



Universidade do Minho

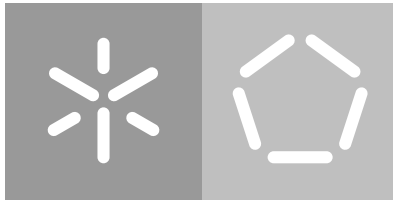
Escola de Engenharia

Departamento de Informática

Rui Pedro Abreu Calheno

Process Mining applied to BPMN-E²

May 2021



Universidade do Minho

Escola de Engenharia

Departamento de Informática

Rui Pedro Abreu Calheno

Process Mining applied to BPMN-E²

Master dissertation

Master Degree in Informatics Engineering

Dissertation supervised by

Professor Doctor Solange Rito Lima

Professor Doctor Pedro Rangel Henriques

May 2021

COPYRIGHT AND TERMS OF USE FOR THIRD PARTY WORK

This dissertation reports on academic work that can be used by third parties as long as the internationally accepted standards and good practices are respected concerning copyright and related rights.

This work can thereafter be used under the terms established in the license below.

Readers needing authorization conditions not provided for in the indicated licensing should contact the author through the RepositóriUM of the University of Minho.

LICENSE GRANTED TO USERS OF THIS WORK:



CC BY

<https://creativecommons.org/licenses/by/4.0/>

ACKNOWLEDGEMENTS

I hereby recognise those who supported me during the writing of this dissertation.

I express my deepest gratitude to my father and mother. For the untiring efforts and sacrifices they made to build my foundations as a student and as a person. Nothing of this would be possible without you.

To my uncle Rui Maranhão. For the endless insights, teachings and opportunities you offered me from day one. I am certain it was you, who gave me the *bug* for Software Engineering.

To my grandfather Bernardino Capitão de Abreu. For that I know you would be the proudest of my accomplishments. No amount of words can begin to describe my admiration for you. Everyday, I look up to you and your endless affection, determination and resolve. I try my best to follow your steps. It is so hard and you made it look so effortless.

To all the other members of my family who, in their own ways, cared and supported me in this journey.

To my girlfriend. For always finding a place and a time to hear my ramblings, even when you were full of concerns of your own.

To all the professors who supervised this dissertation Solange Rito Lima, Pedro Rangel Henriques, Paulo Martins Carvalho and Mateo Ramos-Merino. I thank the guidance you offered me and the trust you placed on my work. I also thank Mateo for the development of the BPMN-E² notation which lead to the proposal of such an interesting and challenging dissertation topic.

Finally, to my friends of now and my friends of then. For the countless *small* moments we shared during the pursuit of our greatest, individual achievements.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ABSTRACT

Process Mining is characterized by a group of techniques that aim to mine and analyze event logs in an effort to extract patterns and useful insights regarding a business process, allowing for a better and more efficient understanding of it.

This topic is sparking increasing interest in both academia and business contexts, which results in fast advances in the algorithms being applied, as well as in the subjacent notations used for process modeling. One of the most used notations for process modeling is Business Process Model and Notation (BPMN), being its expressiveness in representing processes its strongest attribute. However, this notation reveals some flaws when dealing with some specific contexts, struggling to model activity duration, quality control and activity effects in context-specific resources. For this particular purpose, an extension named Business Process Model and Notation Extended Expressiveness (BPMN-E²) was developed to tackle the limitations found on the original notation.

In this dissertation, a new conformance checking algorithm was developed focusing on finding non-conformities between an event log and process models, taking into consideration the new elements that BPMN-E² has to offer. Fuelled by a few setbacks found during this work, an event log clustering technique was also developed to downsize large event logs without stripping its representativity.

Furthermore, the BPMN-E² notation was used to model a real-life process and the developed conformance checking algorithm was applied to illustrate its analytical potential.

