quality, and flexibility.

Process cost and cycle time as batch activity performance measures

As discussed in the related work chapter in Section 3.1, batch processing can be used to lower the average execution costs per instance as several instances are served in one batch. However, the cycle time of an instance can increase by waiting for other instances to be batched with them. On the contrary, batching can also decrease processing times by handling a set of instances in one step. Therefore, we focus on performance measures with regards to cost and time on which we want to elaborate in the following:

- *Process cost*: Process costs are the costs associated to execute a process instance. There can be *variable costs*, which occur each time a process is executed, and *fixed costs*, which are overhead costs unaffected by the intensity of processing and have to be distributed over the process instances [24]. Serving several process instances in a batch can save *processing costs*, such as labor or material costs, but as well as *setup costs*, such as setting-up a machine or time needed to get familiar with a specific type of work [2, 107]. The size of batches influences the cost of process with a batch activity:

  - *Batch size*: The higher the number of instances in a batch, the lower are the average costs per process instance, as the costs of setting up and executing a batch are distributed over all instances. Therefore, the batch size can explain the cost per instance. Batch activation rules as introduced in Section 5.2 have an implication on the size of the batch cluster and can, thereby, influence the process cost.

- *Process cycle time*: Although the main goal of batch processing is to reduce process cost, it is equally important to measure the cycle time of the process instances influenced by a batch activity. The cycle time is the time to handle a process instance from the start to the end [26]. The goal of batch processing is to reduce the cycle time of process instances by handling several instances in a group [77], because the average time per instance needed for setups or familiarization can be reduced by distributing it over the group. However, process instances have to be interrupted in their execution at a batch activity to add them to a batch cluster and to fill the batch cluster until the batch activation rule is fulfilled [70]. In some use case, the cycle time might be not crucial and an increase of it is accepted, if the process costs can be significantly lowered. Still, it is important to analyze the success of a batch activity. Further measures are helpful to explain a certain cycle time.
  - *Turnaround time*: Turnaround time is defined [108] as the time needed from the moment of proving a job to a batch until returning the results of it. In the batch activity concept introduced in this thesis, an activity instance being enabled at a batch activity is immediately added to a cluster. Thus, we define the turnaround time as the time a process instance spends in a batch cluster, from adding it until the termination of the batch cluster.
  - *Waiting time*: The turnaround time also includes the waiting time of process instances until a batch is started. This is idle time for a process instance. We can differentiate between the waiting time until a batch cluster is enabled which is dependent on the design of the batch activation rule, and the waiting time until the batch cluster is started which is additionally dependent on the availability of the resource being responsible for the batch execution. Further, waiting time can occur at the activity succeeding the batch activity, because the batch activity releases groups of instances. These three different types of waiting times can help to identify a beneficial batch activity configuration and can also give implications on the overall process design, e. g., the number of resources of the activity succeeding a batch activity.

– *Proportion of reached due date*: In some business processes, due dates have to be fulfilled. Then, it is important to measure how often the due date is reached. A batch activity should be configured, such that the requested proportion of reached due dates can be fulfilled.

Further measures for batch activities, which can be investigated, but are not considered in this validation, are *throughput time* [108], and *resource utilization* [107] – both mainly used in the production and computer systems domain. *Throughput time* is the amount of work done per unit of time. It can be applied to analyze the performance of the resource allocated to a batch activity, or it is also helpful to compare different types of resources. If a resource has high fix costs (i. e., asset and maintenance costs), then the goal is to have a high utilization of the resource which can be measured by the *resource utilization*.

9.3 APPLICATION TO AN ADMINISTRATIVE PROCESS

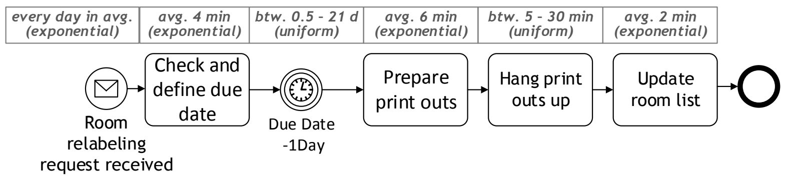
In this section, we present an application of the *batch activity concept* to an administration process in a single-case mechanism experiment. For validating the concept before an application in real life, the batch activity is used in a simulation of the process. The process itself is about the relabeling of the nameplates of office rooms in an organization, for instance, in case of new arriving employees. Batch processing can be used in this process to decrease the working time of the responsible clerk by combining the relabeling of several office rooms. In the following, the setup of the simulation experiments is explained, and then the results of them are presented and discussed.

SETUP OF SIMULATION EXPERIMENTS. Based on interviews with the expert, we design the process as shown in Figure 56a. As soon as a request for relabeling is received, the clerk first checks it and identifies the due date when the nameplate has to be changed. One day before, the clerk usually prints the office labels and goes to the respective office to attach it to the office door. Afterwards, the internal room list is updated.

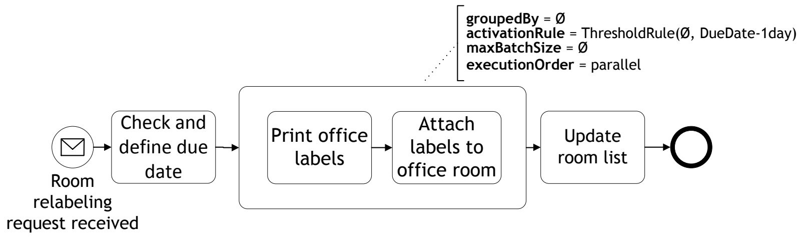
With respect to the proposed concept of batching, the clerk can print the labels, and attach them for several requests together as shown in Figure 56b. Thereby, working time can be saved, because the printer has to be only set up once, the clerk only has to walk around once for a group of requests, etc. In the current process, the clerk sometimes manually batches re-labeling requests. This is done rather randomly and does not follow specific rules; therefore it is not considered in the simulation.

Simulation experiments for the administrative use case

We define two different simulation experiments, each simulating 150 instances (i. e., simulation of half a year, because in average one request per day arrives), to compare the performance of the process in case of (1)


every day in avg. (exponential)
avg. 4 min (exponential)
btw. 0.5 - 21 d (uniform)
avg. 6 min (exponential)
btw. 5 - 30 min (uniform)
avg. 2 min (exponential)
Room relabeling request received
Check and define due date
Due Date -1Day
Prepare print outs
Hang print outs up
Update room list

(a) Current process with simulation parameters.


groupedBy = Ø
activationRule = ThresholdRule(Ø, DueDate-1day)
maxBatchSize = Ø
executionOrder = parallel
Room relabeling request received
Check and define due date
Print office labels
Attach labels to office room
Update room list

(b) Process with batch activity.

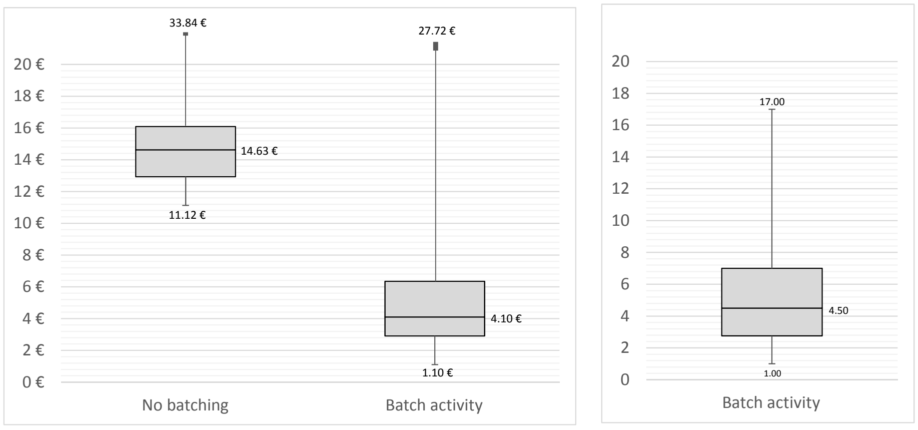
Figure 56: Application of batch activity concept to an office room relabeling process.

no batching, and (2) applying the batch activity. The two experiments are described in more detail in the following:

1. *No batching*: For this case, the process model as given in Figure 56a is used. Further, the simulation parameters discussed with the process expert are used for the simulation, which are visualized in Figure 56a. These parameters are explained in the following in more detail. Once a day a request for a re-labeling is received, sometimes the inter-arrival time is higher, therefore an exponential distribution with a mean value of one day is selected for the inter-arrival time of instances. The request is then checked mostly in four minutes. For this task duration, an exponential distribution is selected, because sometimes additional inquiries are necessary for a few requests leading to a longer task duration. In some cases, requests are arriving short-termed, and in other cases, they are handed in much in advanced. Therefore, we assume together with the process expert a uniform distribution for the timer event where all values between 0.5 and 21 days are equally possible. For preparing the label, the clerk has to feed special paper into the printer, and then print the prepared forms. This takes usually around six minutes, and in case there is a issue with the printer, sometimes longer, leading to an exponential distribution with a mean of 6 minutes. For attaching the label a uniform distribution with values between 5 to 30 minutes was selected as the clerk not only has to go to offices in their own building, but they also have to walk to other office buildings. Finally, the room list is updated, usually taking around two minutes. Additionally, it was considered in the simulation that the clerk responsible for this process works on it two hours in the morning from Monday to Friday.

2. *Batch activity*: For the application of the batch activity concept, we use the batch configuration depicted in Figure 56b. The batch activity sub-process includes the printing of office labels, and hanging them up. Updating the room list could be also included in the batch activity, but we exclude it in order to observe the waiting time at it as subsequent activity to the batch activity. For this process, no grouping of instances is necessary, and the maximal batch size is unlimited. The activation rule is a threshold rule with a due date, such that a batch cluster is activated one day before the earliest due date of its included instances. We selected *parallel* processing. Although the clerk executes the requests sequentially in any order, all cluster items are shown at once to the task performer in one batch work item. For the simulation, the process model as given in Figure 56b is used, and the inter-arrival time of instances and activity duration as visualized in Figure 56a are re-used to allow a comparison.

SIMULATION RESULTS. Running the simulation experiments resulted in different activity event logs based on which the relevant performance measures are calculated, such as *process cost*, *batch size*, *cycle time*, *proportion of reached due date*, and *waiting time* at the subsequent activity. *Turnaround time* and *waiting time* at the batch activity as proposed in the previous section are not so relevant for this use case, because the most important aspect is to reach the due date. The simulation files, the activity event logs and the calculation are available at http://bpt.hpi.uni-potsdam.de/Public/BatchProcessing.



(a) Process cost. (b) Batch size.

Figure 57: Simulation results regarding *process costs* and *batch size* for the administrative use case shown as *box plot* diagrams visualizing the minimum value, the lower quartile, the median, the upper quartile, and the maximum value of the data.

Results regarding process cost

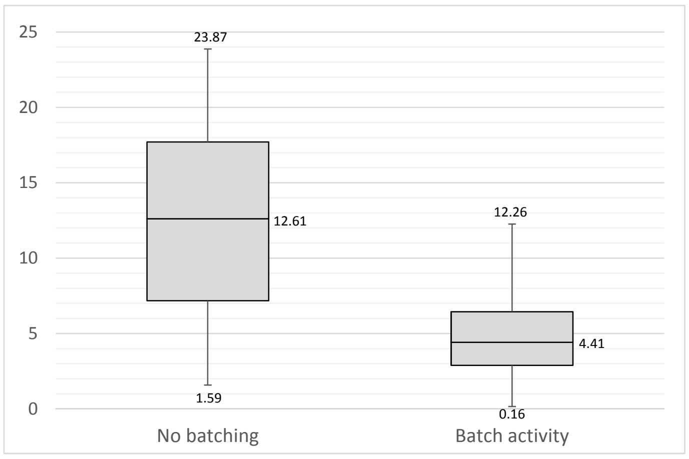
First, the process cost are presented. Thereby, we assume that the clerk has an hourly wage of 30 €. For the calculation of the costs, the actual working time of the clerk (i. e., the sum of all activity durations without waiting time) per instance is multiplied by their wage. In case of the batch activity, the activity duration of the two activities *Prepare print outs* and *Hang print outs up* are divided by the number of instances of the batch cluster. The results are shown in Figure 57a as box plot diagrams. Box plot diagrams [62] are means to represent numerical data through their quartiles; thereby, the maximum value, the minimum value, the upper and lower quartile, as well as the mean value are shown. In case of no batching, most cases have costs between 12.93 € to 16.09 €, whereas, in case of a batch activity, many cases have costs between 2, 90 € to 6.35 €. This means that the batch activity reduces the process costs by around 72% (i. e., around 10.00 € per process instances). The maximum value of 27.72 € the experiment with the *Batch activity* is similar to the maximum value of 33.84 € in the experiment with *No batching*, because some batch clusters consist only of one instance. Figure 57b shows the batch size; in most cases, the size of a batch cluster is between 2.75 to 7 instances. At maximum, also a batch cluster with 17 instances can be observed.

Results regarding cycle time

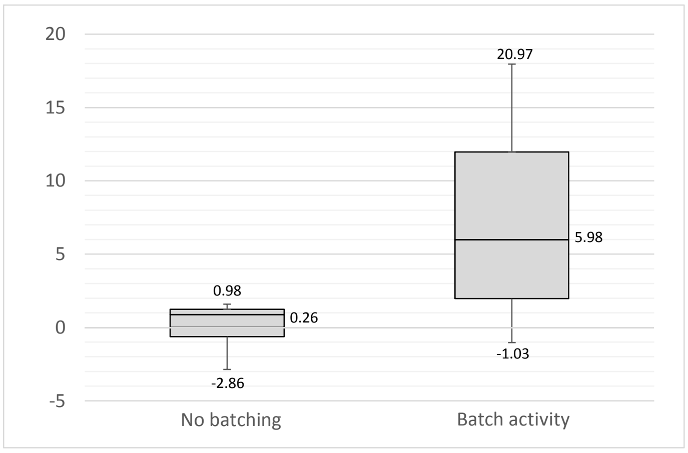
The results regarding the cycle time measuring the time from receiving a request until the end are shown in Figure 58a. In case of no batching, process instances are handled in 7.17 to 17.70 days which includes also the waiting time until the due date is reached (cf. timer event in Figure 56a). If a batch activity is applied with the proposed configuration, most of the cases are then handled in between 2.89 to 6.45 days. By taking the mean value, the cycle time is reduced by 65% in the case of applying a batch activity.

Additionally, it is measured how many days before the actual due date the office room is relabeled. As shown in Figure 58b, if no batching is applied, most requests are fulfilled -0.62 to 0.35 days before the due date. This means that most of the cases are finished exactly on the due date, sometimes half a day later which is still acceptable. In 26 cases out of the 150 cases (17.3%), the due date is not reached by one or two days (cf. minimum value). In 22 of these cases, the due dates are at the weekend where the clerk is not available, but the re-labeling is finished on the next Monday, such that they are still on time. However, 4 cases (2.7%) do not meet the due date in case of no batching. If a batch activity is applied, most requests are fulfilled earlier, between 1.97 to 11.97 days in advance. In 4 cases (2,7%), the due date was not reached by one day (cf. minimum value), but their due date is dated on a weekend. Thus, no case is finished late in the case of a batch activity. Finally, the waiting time at the subsequent activity of the batch activity (cf. *Update room list* in Figure 56a) is on average 4:14 minutes with a median value of 3:18 minutes. This is, compared to a cycle time of several days, very low.

DISCUSSION. The results of the simulation experiments show significant process costs reduction by around 72%, if a batch activity is ap-



(a) Cycle time.



(b) Amount of days the re-labeling is finished before due date.

Figure 58: Simulation results regarding cycle time for the administrative use case shown as box plot diagrams.

plied. For the real-world process, the cost reduction might be a little less, because batching is already randomly applied. However, if batch processing is regularly used based on specific rules, a significant improvement of the process cost is still expected.

The simulation experiments show that batch processing can reduce the cycle time of the cases, because the waiting time until the due date is actively used to handle requests together with others. This leads to the situation that the relabeling is terminated in advanced for most of the cases (mostly between 2 to 12 days in advance). In this presented administrative use case, the process expert does not consider the advanced termination as an issue. However, if it is desired that the request should be finished less days before the due date, two counteractions might be possible. On the one hand, the batch configuration could be adapted, for example, the *groupedBy*-parameter could be used to group requests having the due date in the same work week. Then, the requests would be finished only a few days before the due dates. On the other hand, the user involvement strategy allowing the task performers to reassign instances to other clusters as presented in Section 5.4 could be activated. With this, the task performers could reassign requests, whose due date is later than the others, to a new batch cluster. Both counteraction would lead to a smaller average batch size, such that cost benefits might be less as in the current setting.

In this use case, the waiting time of the subsequent activity to the batch activity is quite low, because its task duration is low and its allocated resource, the clerk, has also a low resource utilization, also due to batch processing. This might be different in other use cases.

Summarized, we conclude based on the insight of the simulation experiment that batch processing applied in a regular manner can offer the potential to reduce processing time in this relabeling process, and thereby, lead to process cost reductions. However, it needs to be mentioned that the reduction of labor cost is only valuable, if the labor can be allocated to other relevant tasks for an organization.

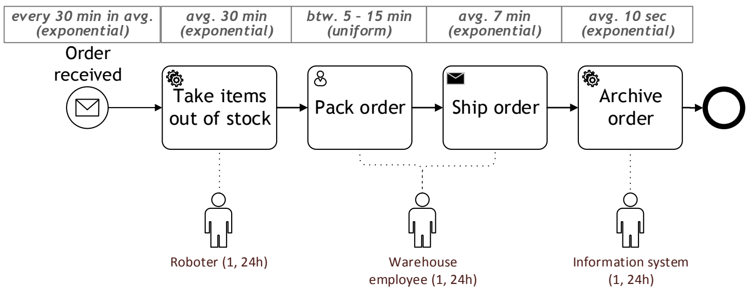
9.4 APPLICATION TO AN ONLINE RETAILER PROCESS

In this section, the batch activity is applied to the running example of this thesis – the *Online retailer* process – also in a simulated environment. As described in Section 5.1, batch processing can be used in this process to consolidate orders of the same customers during the packing- and sending-activity in order to save shipment cost. Thereby, we want to investigate different batch activation rules, on the one hand the *ThresholdRule* and on the other hand the *MinMaxRule* (cf. Section 5.2). In the following, the setup of the simulation experiments is explain, and then the results of them are presented and discussed.