Keywords: Process Mining, Conformance checking, Data-aware conformance checking, event log, Process modelling, BPMN, BPMN-E², Trace clustering, Cluster Analysis, Data Mining

RESUMO

Process Mining caracteriza um conjunto de técnicas que permitem a mineração e análise de *event logs* com o principal objetivo de extrair destes padrões e informações relevantes que permitam uma melhor percepção e eficiência dos processos realizados num determinado contexto.

Esta área tem verificado um interesse crescente, tanto em meio acadêmico como em meio empresarial, sendo notados avanços quer nos algoritmos de mineração utilizados, quer nas notações subjacentes utilizadas para modelar processos. Uma das notações mais utilizadas por profissionais e acadêmicos é o Business Process Model and Notation (BPMN) devido à sua expressividade na representação de processos. No entanto, esta mesma notação apresenta alguns inconvenientes quando é usada em determinados contextos, sendo difícil representar, por exemplo, durações de atividades, controlo de qualidade e efeitos da atividade nas características de um produto. Num esforço para resolver estes problemas, foi desenvolvida uma extensão chamada Business Process Model and Notation Extended Expressiveness (BPMN-E²).

Neste projeto foi desenvolvido um novo algoritmo de conformance checking, tendo em consideração a informação complementar oferecida pelo BPMN-E². Motivada por alguns contratemplos durante o trabalho, uma técnica de *clustering* foi também desenvolvida para reduzir o tamanho de *event logs* sem afetar a sua representatividade.

A notação BPMN-E² foi também usada para modelar um processo real e o algoritmo de *conformance checking* usado nesse contexto para ilustrar o seu potencial analítico.

Palavras-chave: Mineração de processos, Verificação de Conformidade, Verificação de conformidade *data-aware*, Registo de eventos, Modelação de processos, BPMN, BPMN-E², *Clustering* de traços, Análise de *clusters*, Mineração de dados

CONTENTS

| | | |
|-------|--|----|
| 1 | INTRODUCTION | 1 |
| 1.1 | Motivation | 1 |
| 1.2 | Objectives | 2 |
| 1.3 | Dissertation layout | 3 |
| 2 | RELATED WORK | 4 |
| 2.1 | Process Mining | 4 |
| 2.1.1 | Event logs and Process Mining perspectives | 4 |
| 2.1.2 | Conformance checking | 5 |
| 2.1.3 | Data-aware conformance checking | 8 |
| 2.2 | BPMN and BPMN-E ² | 9 |
| 2.3 | Satisfiability Modulo Theories | 10 |
| 2.4 | Cluster Analysis | 11 |
| 2.4.1 | Distance measures | 11 |
| 2.4.2 | Hierarchichal clustering | 12 |
| 2.4.3 | Non-hierarchical clustering | 12 |
| 2.4.4 | Clustering in Process Mining | 13 |
| 2.5 | Summary | 14 |
| 3 | DATA-AWARE CONFORMANCE CHECKING | 15 |
| 3.1 | Design goals | 15 |
| 3.2 | Conversion phase | 16 |
| 3.2.1 | Directly follows Rules Model | 16 |
| 3.2.2 | SMT Solvers as conformance rules | 17 |
| 3.2.3 | Dealing with non-monitored activities | 18 |
| 3.2.4 | Associating rules to a DFRM | 18 |
| 3.3 | Conformance checking phase | 20 |
| 3.4 | Summary | 22 |
| 4 | EVENT-BASED TRACE CLUSTERING FOR LOG REDUCTION | 23 |
| 4.1 | Motivation | 23 |
| 4.2 | Vectorizing a trace | 24 |
| 4.3 | Clustering and sampling | 24 |
| 4.4 | Summary | 26 |
| 5 | IMPLEMENTATION | 27 |
| 5.1 | Architecture | 27 |