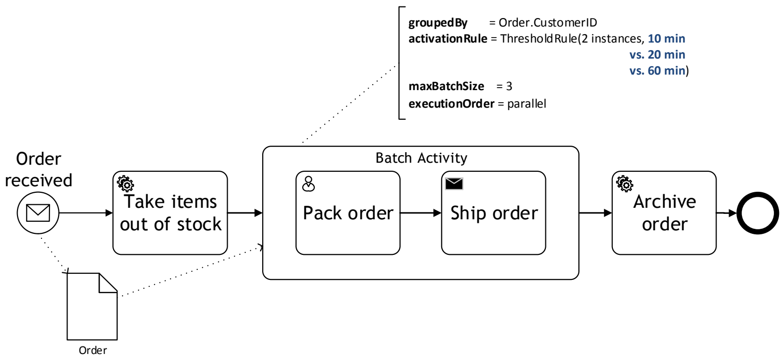
Simulation experiments for the retailer use case

SETUP OF SIMULATION EXPERIMENTS. We define three different type of simulation experiments to compare the process performance in case of (1) no batching, (2) a batch activity with a *ThresholdRule*, and (3) a batch activity with a *MinMaxRule*. These types of experiments are used for running several experiments to analyze different configurations. Each experiment simulates 15,000 instances (i. e., simulation of one month with around 50 orders arriving each day). The three types of experiments are described in the following in more detail:

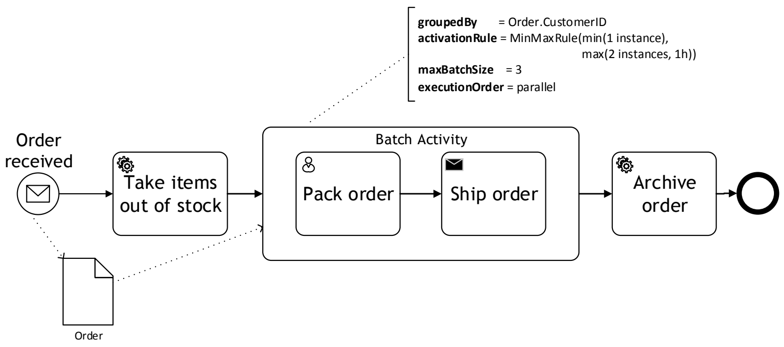
1. *No batching*: The simulation focuses on the activities happening in the warehouse of the retailer as shown in Figure 59a which also visualizes the simulation parameters. We assume a mid-size retailer which receives in average every 30 minutes an order, sometimes the inter-arrival time is higher. Therefore, an exponential distribution with a mean value of 30 minutes is selected for the inter-arrival time of instances. At first, a robot, available 24h a day, takes the items of an order out of the stock and delivers them for packing. This takes in the most cases 30 minutes. In


every 30 min in avg. (exponential)
avg. 30 min (exponential)
btw. 5 - 15 min (uniform)
avg. 7 min (exponential)
avg. 10 sec (exponential)
Order received
Take items out of stock
Pack order
Ship order
Archive order
Roboter (1, 24h)
Warehouse employee (1, 24h)
Information system (1, 24h)

(a) Current process with simulation parameters.


groupedBy = Order.CustomerID
activationRule = ThresholdRule(2 instances, 10 min vs. 20 min vs. 60 min)
maxBatchSize = 3
executionOrder = parallel
Batch Activity
Order received
Take items out of stock
Pack order
Ship order
Archive order
Order

(b) Process with a batch activity having a *ThresholdRule* with different time-outs.


groupedBy = Order.CustomerID
activationRule = MinMaxRule(min(1 instance), max(2 instances, 1h))
maxBatchSize = 3
executionOrder = parallel
Batch Activity
Order received
Take items out of stock
Pack order
Ship order
Archive order
Order

(c) Process with a batch activity having a *MinMaxRule*.

Figure 59: Application of batch activity concept to the online retailer process.

rare cases, it takes longer, if an issue occurs, such that an exponential distribution is chosen. A warehouse employee packs the items of an order in a parcel and prepares it for the shipment. The actual shipment of the order is not simulated, because this is performed by an external logistic provider. It is assumed that the warehouse employees work in shifts, such that 24h one warehouse employee is available. The duration of packaging is depending on the parcel size which can take between 2 to 15 minutes uniformly distributed. For preparing the shipment, an exponential distribution with a mean value of 7 minutes is selected. Finally, an order is archived by an information system which needs usually 10 seconds. It might take longer in some rare cases, if its utilization is high due to the usage by other departments. Hence, an exponential distribution with a mean value of 10 seconds is used.

2. *Batch activity - ThresholdRule*: For the simulation experiments with the batch activity, the process model depicted in Figure 59b is utilized. In this process model, a batch activity surrounding the activities *Pack order* and *Ship order* is added. For batching orders by the same customer, the *groupedBy*-parameter of the batch activity makes sure that only orders with the same `Order.CustomerID` are added to the same batch cluster. The presented BPMN process simulator in Section 9.1 is also able to simulate the creation of data objects consisting of several data attributes. For each given data attribute, the type and a distribution has to be selected for simulating different values. Thus, an *Order*-data object including `CustomerID` as data attribute is simulated. In the current simulation setup, the orders are handled within a day and around 50 orders are arriving each day. Therefore, a discrete distribution for the `CustomerID` over 50 different `IDs` is selected where each has proportion of 0.02 to be selected. Further, a threshold rule is installed as batch activation which is triggered, if at least two process instances are added to a cluster or a certain timeout is met. It is assumed that most customers only send a second order within one day, not a third one. Thus, a threshold of two instances is picked in this use case, but we want to study the effect of different timeouts. We selected 60 minutes, 30 minutes, and 10 minutes, because we assumed that an order should not be interrupted for more than an hour. The orders are handled by the batch activity in parallel. Additional to the process model, the inter-arrival time of instances and activity duration as visualized in Figure 59a are re-used for the simulation to allow a comparison.

3. *Batch activity - MinMaxRule*: This simulation experiment is similar to the last one, only the batch activation rule of the batch activity is adapted. The threshold rule has the disadvantage that a batch cluster waits the maximum waiting time also if no other potential instance for the batch cluster exists. The proposed *MinMaxRule* in Section 5.2 tries to avoid this by observing the instances which might arrive in future at the batch activity. In this simulation experiment, a *MinMaxRule* as depicted in Figure 59b is used which is activated when a cluster has at least one instance (minimum condition). If future arriving instances with the same data view are observed, then it is activated in case of two instances or the timeout of one hour has reached (maximum condition), because we want to make sure that a cluster waits enough time for the future arriving instance.

SIMULATION RESULTS. The simulation reports of five different simulation experiments are used to calculate process cost and cycle time. The simulation files, the simulation reports and the calculation are available at http://bpt.hpi.uni-potsdam.de/Public/BatchProcessing.

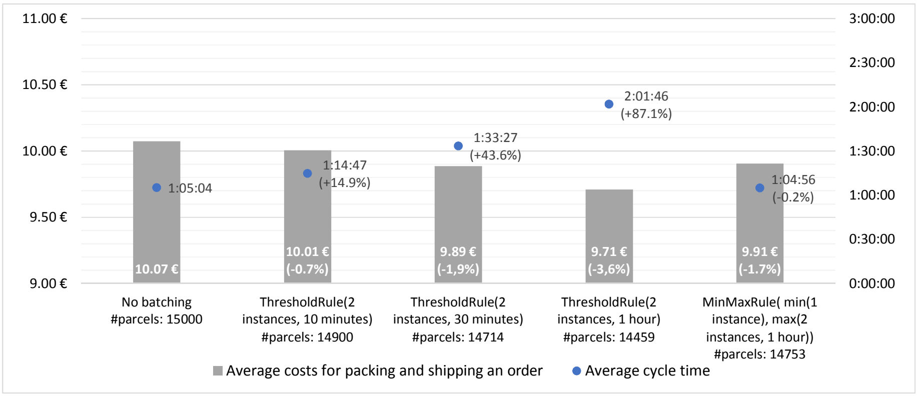


Figure 60: Simulation results for the retailer use case showing the average activity costs of packing and shipping as well as the average cycle time per instance for the case of (1) no batching, (2) a batch activity with a *ThresholdRule* with different timeouts, (2) a batch activity with a *MinMaxRule*.

In Figure 60, the *average costs* (depicted as balk) for the *Pack order* and *Ship order* activity as well as the *average cycle time* (depicted as measuring point) resulting from the different simulation experiments are shown. For the calculation of the activity costs, an hourly wage for the warehouse employees of 25 €, and shipment costs of 3 € per parcel are assumed. In the case of no batching, a parcel is packed and shipped for each received order to the customer, in total 15,000 parcels (cf. X-axis in Figure 60). In this case, the costs for an order are in average 10.07 € and the cycle time is in average 1:05 hours.

Results regarding cost and time

By applying a batch activity with a *ThresholdRule(2 instances, 10 minutes)*, the average costs can be slightly decreased by 0.7% to 10.01 €, because 100 parcels less are sent, in total 14900. The average cycle time increases by a few minutes to 1:15 hours. In Figure 61, the average waiting times of a batch cluster until its enablement are depicted for the different experiments with a batch activity. It shows an waiting time of 09:58 for this threshold rule. Hence, the increase in cycle time is mainly due to the specified maximum waiting time in the activation rule. Almost all instances have to wait additional 10 minutes; only a few less which can be batched with another instance.

With a *ThresholdRule(2 instances, 30 minutes)*, the average costs can be decreased to 9.89 € (−1.9%). With this batch activity configuration, the online retailer can save 2,813 € in one month (handling 15000 orders). On the contrary, the cycle time is increased by 43.6% to 1:33 hours, also mainly due to the waiting time until enablement of a batch cluster (cf. Figure 61). For the third batch activity configuration, the average costs can be decreased to 9.71 € (−3.6%). Here, the online retailer can save 5, 448 € in a month. However, the cycle time is increased by 87.1% to 2:02 hours.

In contrast, the *MinMaxRule* can decrease the average costs by −1.7% to 9.91 € (similar to the case of *ThresholdRule(2 instances, 30 minutes)*) with also a minor positive effect on the cycle time, a reduction of −0.2%. Reasons for this are that average waiting time at the batch activity is only 00:53 minutes, and the average waiting time at the pack and send order activities is decreased by few seconds, because 247 process instances are handled in a batch.