| | | |
|-------|--|----|
| 5.2 | Main functionalities | 29 |
| 5.2.1 | Conversion | 30 |
| 5.2.2 | Conformance Checking | 32 |
| 5.2.3 | Event-based trace clustering for log reduction | 33 |
| 5.3 | Summary | 34 |
| 6 | PROOF OF CONCEPT | 35 |
| 6.1 | Real-life use case | 36 |
| 6.1.1 | Event log | 36 |
| 6.1.2 | BPMN-E2 Process Model | 37 |
| 6.1.3 | Analyzing the results | 39 |
| 6.2 | Evaluation using synthetic data | 42 |
| 6.2.1 | Experimental setup | 42 |
| 6.2.2 | Conformance Checking evaluation | 43 |
| 6.2.3 | Event log reduction evaluation | 45 |
| 6.3 | Summary | 46 |
| 7 | CONCLUSIONS AND FUTURE WORK | 47 |

LIST OF FIGURES

| | | |
|------------|--|----|
| Figure 1.1 | BPMN documentation excerpt of Parenteral Nutrition (PN) mixtures elaboration process. | 2 |
| Figure 1.2 | Natural language documentation excerpt of PN mixtures elaboration process. | 2 |
| Figure 2.1 | Common structure of an event log [50]. | 5 |
| Figure 2.2 | Types of Process Mining explained in terms of input and output [48]. | 6 |
| Figure 2.3 | BPMN core elements [25]. | 9 |
| Figure 3.1 | Example of conversions from different BPMN-E ² extension elements. Underscores () are used to distinguish between attributes belonging to the source activity (prefix) and the target activity (suffix). | 17 |
| Figure 3.2 | Rule association when event data is recorded at the start of an activity. | 19 |
| Figure 3.3 | Rule association when event data is recorded at the end of an activity. | 19 |
| Figure 3.4 | Rule association when event data is recorded at the start and at the end of an activity. | 20 |
| Figure 3.5 | Excerpt of a BPMN-E ² model, extended with advanced decision points, activity durations and activity effects (left), and the respective DFRM graph after conversion (right). | 20 |
| Figure 4.1 | Stages of <i>Event-based trace clustering for log reduction</i> . Clustering stages were taken from [24]. | 25 |
| Figure 5.1 | Class diagram showing of the PM_BPMN_E2 library. | 28 |
| Figure 5.2 | BPMNE2DiagramGraph class diagram . | 30 |
| Figure 5.3 | DFRM class diagram. | 31 |
| Figure 6.1 | Example of a conformance checking HTML report. | 35 |
| Figure 6.2 | BPMN-E ² model of the road traffic fine management process. | 38 |
| Figure 6.3 | Fitness values for each conformance rule. The results are colour-coded, shades of green indicate satisfactory results while shades of red indicate less than ideal results (with a threshold of 0.7). | 39 |
| Figure 6.4 | Boxplot of the number of deviations per case. | 40 |
| Figure 6.5 | Pie chart highlighting deviations per rule type. | 41 |
| Figure 6.6 | Boxplot of duration for each duration-typed conformance rule. | 41 |

- Figure 6.7 Diagram ALL used during evaluation. Note that for each path, despite the distinct number of activities the number of rules to test was kept. 43
- Figure 6.8 Evaluation heatmap. Execution time's correlations are highlighted. 44
- Figure 6.9 Conformance Checking errors after applying the event log reduction technique, using K-means (left) and DBSCAN (right) clustering algorithms. 45

LIST OF TABLES

| | | |
|-----------|---|----|
| Table 2.1 | New elements introduced by the BPMN-E ² notation [40]. | 10 |
| Table 4.1 | Excerpt of the event log used in Section 6. Used here to illustrate the vectorization of event log traces. | 24 |
| Table 4.2 | Vectorized traces extracted from Table 4.1, using the attributes <i>amount</i> , <i>dismissal</i> , <i>paymentAmount</i> and <i>expense</i> . | 24 |
| Table 6.1 | Excerpt of the road traffic fine management process' event log. | 36 |
| Table 6.2 | Rules extracted from process' description. | 37 |
| Table 6.3 | Evaluation results. | 43 |

ACRONYMS

B

BPMN **B**usiness **P**rocess **M**odel and **N**otation.