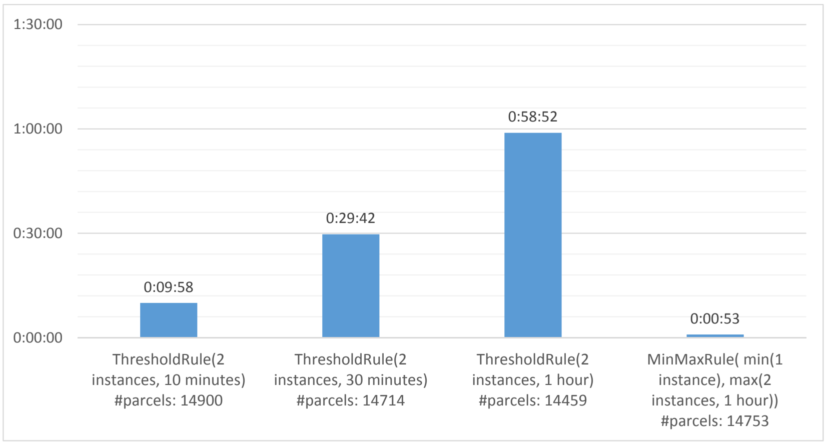


Figure 61: Simulation results regarding the average waiting time until enablement of a batch cluster for the simulation experiments with a batc activity.

DISCUSSION. This use case demonstrates the effect of different batch activation rules. Based on the simulation results, the online retailer can reduce the cost of packing and shipping a parcel by the *ThresholdRule*, if an increase of the cycle time is accepted. In case of a threshold rule with a timeout of one hour, process cost reduction of 5, 448 € per month can be achieved, if an increase of the average cycle time of one hour is acceptable. In this case, it seems beneficial to include the *Take items out of stock*-activity in the batch activity to avoid any in-process inventories.

If no increase in the cycle time is accepted, then the proposed *MinMaxRule* outperforms the *ThresholdRule* suggested in queuing theory literature. It offers a cost reduction of 2, 521 € per month, and a slightly improved cycle time. For comparing the success of different activation rules, the increase in cycle time needs to be associated with certain costs, loss in future sales [18] or in-process inventory cost, as we have discussed in [87]. This can help to identify the most beneficial activation rule based on the notion of cost.

9.5 CONCLUSION

In this chapter, the results of the validation of the *batch activity concept* were presented. Two different use cases - an administrative use case and an online retailer use case - were simulated to compare their process performance regarding time and cost in the case of batching and no batching. Therefore, an extensible BPMN process simulator was developed which supports the simulation of batch activities. This simulator can be used by process designers to validate the configuration of a batch activity. In future, we want to extend the simulator that it also supports the identification of an optimal batch activation rule for a business process based on the notion of costs. Therefore, process analysts have to provide additionally to the resource and material cost of a batch activity also the cost of waiting.

For the validation, performance measures for a batch activity were introduced with regards to cost and time. In future, these can be extended to quality and flexibility performance measures, from which we abstract in this thesis.

Both use cases showed that a trade-off between time and costs have to be solved when applying a batch activity. Depending on the specific goals in a process, a suitable batch configuration have to be identified. For example in the first use case, the re-labeling of nameplates at office rooms, labor costs could be significantly reduced by handling several requests in a batch. On the contrary, many requests were finished several days in advanced. If this should be avoided, a different batch activation rule as the proposed one needs to be selected. In the online-retailer process, the newly introduced *MinMaxRule* in this thesis offers cost reduction with no increase in cycle time outperforming the *ThresholdRule* proposed by queuing theory [70]. However, if an increase in cycle time is accepted, further cost savings can be reached with a *ThresholdRule*, or a differently configured *MinMaxRule*.

Currently, simulation models of the business processes and a model of the artifact (the *batch activity plug-in* of the simulator) were used for the validation with certain assumptions. For instance, we assume that the clerk in the administrative process is available each working day which is different in real life due to vacation and illness. Based on this, the effects of a batch activity can only be predicted. However, the results of the simulation experiments demonstrate that a batch activity can offer cost savings, if a suitable batch activation rule for the respective use case is selected. In future, the batch activity applied in real world use cases should be evaluated.

10 CONCLUSIONS

This chapter provides a summary of the main results of this thesis and concludes the work. First, the main contributions, i. e., (1) the requirements framework, (2) the batch activity *concept, (3) the integration of flexibility in batch activities, (4) the multi-process batch processing concept, are summarized. Then, the chapter provides a discussion of each contribution and its limitations. Based on this, the options for future research are discussed with regards to batch processing in business processes.*

Organizations use business process models to document, analyze, and redesign their conducted work, but also to enact and monitor their business processes. *Batch processing*, where groups of process instances (i. e., the actual executions of a business process) are collected for specific process activities and are executed collectively in order to save cost or time, is present in many real life business processes. In this thesis, it was shown that batch processing requirements could not be captured neither on the process design level (i. e., by a process modeling language), nor on the process execution level (i. e., by a business process management system (BPMS)). Manually executed batch activities or hard-coded solutions have the disadvantages that the batch processing rules are not traceable for the stakeholders, cannot be easily validated or improved, and cannot be easily monitored. In this thesis, we introduced a concept to capture batch activities explicitly in process models – not hidden on the implementation level. It also provides an execution semantics to automatically execute batch activities in a BPMS. This concept was extended to consider flexibility aspects during batch execution and to allow batch processing over multiple business processes; both research direction have not been discussed in related work, so far.

In the remainder of this chapter the main results are presented in Section 10.1. The results and their limitations are discussed in Section 10.2. In Section 10.3, areas of future work are presented based on the discussed limitations.

10.1 SUMMARY OF THESIS RESULTS

In the related work provided in Chapter 3, it was observed that batch processing is considered in other domains, such as computer science and operations management. Whereas batch processing in computer science is used to efficiently process large amount of data with no user interaction where the design and implementation of such systems is discussed, in operations management batch processing is used to process similar products or groups of customers for being more efficient where the balance between reduced costs and increase in cycle time is studied. Only a small set of research works in the Business Process Management (BPM) domain exists on the integration of batch processing in business process models by explicitly representing batch activities. The presented solutions in those works focuses on a specific scenarios and lack on a complete understanding of requirements. Therefore, the work of this thesis has started in Chapter 4 with a requirements analysis for integrating batch processing in process models. Based on the resulting requirements framework, the design objectives were set. The thesis provided in Chapter 5 a new modeling element – the *batch activity* with several configuration parameters – and explains how the batch activity can be automatically executed. The feasibility of the batch activity concept is shown in Chapter 8 by a proof-of-concept implementation in an existing BPMS. An application of it to different use cases in a simulated environment in Chapter 9 implied process cost reduction for business processes, if a suitable batch configuration is selected. Aspects of flexibility during the batch execution by different means (i. e., flexible design of the batch activation rule, user involvement strategies, and flexible batch configuration based on events) were presented and discussed in Chapter 5 and Chapter 6. Additionally, the basic concept was extended to batch processing across multiple different process models in Chapter 7 which was also evaluated by a proof-of-concept implementation in Chapter 8. In particular, the results of this thesis can be summarized as follows:

I. *Requirements Framework*: In this thesis, a requirements framework for integrating batch processing in business processes was presented. It was developed based on related work and complemented by requirements from collected industry examples, taken from different domains. The requirements framework gives insight into the aspects which need to be considered for developing a batch processing concept for business process models. Additionally, it fosters also the comparison of existing solutions. In this thesis, the requirements framework was used to structurally compare the requirements of the collected real world scenarios whereby two preliminary types of batch activities – *automated* batch activities and *user-involved* batch activities – were identified. Further, it was used to set our design objectives, and to compare the developed *batch activity* concept to other related work.

II. *Batch Activity Concept*: The main result of this thesis is the *batch activity* concept. The concept describes the syntax of a batch activity with its batch configuration parameters which need to be specified by the process designer. Specifically, we present in details the *groupedBy*-parameter for grouping instances in specific batches based on their data context with so-called *data views* and the *batch activation rule* being responsible for balancing the cost reductions with additional waiting time. Different types of activation rules were formalized, such as the threshold rule identified in operations management of which extensions were developed, e. g., the *MinMaxRule* to consider future instances. Further, it gives an execution semantics of the batch activity. This describes the life cycle of batch clusters (i. e., the actual representations of batch executions) and the interaction of batch clusters with process instances and the activity resources (i. e., task performers or services). The feasibility of the concept was shown by a prototypical implementation in an existing, open-source BPMS. The application of the batch activity to different use cases shows that process cost can be reduced with acceptable or even positive influence on the cycle time, if a suitable batch activation rule is selected. For supporting automated batch activities as well as user-involved batch activities, different levels of user involvement and the way they might be realized were presented in this thesis.

III. *Integration of Flexibility in Batch Activities*: This thesis showed that flexibility aspects of batch activities are not discussed by existing related work in the BPM domain. In the work of this thesis, flexibility for batch activities was provided by different means: First of all, batch activation rules, which are using Event-Condition-Action (ECA)-rules, can be flexibly designed, such that special cases that have a need for immediate execution can be considered. The thesis presented here the so-called *FastTrackRule*. Further, the presented user involvement strategies allow task performers to react dynamically on changes and exceptions in the process environment (e. g., by starting batch clusters when needed, re-assigning instances to another batch cluster, or waiting for specific future instances). Finally, this thesis presents a concept to integrate event processing techniques for a flexible batch configuration. In this, batch adjustment rules are defined by the process designer which specify for which event type which type of batch clusters need to be adapted and how. Thereby, an analysis was also provided on how events can change the batch configuration parameters. The application of the batch adjustment rules to a healthcare scenario in a simulated environment showed that they help to compensate the losses caused by the exceptional behavior in this use case.

IV. *Multi-process Batch Processing Concept*: This thesis could show that batch processing across several different business processes is useful is certain use cases and was not yet discussed by existing related work. Therefore, the thesis provide a requirements analysis for the multi-process setting based on a motivating example. The presented multi-process batch processing concept allows a centrally defined batch specification in an object life cycle (OLC). OLCs complement process models and describe allowed actions of business processes on data artifacts across the process-model boundaries. The basic concept of *batch transitions* was additionally extended in this thesis to multiple connected batch transitions (to allow also batching in process fragments) and to multiple similar batch transitions (to allow batching of activities having different data inputs but producing the same output). The requirement of *optional* batch processing in a multi-process setting was enabled by activating the batch processing only when similar activity instances are detected. Further, the concept includes a user approval where the task performer has to accept identified batches by the system. The feasibility of the concept was shown by a prototypical implementation in an existing open-source BPMS.

10.2 LIMITATION AND DISCUSSION

REQUIREMENTS FRAMEWORK. In this thesis, the requirements framework supported the definition of the design objectives as well as the comparison of the developed batch activity concept with other related work. However, currently none of the existing batch processing solutions proposed by the BPM research is actively used in practice. Hence, the framework might get more detailed or extended with the application of the solutions in future. Especially, the information for multi-process setting were restricted to one collected batch processing scenario having involved several process models. But also the information on the flexibility needs were limited, because many of the collected scenarios for the requirements analysis are currently executed manually. Thereby, flexibility is always possible and, thus, practitioners do not foresee which requirements regarding flexibility they might have.

So far, the requirements framework was validated with a group of BPM experts. Further validations with practitioners or BPMS providers might be needed. By applying the framework to structurally compare the requirements of the collected real world scenarios, we identified two type of batch activities – automated batch activities and *user-involved* batch activities. More use cases should be investigated to validate these types.

BATCH ACTIVITY CONCEPT. With the batch activity concept, we targeted *O1 Usability*, *O2 Usefulness*, and *O3 Generalization*. Next, we want to discuss the concept with regard to these objectives:

- *O1 Usability*: The batch activity is new type of activity with a list of configuration parameters. It was design in the same manner as existing process modeling elements. With the configuration parameters of a batch activity, a process designer is able to specify the batch execution: which instances are grouped in a batch (*groupedBy*), when a batch is started (*activationRule*), how many instances are allowed at maximum in a batch (*maxBatchSize*) and how the batch is executed, either parallel or sequential (*executionOrder*). Thereby, it is assumed that process designers are able to fill the configuration parameters based on expert knowledge. The latter two parameters, *maxBatchSize* and *executionOrder*, are mainly dependent on the resource which handles the batch cluster. The *groupedBy*-parameter depends on the type of cases being processed in batches whether they need to be grouped for batching or not. Here, a static grouping concept based on the process instance data was used, because we observed in the collected use cases that the grouping parameters are usually known before, e. g., by customer, by country, by responsible employee. A dynamic grouping concept would have the advantages that no grouping parameter needs to be pre-specified and the way of grouping can also change during the execution. It is especially helpful for use cases where batching is used to share the same setup (e. g., a familiarization phase) between process instances, because a dynamical grouping can identify most optimal groups. However, dynamically grouping requires a clustering algorithm which leads to meaningful results for the respective use case. Finding the most useful clustering algorithm and its correct specification is realized with high effort. The concept of activation rules allows to flexible design different types of rules. As it is a technical concept which uses ECA-rules, we proposed that BPMS vendors design the activation rules and offer them to the practitioners. A set of batch activation rule types were given in this thesis as examples based on insights of the queuing theory. Then, process designers only have to provide the requested input to the configuration parameters. Summarized, we believe, based on the given discussion, that practitioners can quickly apply the batch activity concept. However, the usability of the batch activity was not yet studied in a user study. If batch activities were already manually executed and process logs exist on those executions, batch processing discovery techniques as proposed by Liu et al. [128] and Martin et al. [58] can also be used to solve the design problem and to identify the batch configuration. However, these are first attempts in this direction and need to be further researched to discover a batch activity and its configuration.

- *O2 Usefulness*: In practice, every process is different and has unique characteristics. The batch activity, therefore, represents a concept that can be adapted to the specific circumstances for improving the process performance. Especially, the activation rule of the batch activity provides a mean to configure batch execution, such that cost benefits can be achieved without major influence on time and quality of the process. The application to use cases showed that a suitable configuration can lead to success, process cost reduction with an acceptable increase in waiting time, whereas non-suitable configurations can lead to not wanted behavior. Usually practitioners build on their working experience and are able to define a correct configuration. Queuing theory can help to identify for the batch activation rule, the most optimal threshold [63]. Additional to that, this thesis presented a BPMN process simulator extended with batch activities which can be used to validate a batch configuration.

- *O3 Generalization*: In this thesis, it was shown that the batch activity concept can be integrated in existing process modeling languages on the example of the Business Process Model and Notation (BPMN) notation dominating the standard space. Additionally, an integration to Unified Modeling Language (UML) Activity Diagrams and Event-driven Process Chains (EPCs) was also discussed in this thesis. Concretely in BPMN, the batch activity is a user task, a service task or a sub-process with additional attributes whereby none of the existing concepts were changed; they are extended by utilizing the extensions elements explicitly supported by the BPMN specification. Therefore, the concept is standards-conform with regards to the BPMN notation. Currently, we have focused on process modeling languages with explicit control-flow relations. However, the batch activity concept might be also useful for flexible business process modeling languages, such as Case Management Model and Notation (CMMN) [73] or declarative modeling [119], to batch certain cases for specific tasks. For instance in CMMN, the proposed extensions on BPMN notation can be similarly added to the CMMN tasks, bu this needs further evaluations.

INTEGRATION OF FLEXIBILITY IN BATCH ACTIVITIES. In the requirements framework, adaptation and variability of batch activities was required to increase the flexibility of those. *Adaptation* is the ability to react on exceptions or special cases during batch creation or batch execution, whereas *variability* is the ability to provide different batch configuration variants for different customer groups, product types, etc. being handled by a process model. In this thesis, we provided means to support adaption ability of batch activities. Known special cases and exceptions occurring during the *batch creation* phase can be handled by means of the presented *FastTrackRule* activation rule (providing an immediate start for special cases) and by the introduced batch adjustments rules (providing a flexible batch configuration based on events). For the latter, a first application to a use case indicates positive results in Section 6.3, but further evaluations are needed.

Changes which occur during the *batch execution* can not be handled by these presented means, this is enabled by the user involvement strategies for the batch activity presented in this thesis. In particular, these involvement strategies foster adaptations in the *batch creation* as well as in the *batch execution* phase for known and unknown exception or special cases. This thesis has focused on presenting those different involvement strategies, but it was not discussed so far, how a user interface for batch monitoring and adaptation should be designed. Further, theses strategies are driven manually. They depend on the amount of time which users have to observe batch clusters, such that additional technical support might be helpful. Especially handling of errors, internal as well as external, are usually handled automatically and was not discussed yet.

MULTI-PROCESS BATCH PROCESSING CONCEPT. The multi-process batch processing concept introduced in this thesis uses OLCs to centrally specify the batch configurations. OLCs might not be necessarily be available in an organization, but they can also be automatically deduced from the existing process models in a process repository based on the work by Meyer and Weske [65]. In the prototypical implementation, it could be shown that the needed OLCs can be integrated in an existing BPMS by adding a few new components. For the concept and the prototypical implementation, we assumed in this work that the process model, which should be involved in batch processing, are deployed together with the respective OLC in one BPMS.

Currently all task performers assigned to one of the activities referencing the batch transition are allowed to perform a batch cluster. This could be enhanced by a user authorization concept for the batch creation and execution.

The presented concept is an optional batch processing approach that is started as soon as matching partners for an instance can be identified based on run time information. Here, the activity instances do not wait actively for each other. However, if scenarios exist where this is required, the introduced batch transition can be simply extended with the batch activation rule introduced in the basic concept.

10.3 FUTURE RESEARCH

In this thesis, a batch activity concept was introduced to capture batch requirements in process models and to automatically execute them based on a requirements framework. Additionally, means to allow flexibility during the batch activity execution and a concept for batch processing across multiple process models were presented. These introduced batch processing concepts are first important steps, but future work can be focused on the directions provided below.

The introduced batch processing concepts for business processes provide modeling as well as execution concepts. However, the support of users in the batch activity design and execution can be elaborated in future research:

BATCH ACTIVITY DESIGN. In particular, in the batch activity design, batch activities (or batch transition in a multi-process setting) are currently identified and configured based on expert knowledge by the process designers. The developed BPMN process simulator, which is able to simulate business process with batch activities, could be extended to identify an optimal batch activation rule for a process. Another possibility, in case execution logs exist, is to use process mining techniques allowing for the automated discovery of batch activities from those event logs; first approaches in this direction are presented in [58, 128]. However, those techniques are not capable to mine a complete batch configuration, such that more research work in this direction is needed. Further, an application of the batch processing concepts presented in this thesis, e. g., in a user study or in a technical action research [133], will give more insights into the usability and usefulness of the presented concepts. These might also lead to new requirements which extend or detail the introduced requirements framework.

BATCH ACTIVITY EXECUTION. Regarding the batch execution, we plan to develop a user interface for the monitoring and adaptation of batch clusters. Additionally, it should be researched whether event processing techniques, which have been already applied in the batch creation phase to adapt clusters, can be also applied in the batch execution phase to allow additional flexibility and to handle exceptions. Recent research efforts focus on process predictions, for instance, on predicting the remaining time [81, 94], delays [101], or the process outcome [56]. These types of techniques could support and optimize the batch activation. In this work, we assume that the instances for batching are all running in the same BPMS. In future work, this assumption could be relaxed and batching of processes and process instances in distributed systems could be researched. Thereby, research works on coordinating services, for example in [125], could be considered.