BPMN-E² **B**usiness **P**rocess **M**odel and **N**otation - **E**xtended **E**xpressiveness.

C

CSV **C**omma **S**eparated **V**alues.

D

DBSCAN **D**ensity-**b**ased **S**patial clustering of **a**pplications with **n**oise.

DFM **D**irectly **F**ollows **M**odel.

DFRM **D**irectly **F**ollows **R**ules **M**odel.

H

HACCP **H**azard **A**nalysis and **C**ritical **C**ontrol **P**oints.

P

PN **P**arenterl **N**utrition.

S

SMT **S**atisfiability **M**odulo **T**heories.

X

XES **e**Xtensible **E**vent **S**tream.

INTRODUCTION

Nowadays, people and organizations are growing more dependant on technology and, as a consequence, an increasing amount of data is constantly being collected [50]. Considering this, today's organizations aim to extract information and value from data stored in their information systems to better improve their business and gain a competitive advantage over other organizations [22, 27, 50]. Process Mining is a rather recent discipline that combines data mining and process modeling to properly analyze the ever growing event data. It aims to discover, monitor and improve real processes, by extracting meaningful insights and knowledge from event logs [26, 49]. To do so, Process Mining relies on techniques that can be grouped into one of three areas considering its intended purpose [48]: a) *Discovery*, given an event log, produces a process model; b) *Conformance Checking*, given both an event log and a process model, produces a report comparing the two and identifying possible non-conformities between them; and c) *Enhancement*, given both an event log and a process model, improves the latter with new information recorded on the former.

All of these techniques have been proven useful for many organizations that didn't thought possible the extraction of such valuable information out of their recorded logs [48]. In fact, since its appearance, a community of over 60 organizations [26, 48] has been found with members ranging from software vendors to research institutes [48]. In addition, several case studies were conducted [35, 36, 41, 47], proving the usefulness of Process Mining on domains such as Healthcare, Finance and IT.

Over the years, several Process Mining tools were built to help both academics and business people better understand process models and "mine" useful information that otherwise would not be discovered. Currently, some of the most relevant tools for Process Mining are ProM [7], Disco [3], PM4py [6], Celonis [2].

1.1 MOTIVATION

Business Process Model and Notation - Extended Expressiveness (BPMN-E²) is an extension for the BPMN notation language [38] that gives to the process model designer the possibility to describe in a more detailed way the workflow behaviour, the activities being performed and

the context of one particular process [40]. The necessity of a more context-aware approach surfaced when dealing with process control environments, particularly Hazard Analysis and Critical Control Points (HACCP) based environments where it is vital to ensure the safety and security of products manufactured in a given industry (e.g., food, pharmaceutical or cosmetics) [40]. In these scenarios, the business processes are explained in detail using both a BPMN diagram and a natural language description that explains activity behaviour, quality controls or how to proceed in case of hazards. An example of such a process description can be found in Figure 1.1 and Figure 1.2.

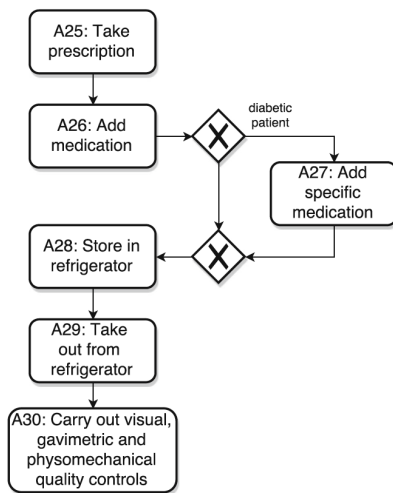


Figure 1.1: BPMN documentation excerpt of Parenteral Nutrition (PN) mixtures elaboration process.