With regards to flexibility of batch activities, this thesis presented different means to support the batch activity adaptation, but the ability to allow variability of batch activities was not discussed yet. In future, a concept to allow variability of the batch activity configurations, maybe supported by the batch processing discovery techniques, can be developed. Going a step further, the current batch activity concept which is a design and implementation concept could be used as basis for a dynamically batch processing concept where batches can be created at any activity, if certain conditions are fulfilled.

Part V

APPENDIX

BIBLIOGRAPHY

[1] Madis Abel. Lightning fast business process simulator. Master's thesis, Institute of Computer Science, University of Tartu, 2011. (Cited on page 127.)

[2] K.C. Arora. *Production and Operations Management*. Laxmi Publications Pvt Limited, 2004. (Cited on pages 27, 28, 29, and 129.)

[3] Norman T.J. Bailey. On queueing processes with bulk service. *Journal of the Royal Statistical Society. Series B (Methodological)*, 16(1):80–87, 1954. (Cited on pages 30 and 50.)

[4] Jerry Banks. *Discrete-event system simulation*. Pearson Education India, 1984. (Cited on page 127.)

[5] Roland Barcia, Geoffrey Hambrick, Kyle Brown, Robert Peterson, and Kulvir Singh Bhogal. *Persistence in the Enterprise: A Guide to Persistence Technologies*. developerWorks Series. Pearson Education, 2008. (Cited on pages 31 and 32.)

[6] Alistair Barros, Thomas Hettel, and Christian Flender. Process choreography modeling. In *Handbook on Business Process Management 1*, pages 257–277. Springer, 2010. (Cited on page 13.)

[7] Anne Baumgraß, Mirela Botezatu, Claudio Di Ciccio, Remco Dijkman, Paul Grefen, Marcin Hewelt, Jan Mendling, Andreas Meyer, Shaya Pourmirza, and Hagen Völzer. Towards a methodology for the engineering of event-driven process applications. In *International Conference on Business Process Management (BPM)*, pages 501–514. Springer, 2015. (Cited on page 25.)

[8] Jörg Becker, Martin Kugeler, and Michael Rosemann. *Process Management: A guide for the design of business processes*. Springer Science & Business Media, 2013. (Cited on pages 13 and 14.)

[9] Philip A Bernstein and Eric Newcomer. *Principles of Transaction Processing*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers, 1997. (Cited on pages 31 and 32.)

[10] Bizagi. Bizagi BPM Suite. `https://www.bizagi.com/`. (Cited on pages 41 and 42.)

[11] Patrice Briol. *BPMN, the Business Process Modeling Notation Pocket Handbook*. LuLu. com, 2008. (Cited on pages 15, 16, and 17.)

[12] Camunda. Batch processing interface to the camunda engine. `https://docs.camunda.org/manual/7.7/user-guide/process-engine/batch/`, . (Cited on page 32.)

[13] Camunda. Camunda BPM platform. `https://www.camunda.org`, . (Cited on pages 11, 32, 41, 42, and 117.)

[14] Richard B. Chase, F. Robert Jacobs, and Nicholas J. Aquilano. *Operations Management for Competitive Advantage*. McGraw-Hill Higher Education, 10 edition, 2004. (Cited on pages 27 and 28.)

[15] Michele Chinosi and Alberto Trombetta. BPMN: An introduction to the standard. *Computer Standards & Interfaces*, 34(1):124–134, 2012. (Cited on page 15.)

[16] David Cohn and Richard Hull. Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes. *IEEE Data Engineering Bulletin*, 32(3):3–9, 2009. (Cited on page 107.)

[17] Sue Conger. Six sigma and business process management. In *Handbook on Business Process Management 1*, pages 127–146. Springer, 2015. (Cited on page 3.)

[18] Mark M Davis. How long should a customer wait for service? *Decision Sciences*, 22(2):421–434, 1991. (Cited on pages 30 and 140.)

[19] Umeshwar Dayal. Active Database Management Systems. In *Proceedings of the Third International Conference on Data and Knowledge Bases: Improving Usability and Responsiveness*, pages 150–169, 1988. (Cited on pages 65 and 90.)

[20] Rajat K. Deb and Richard F. Serfozo. Optimal control of batch service queues. *Advances in Applied Probability*, 5(2):340–361, 1973. (Cited on page 30.)

[21] Gero Decker. *Design and analysis of process choreographies*. PhD thesis, University of Potsdam, 2009. (Cited on page 13.)

[22] Dhananjay M Dhamdhere. *Operating Systems: A Concept-based Approach, 2E*. Tata McGraw-Hill Education, 2006. (Cited on page 31.)

[23] Dirk Draheim. *Business process technology: A unified view on business processes, workflows and enterprise applications*. Springer Science & Business Media, 2010. (Cited on page 3.)

[24] Marlon Dumas, Wil MP van der Aalst, and Arthur HM ter Hofstede. *Process-aware information systems: bridging people and software through process technology*. John Wiley & Sons, 2005. (Cited on page 129.)

[25] Marlon Dumas, Luciano García-Bañuelos, Marcello La Rosa, and Reina Uba. Fast detection of exact clones in business process model repositories. *Information Systems*, 38(4):619–633, 2013. (Cited on page 101.)

[26] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of business process management*, volume 1. Springer, 2013. (Cited on pages 3, 4, 8, 13, 15, 20, 30, 32, 37, 51, 55, 126, 129, and 130.)

[27] M. Ebbers, E. Ramos, J. van Cappelle, L. Duijvestijn, T. Kaneki, M. Packer, and IBM Redbooks. *Approaches to Optimize Batch Processing on z/OS*. IBM Redbooks, 2012. (Cited on pages 31 and 32.)

[28] Chathura C Ekanayake, Marcello La Rosa, Arthur HM Ter Hofstede, and Marie-Christine Fauvet. Fragment-based version management for

repositories of business process models. In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems" (CoopIS)*, pages 20–37. Springer, 2011. (Cited on page 101.)

[29] Opher Etzion and Peter Niblett. *Event processing in action*. Manning Publications Co., 2010. (Cited on pages 25 and 85.)

[30] Walid Fdhila, Manuel Gall, Stefanie Rinderle-Ma, Juergen Mangler, and Conrad Indiono. Classification and formalization of instance-spanning constraints in process-driven applications. In *International Conference on Business Process Management (BPM)*, pages 348–364. Springer, 2016. (Cited on page 33.)

[31] António Paulo Freitas and José Luís Mota Pereira. Process simulation support in BPM tools: The case of BPMN. In *5th International Conference on Business Sustainability (BS)*. 2100 Projects, 2015. (Cited on page 127.)

[32] Jakob Freund and Bernd Rücker. *Praxishandbuch BPMN 2.0*. Carl Hanser Verlag GmbH Co KG, 2014. (Cited on page 15.)

[33] Johannes Göbel, Philip Joschko, Arne Koors, and Bernd Page. The Discrete Event Simulation Framework DESMO-J: Review, Comparison To Other Frameworks And Latest Development. In *European Conference on Modelling and Simulation (ECMS)*, pages 100–109, 2013. (Cited on pages 97 and 127.)

[34] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979. (Cited on page 29.)

[35] Michael Hammer. What is business process management? In *Handbook on Business Process Management 1*, pages 3–16. Springer, 2010. (Cited on page 3.)

[36] Paul Harmon. State of business process management 2016, 2016. URL `http://www.bptrends.com/bpt/wp-content/uploads/2015-BPT-Survey-Report.pdf`. (Cited on pages 10 and 15.)

[37] Paul Harmon and Celia Wolf. The state of business process management, 03 2016. URL `http://www.bptrends.com/bpt/wp-content/uploads/2015-BPT-Survey-Report.pdf`. (Cited on page 3.)

[38] Nico Herzberg, Andreas Meyer, and Mathias Weske. An Event Processing Platform for Business Process Management. In *Enterprise Distributed Object Computing (EDOC)*, pages 107–116. IEEE, 2013. (Cited on pages 24, 25, and 85.)

[39] David Hollingsworth and UK Hampshire. Workflow management coalition: The workflow reference model. *Document Number TC00-1003*, 1995. (Cited on page 21.)

[40] Process Maker Inc. Batch routing plugin of process maker 3.0. `http://wiki.processmaker.com/3.0/Batch_Routing{#}Overview`, . (Cited on pages 5 and 32.)

[41] Process Maker Inc. Processmaker workflow & bpm software suite. https://www.processmaker.com/, . (Cited on pages 5 and 32.)

[42] Marta Indulska, Peter Green, Jan Recker, and Michael Rosemann. Business process modeling: Perceived benefits. *Conceptual Modeling (ER)*, pages 458–471, 2009. (Cited on page 7.)

[43] Christian Janiesch, Martin Matzner, and Oliver Müller. Beyond process monitoring: A proof-of-concept of event-driven business activity management. *Business Process Management Journal*, 18(4):625–643, 2012. (Cited on page 25.)

[44] Monique Jansen-Vullers and Mariska Netjes. Business process simulation–a tool survey. In *7th Workshop on the Practical Use of Coloured Petri Nets and CPN Tools*, 2006. (Cited on page 127.)

[45] Gerti Kappel and Michael Schrefl. Object/behavior diagrams. In *International Conference on Data Engineering (ICDE)*, pages 530–539. IEEE, 1991. (Cited on pages 19 and 101.)

[46] David G Kendall. Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain. *The Annals of Mathematical Statistics*, pages 338–354, 1953. (Cited on page 126.)

[47] Andres Knöpfel, Bernhard Gröne, and Peter Tabeling. *Fundamental Modeling Concepts: Effective Communication of IT Systems*. Wiley, 2005. (Cited on pages 21, 95, 119, and 122.)

[48] Julian Krumeich, Benjamin Weis, Dirk Werth, and Peter Loos. Event-driven business process management: Where are we now? *Business Process Management Journal*, 20(4):615–633, 07 2014. (Cited on page 24.)

[49] Vera Künzle and Manfred Reichert. PHILharmonicFlows: Towards a Framework for Object-aware Process Management. *Journal of Software Maintenance and Evolution: Research and Practice*, 23(4):205–244, 2011. (Cited on page 34.)

[50] Jochen Küster, Ksenia Ryndina, and Harald Gall. Generation of business process models for object life cycle compliance. *Business Process Management*, pages 165–181, 2007. (Cited on page 20.)

[51] Zakir Laliwala, Rahul Khosla, Pritha Majumdar, and Sanjay Chaudhary. Semantic and rules based event-driven dynamic web services composition for automation of business processes. In *Services Computing Workshops (SCW)*, pages 175–182. IEEE, 2006. (Cited on page 65.)

[52] Frank Leymann and Dieter Roller. *Production workflow: Concepts and Techniques*. Prentice Hall, 2000. (Cited on pages 3, 4, 5, 20, and 33.)

[53] Jianxun Liu and Jinmin Hu. Dynamic batch processing in workflows: Model and implementation. *Future Generation Computer Systems*, 23(3): 338–347, 2007. (Cited on pages 4, 5, 34, 35, 37, 49, 50, 79, 80, 81, and 101.)

[54] Jianxun Liu, Yiping Wen, Ting Li, and Xuyun Zhang. A data-operation model based on partial vector space for batch processing in workflow. *Concurrency and Computation: Practice and Experience*, 23(16):1936–1950, 2011. (Cited on page 35.)

[55] David Luckham. The power of events: An introduction to complex event processing in distributed enterprise systems. In *International Workshop on Rules and Rule Markup Languages for the Semantic Web*, pages 3–3. Springer, 2008. (Cited on pages 24 and 85.)

[56] Fabrizio Maria Maggi, Chiara Di Francescomarino, Marlon Dumas, and Chiara Ghidini. Predictive monitoring of business processes. In *International Conference on Advanced Information Systems Engineering*, pages 457–472. Springer, 2014. (Cited on page 150.)

[57] Arunava Maity and UC Gupta. Analysis and optimal control of a queue with infinite buffer under batch-size dependent versatile bulk-service rule. *Opsearch*, 52(3):472–489, 2015. (Cited on page 30.)

[58] Niels Martin, Marijke Swennen, Benoît Depaire, Mieke Jans, An Caris, and Koen Vanhoof. Retrieving batch organisation of work insights from event logs. *Decision Support Systems*, 2017. (Cited on pages 33, 50, 51, 147, and 150.)

[59] Joseph S Martinich. *Production and operations management: An applied modern approach*. John Wiley & Sons, 2008. (Cited on pages 27, 28, 29, and 49.)

[60] Muthu Mathirajan and Appa Iyer Sivakumar. A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. *The International Journal of Advanced Manufacturing Technology*, 29(9-10):990–1001, 2006. (Cited on pages 5, 29, and 51.)

[61] D McCoy, R Schulte, Frank Buytendijk, Nigel Rayner, and A Tiedrich. Business activity monitoring: The promise and reality. *Gartner, Gartner's Marketing Knowledge and Technology Commentary COM-13-9992*, 2001. (Cited on page 14.)

[62] Robert McGill, John W. Tukey, and Wayne A. Larsen. Variations of box plots. *The American Statistician*, 32(1):12–16, 1978. (Cited on page 134.)

[63] Jyotiprasad Medhi. *Stochastic models in queueing theory*. Academic Press, 2002. (Cited on pages 5, 28, 30, 126, and 148.)

[64] Andreas Meyer. *Data perspective in business process management*. PhD thesis, University of Potsdam, 2015. (Cited on page 19.)

[65] Andreas Meyer and Mathias Weske. Activity-centric and Artifact-centric Process Model Roundtrip. In *Business Process Management Workshops*, pages 167–181. Springer, 2013. (Cited on pages 20 and 149.)

[66] Andreas Meyer and Mathias Weske. Weak Conformance between Process Models and Synchronized Object Life Cycles. In *Service-Oriented Computing (ICSOC)*, pages 359–367. Springer, 2014. (Cited on page 20.)

[67] Andreas Meyer, Luise Pufahl, Dirk Fahland, and Mathias Weske. Modeling and Enacting Complex Data Dependencies in Business Processes. In *International Conference on Business Process Management (BPM)*, pages 171–186. Springer, 2013. (Cited on page 121.)

[68] Christine Natschläger, Andreas Bögl, and Verena Geist. *Optimizing Resource Utilization by Combining Running Business Process Instances*, pages 120–126. Springer International Publishing, 2015. (Cited on page 35.)

[69] Christine Natschläger, Andreas Bögl, Verena Geist, and Miklós Biró. Optimizing resource utilization by combining activities across process instances. In *European Conference on Software Process Improvement*, pages 155–167. Springer, 2015. (Cited on pages 5, 34, 35, 37, 49, 50, 79, 80, 81, 101, and 104.)

[70] M. Neuts. A general class of bulk queues with poisson input. *The Annals of Mathematical Statistics*, 38(3):759–770, 1967. (Cited on pages 5, 30, 35, 65, 130, and 141.)

[71] OMG. Business Process Model and Notation (BPMN), V. 2.0, 2011. (Cited on pages 4, 15, 16, 24, 33, 45, 57, 68, and 69.)

[72] OMG. Unified Modeling Language (UML), Version 2.5, 2015. (Cited on pages 4, 15, 33, 57, 63, 70, and 91.)

[73] OMG. Case Management Model and Notation (CMMN), V. 1.1, 2016. (Cited on pages 4, 33, 71, and 148.)

[74] Djamila Ouelhadj and Sanja Petrovic. A survey of dynamic scheduling in manufacturing systems. *Journal of scheduling*, 12(4):417–431, 2009. (Cited on pages 29 and 51.)

[75] Katerina P Papadaki and Warren B Powell. Exploiting structure in adaptive dynamic programming algorithms for a stochastic batch service problem. *European Journal of Operational Research*, 142(1):108–127, 2002. (Cited on page 49.)

[76] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45–77, 2007. (Cited on pages 6, 7, 8, and 10.)

[77] Johannes Pflug and Stefanie Rinderle-Ma. Dynamic instance queuing in process-aware information systems. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1426–1433. ACM, 2013. (Cited on pages 34, 36, 49, 50, 51, 81, 101, and 130.)

[78] Johannes Pflug and Stefanie Rinderle-Ma. Application of dynamic instance queuing to activity sequences in cooperative business process scenarios. *International Journal of Cooperative Information Systems*, page 1650002, 2016. (Cited on pages 36, 79, and 80.)

[79] Johannes Pflug and Stefanie Rinderle-Ma. Process instance similarity: Potentials, metrics, applications. In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems" (CoopIS)*, pages 136–154. Springer, 2016. (Cited on page 59.)

[80] Benoît Polack, Jean-François Schved, and Bernard Boneu. Preanalytical recommendations of the 'groupe d'etude sur l'hemostase et la thrombose'(geht) for venous blood testing in hemostasis laboratories. *Pathophysiology of Haemostasis and Thrombosis*, 31(1):61–68, 2001. (Cited on page 97.)

[81] Mirko Polato, Alessandro Sperduti, Andrea Burattin, and Massimiliano de Leoni. Data-aware remaining time prediction of business process

instances. In *Neural Networks (IJCNN), 2014 International Joint Conference on*, pages 816–823. IEEE, 2014. (Cited on page 150.)

[82] Chris N Potts and Mikhail Y Kovalyov. Scheduling with batching: A review. *European journal of operational research*, 120(2):228–249, 2000. (Cited on pages 5, 29, 50, 51, and 52.)

[83] Luise Pufahl and Mathias Weske. Batch Activities in Process Modeling and Execution. In *International Conference on Service-Oriented Computing (ICSOC)*, pages 283–297. Springer, 2013. (Cited on page 57.)

[84] Luise Pufahl and Mathias Weske. Batch processing across multiple business processes based on object life cycles. In *International Conference on Business Information Systems (BIS)*, pages 195–208. Springer, 2016. (Cited on page 101.)

[85] Luise Pufahl and Mathias Weske. Enabling batch processing in bpmn processes. In *BPM (Demos)*, pages 28–33, 2016. (Cited on page 117.)

[86] Luise Pufahl and Mathias Weske. Requirements framework for batch processing in business processes. In *Business Process Modeling, Development and Support (BPMDS)*. Springer, 2017. (Cited on pages 27 and 41.)

[87] Luise Pufahl, Ekaterina Bazhenova, and Mathias Weske. Evaluating the performance of a batch activity in process models. In *Business Process Management Workshops*, pages 277–290. Springer, 2014. (Cited on pages 126 and 140.)

[88] Luise Pufahl, Nico Herzberg, Andreas Meyer, and Mathias Weske. Flexible batch configuration in business processes based on events. In *International Conference on Service-Oriented Computing (ICSOC)*, pages 63–78. Springer, 2014. (Cited on page 85.)

[89] Luise Pufahl, Andreas Meyer, and Mathias Weske. Batch regions: process instance synchronization based on data. In *Enterprise Distributed Object Computing Conference (EDOC)*, pages 150–159. IEEE, 2014. (Cited on page 57.)