In this phase, it is necessary to read the prescription and take note of all relevant aspects (diabetic, [...]). Then, the pharmaceutical operator must add now the prescribed medicaments for this type of parenteral nutrition. First, **add 100 mg of ranitidine**, [...] and remove the mixture during **30 seconds**. Second, if the patient is diabetic, **add the units of U-100 insulin pointed by the prescription**. Next, store the mixture in the refrigerator. **One hour** after, take out the parental nutrition from the refrigerator and **measure its temperature**. Some **quality controls** must be carried in order to validate the correct elaboration of the mixture. First, you have to perform a **visual control by checking if the closure of the PN bag is ok**. Then, **check the temperature** measured in A28, it must be **between 2 and 8°C**. Finally, check the physicochemical characteristics by **measure the pH**. It must be **between 5.4 and 6.5**

Figure 1.2: Natural language documentation excerpt of PN mixtures elaboration process.

With both a BPMN model and a natural language description, one has a rich and complete representation of the business process, however, only the model can be used during Process Mining tasks, which results in a loss of valuable information [40]. With this in mind, the authors of the BPMN-E² notation aimed to extend the BPMN element set with new stereotypes that represent the most valuable information, formerly only present in natural language documents.

1.2 OBJECTIVES

Considering the previous discussion, the main goal of this dissertation is the development of a new conformance checking algorithm, that takes into consideration the extended expressiveness that the BPMN-E² notation offers and builds a detailed report regarding duration, side-effects, and control flow related non-conformities.

To accomplish this primary goal, several secondary goals must be defined to assure the success of the former. They are: a) identification and comprehension of the new elements offered by the BPMN-E² notation; b) study and comprehension of the state-of-the-art algorithms used for conformance-checking purposes; c) development of a tool or plugin that applies the new algorithm; d) evaluation of the algorithm's performance and e) application of the algorithm to a real life use case.

1.3 DISSERTATION LAYOUT

This dissertation is organized in seven chapters.

The first chapter introduces Process Mining and explains the motivation behind this work, as well as, its intended objectives.

The second chapter presents the related work on Process Mining, process modeling (BPMN) and cluster analysis, introducing relevant concepts that inspired the remainder of the work and, therefore, are essential for the remaining of the dissertation.

The third chapter explores the approach thought to accomplish the proposed objectives. The main design goals are introduced and the several steps of the proposed conformance checking algorithm are thoroughly explained.

The fourth chapter introduces an *event-based trace clustering* technique aimed at reducing the size of an event log. Although it diverges slightly from the conformance checking aspect discussed until then, the proposal discussed in this chapter is, at its core, intended to reduce execution time of the existing data-aware conformance checking techniques.

The fifth chapter focus on the implementation of the proposed algorithms. It introduces the chosen programming language; highlights the main architectural choices and lists the main functionalities of the developed library.

The sixth chapter evaluates the developed conformance checking algorithm by applying it in the analysis of a real-life use case and testing its performance on synthetic event logs. The developed clustering technique is also evaluated.

Finally, the seventh, and last chapter, concludes this document by summarising the work done and exploring future work possibilities.

RELATED WORK

This chapter introduces and explains important concepts in the domains of: Process Mining, namely *event logs*, *perspectives*, *conformance checking* and *data-aware conformance-checking*; Process Modeling, focusing on the core notations of this dissertation, BPMN and BPMN-E²; and cluster analysis.

At the same time, it explores the work and state-of-the-art techniques that have been developed in the recent years regarding those same topics.

2.1 PROCESS MINING

Nowadays, we find ourselves surrounded with IT systems that constantly collect and store data about real-life events, such as, money withdrawal, social services appliances, tax declaration's submission, receipt's emission and others [48].

Process Mining is a relative young research discipline that sits between Machine Learning and Data Mining on the one hand, and Process Modeling and analysis on the other hand. The idea of Process Mining is to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs readily available in today's systems [48, 50].

Process Mining consists in a group of techniques that aim to extract knowledge from the huge amount of event data being produced thus providing insights, identifying bottlenecks and problems [48, 50], improving this way, both the organisation's business processes and the services offered to their clients.

2.1.1 *Event logs and Process Mining perspectives*

Process Mining is based on the assumption that it is possible to sequentially record events such that each event refers to an *activity* (i.e., a well-defined step in some process) and is related to a particular *case* (i.e., a process instance) [37, 48, 50], moreover a sequence of *activities* is referred to as a *trace*.

Therefore an *Event log* is a *multi-set* of *traces* related to a particular business process. To be properly analysed an *event log* must contain the following attributes [37]:

1. *Case id*, to identify the process instance being recorded;
2. *Activity id*, to identify the activity being performed;
3. *Timestamp*, to locate the activity in time, and provide order to the log.

Additional information can also be stored, for example, the resource(s) performing the activity or other relevant data related to an activity (i.e., size of an order, price of a product, type of customer) [37,50]. Information such as this can be proven of great value, allowing for analysis from different *perspectives* that complement the traditional control-flow focused approaches.

The three most common perspectives besides *control-flow* are [48,50]:

- *Organizational Perspective* - Concerns about the use of resources and their relation. Structures the organization by classifying resources or showing the social network;
- *Time perspective* - Concerns about the duration of the activities and the overall process. Enables bottleneck discovery, service levels measurement, resource management, and remaining case duration prediction;
- *Case perspective* - Concerns about properties about cases. These properties are commonly related to data elements that are relevant to the activities being performed. Provides a deeper and more detailed view on cases and on the overall process.

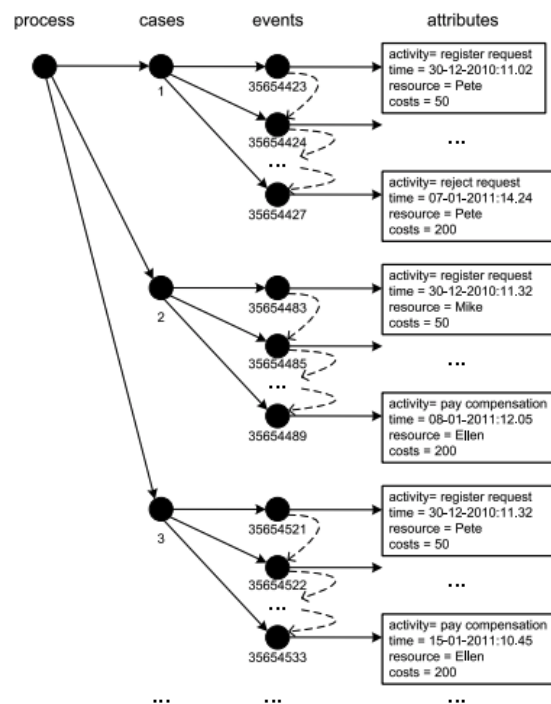


Figure 2.1: Common structure of an event log [50].

2.1.2 Conformance checking

As can be seen in Figure 2.2, Process Mining techniques are divided into three types: a) *Discovery* - techniques that take an event log and produces a process model based on it; b)

Conformance Checking - techniques that takes both an event log and a process model and produces a report to tackle non-conformities issues; and c) *Enhancement* - that produces a new and enhanced model based on both an event log and a process model. The foundations of this techniques are well established by the author Wil van der Aalst in [50], with new approaches being developed based on them.