[90] Luise Pufahl, Tsun Yin Wong, and Mathias Weske. Design of an Extensible BPMN Process Simulator. In *Business Process Management Workshops*, 2017. (Cited on pages 126 and 127.)

[91] Anne Vinter Ratzer, Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank Qvortrup, Martin Stig Stissing, Michael Westergaard, Søren Christensen, and Kurt Jensen. Cpn tools for editing, simulating, and analysing coloured petri nets. In *International Conference on Application and Theory of Petri Nets*, pages 450–462. Springer, 2003. (Cited on page 127.)

[92] Manfred Reichert and Barbara Weber. *Enabling flexibility in process-aware information systems: Challenges, methods, technologies*. Springer Science & Business Media, 2012. (Cited on pages 36 and 51.)

[93] Hajo A Reijers and S Liman Mansar. Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics. *Omega*, 33(4):283–306, 2005. (Cited on page 32.)

[94] Andreas Rogge-Solti and Mathias Weske. Prediction of business process durations using non-markovian stochastic petri nets. *Information Systems*, 54:1–14, 2015. (Cited on page 150.)

[95] Nick Russell, Wil MP van der Aalst, Arthur HM Ter Hofstede, and David Edmond. Workflow resource patterns: Identification, representation and tool support. In *International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 216–232. Springer, 2005. (Cited on pages 4, 21, 33, 51, 74, 78, and 81.)

[96] Shazia Sadiq, Maria Orlowska, Wasim Sadiq, and Karsten Schulz. When workflows will not deliver: The case of contradicting work practice. *International Conference on Business Information Systems (BIS)*, 5:69–84, 2005. (Cited on pages 5, 34, 37, 50, 79, 81, and 101.)

[97] Oumaima Saidani and Selmin Nurcan. Towards context aware business process modelling. In *Business Process Modeling, Development, and Support (BPMDS)*, volume 7, page 1, 2007. (Cited on page 51.)

[98] August-Wilhelm Scheer, Oliver Thomas, and Otmar Adam. Process modeling using event-driven process chains. *Process-aware information systems*, pages 119–146, 2005. (Cited on pages 4, 15, 33, 57, and 70.)

[99] Wasana Sedera, Guy G Gable, Michael Rosemann, and Robert W Smyth. A success model for business process modeling: findings from a multiple case study. In *PACIS 2004 Proceedings. 38.*, 2004. (Cited on page 7.)

[100] Shokri Z Selim. Time-dependent solution and optimal control of a bulk service queue. *Journal of Applied Probability*, 34(1):258–266, 1997. (Cited on page 30.)

[101] Arik Senderovich, Matthias Weidlich, Avigdor Gal, and Avishai Mandelbaum. Queue mining–predicting delays in service processes. In *International Conference on Advanced Information Systems Engineering*, pages 42–57. Springer, 2014. (Cited on page 150.)

[102] Karabi Sikdar and UC Gupta. Analytic and numerical aspects of batch service queues with single vacation. *Computers & operations research*, 32 (4):943–966, 2005. (Cited on page 30.)

[103] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts, 4th Edition*. McGraw-Hill Book Company, 2001. (Cited on page 58.)

[104] SH Sim and JGC Templeton. Steady state results for the m/m (a, b)/c batch-service system. *European Journal of Operational Research*, 21(2):260–267, 1985. (Cited on page 30.)

[105] Jacob V Simons and Gregory R Russell. A case study of batching in a mass service operation. *Journal of Operations Management*, 20(5):577–592, 2002. (Cited on page 30.)

[106] Jacob V. Simons Jr., Gerard Burke, and Gregory R. Russell. A cost-based model for customer batching in mass service operations. *Journal of Service Science Research*, 3(2):123–151, 2011. (Cited on page 30.)

[107] Nigel Slack, Stuart Chambers, and Robert Johnston. *Operations and process management: principles and practice for strategic impact*. Pearson Education, 2009. (Cited on pages 27, 28, 29, 51, 129, and 131.)

[108] Andrew Tanenbaum. *Modern operating systems*. Pearson Education, Inc.,4th Revised edition, 2014. (Cited on pages 27, 31, 49, 130, and 131.)

[109] Kerim Tumay. Business process simulation. In *Proceedings of the 27th conference on Winter simulation*, pages 55–60. IEEE Computer Society, 1995. (Cited on page 127.)

[110] Esko Ukkonen. Algorithms for approximate string matching. *Information and control*, 64(1-3):100–118, 1985. (Cited on page 20.)

[111] Department of Computer Science University of Hamburg. DesmoJ - A Framework for Discrete-Event Modeling and Simulation. `http://desmoj.sourceforge.net/`. (Cited on pages 97 and 127.)

[112] Wil MP van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011. (Cited on page 14.)

[113] Wil MP van der Aalst. Business process management: a comprehensive survey. *ISRN Software Engineering*, 2013, 2013. (Cited on pages 3 and 13.)

[114] Wil MP van der Aalst and Arthur HM ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005. (Cited on pages 4 and 33.)

[115] Wil MP van der Aalst and Kees Max van Hee. *Workflow management: models, methods, and systems*. MIT press, 2004. (Cited on pages 3, 36, and 51.)

[116] Wil MP van der Aalst, Paulo Barthelmess, Clarence A Ellis, and Jacques Wainer. Proclets: A framework for lightweight interacting workflow processes. *International Journal of Cooperative Information Systems*, 10(04): 443–481, 2001. (Cited on page 34.)

[117] Wil MP van der Aalst, Arthur HM ter Hofstede, Bartek Kiepuszewski, and Alistair P Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003. (Cited on pages 5, 14, and 33.)

[118] Wil MP van der Aalst, Michael Rosemann, and Marlon Dumas. Deadline-based escalation in process-aware information systems. *Decision Support Systems*, 43(2):492–511, 2007. (Cited on page 32.)

[119] Wil MP van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science-Research and Development*, 23(2):99–113, 2009. (Cited on pages 71 and 148.)

[120] Wil MP van der Aalst, Joyce Nakatumba, Anne Rozinat, and Nick Russell. Business process simulation. In *Handbook on Business Process Management 1*, pages 313–338. Springer, 2010. (Cited on pages 126 and 127.)

[121] Wil MP van der Aalst, Kees M van Hee, Arthur HM ter Hofstede, Natalia Sidorova, HMW Verbeek, Marc Voorhoeve, and Moe Thandar Wynn. Soundness of workflow nets: classification, decidability, and

analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011. (Cited on page 13.)

[122] Jan vom Brocke and Michael Rosemann. *Handbook on business process management*. Springer, 2010. (Cited on page 3.)

[123] R Hevner Von Alan, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, 28(1): 75–105, 2004. (Cited on pages 6 and 8.)

[124] Barbara Weber, Manfred Reichert, Jan Mendling, and Hajo A Reijers. Refactoring large process model repositories. *Computers in Industry*, 62 (5):467–486, 2011. (Cited on page 49.)

[125] Andreas Weiß, Vasilios Andrikopoulos, Michael Hahn, and Dimka Karastoyanova. ChorSystem: A Message-based System for the Life Cycle Management of Choreographies. In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems" (CoopIS)*, pages 503–521. Springer-Verlag, 2016. (Cited on page 150.)

[126] Howard J Weiss. The computation of optimal control limits for a queue with batch services. *Management Science*, 25(4):320–328, 1979. (Cited on page 30.)

[127] Howard J Weiss and Stanley R Pliska. Optimal control of some markov processes with applications to batch queueing and continuous review inventory systems. *The Center for Mathematical Studies in Economics and Management Science, Discussion Paper*, 214, 1976. (Cited on page 30.)

[128] Yiping Wen, Zhigang Chen, Jianxun Liu, and Jinjun Chen. Mining batch processing workflow models from event logs. *Concurrency and Computation: Practice and Experience*, 25(13):1928–1942, 2013. (Cited on pages 35, 147, and 150.)

[129] Yiping Wen, Jianxun Liu, Zhigang Chen, and Buqing Cao. Dynamic scheduling optimization for instance aspect handling in workflows. In *Semantics, Knowledge and Grids (SKG), 2014 10th International Conference on*, pages 57–62. IEEE, 2014. (Cited on page 35.)

[130] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures. Second Edition*. Springer, 2012. (Cited on pages 3, 4, 13, 14, 15, 16, 20, 21, 22, 23, 70, and 81.)

[131] Matthias Kunze Mathias Weske. *Behavioural Models: From Modelling Finite Automata to Analysing Business Processes*. Springer International Publishing, 2016. (Cited on page 14.)

[132] Stephen A White. *BPMN modeling and reference guide: understanding and using BPMN*. Future Strategies Inc., 2008. (Cited on pages 15 and 16.)

[133] Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014. (Cited on pages 8, 96, 125, and 150.)

[134] Ray Wild. *Essentials of Operations Management*. Cengage Learning EMEA, 2002. (Cited on page 28.)

[135] Tsun Yin Wong, Susanne Bülow, and Mathias Weske. Monitoring batch regions in business processes. In *CAiSE Workshops*, pages 317–323. Springer, 2015. (Cited on pages 36, 51, and 82.)

[136] Zhiqiang Yan, Remco M. Dijkman, and Paul W. P. J. Grefen. Business Process Model Repositories - Framework and Survey. *Information & Software Technology*, 54(4):380–395, 2012. (Cited on pages 101 and 104.)

[137] Armin Zimmermann. *Stochastic Discrete Event Systems*. Springer, 2007. (Cited on page 30.)

[138] Michael zur Muehlen and Robert Shapiro. Business process analytics. In *Handbook on Business Process Management 2*, pages 137–157. Springer, 2010. (Cited on page 24.)

All links were last followed on October 4, 2017.