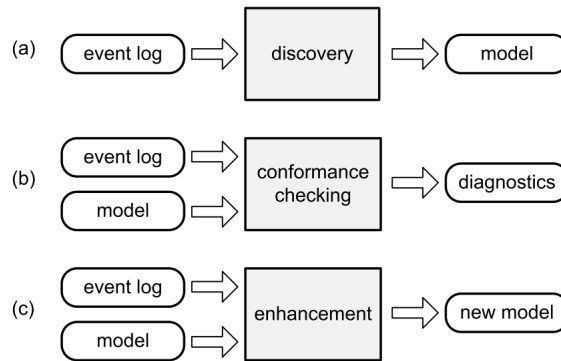


Figure 2.2: Types of Process Mining explained in terms of input and output [48].

Most of the research in Process Mining has mainly focused on discovery techniques, neglecting the importance of conformance. Models must accurately represent the reality of a process to provide trustworthy and effective decisions, however that is not always the case, with models being wrongly constructed or becoming obsolete due to changes in the business processes [37]. Conformance checking aims to pinpoint this deviations, enabling either the correction of the model or the identification of process errors. To measure the level of conformance between a model and an event log, one should be able to quantify how well the model represents reality. For this purpose, there are four quality criteria that together can be used to assess the quality of a given model [9, 37, 50]:

1. *Fitness* - The model should allow for the behaviour seen in the event log;
2. *Precision* - The model should not allow for behaviour completely unrelated to what was seen in the event log;
3. *Generalization* - The model should generalize the example behavior seen in the event log;
4. *Simplicity* - The model should be as simple as possible (*Occam's razor*).

Of the four, fitness is the most related criteria to conformance checking since it measures the proportion of the event's log valid behaviour according to the model [50]. Therefore, most of the existing conformance checking techniques provide ways to compute this fitness value.

Token replay

Token replay fitness is calculated by replaying each trace of the event log on top of the corresponding process model, while tracking all the situations where a transition was forced to fire without being enabled [50] (i.e., a non-conformity) . For this purpose, four counters are used:

1. Produced tokens - Counts the number of tokens that are produced by a transition;
2. Consumed tokens - Counts the number of tokens that are consumed by a transition;
3. Missing tokens - Counts the number of tokens that are missing when a transition needs to be fired;
4. Remaining tokens - Counts the number of tokens that are yet to be consumed when the trace ends.

This way, when comparing an event log with the correspondent process model the non-conformities will be expressed by the missing and remaining tokens. A level of fitness can be computed using the following equation:

$$fitness = \frac{1}{2} \cdot \left(1 - \frac{missing}{consumed}\right) + \frac{1}{2} \cdot \left(1 - \frac{remaining}{produced}\right)$$

Despite its simplicity, token based replayability has some shortcomings. In processes where there are many deviations the model becomes “flooded with tokens” allowing for any type of behaviour and therefore providing untrustworthy high levels of fitness. Moreover, if a case does not fit, the approach does not attempt to create a corresponding path through the model thus not providing a way to relate observed behaviour with modeled behaviour [50].

Alignment-based

An alignment between the event log and the process model indicates the most likely way that a particular trace can be replayed in the model. To establish an alignment between process model and event log we need to relate “moves” in the log to “moves” in the model [49]. A move is a pair (x,y) where the first element refers to the log and the second element refers to the model [50]. This way, an alignment is a sequence of moves, which can be:

1. *Move in log* - Denotes a move in the log, e.g., (a, \gg) represents an "a move" in the log, not mimicked by the model.
2. *Move in model* - Denotes a move in the model, e.g., (\gg, a) represents an "a move" in the model, not mimicked by the log.

3. *Move in both* - Denotes a synchronous move, e.g., (a, a) represents an "a move" in both the event log and the model.

It is important to note that, for a particular trace, there can be multiple alignments. The aim is to find the *optimal alignment*, i.e., the shortest (or less costly) path between the start and the end of the process [16,33]. Together, the optimal and worst alignment can be used to compute the fitness of a trace regarding a process model, base on the following equations:

$$fitness = 1 - \frac{\text{cost of optimal alignment}}{\text{cost of worst alignment}}$$

This way, the fitness for the entire log can be calculated using:

$$fitness = 1 - \frac{\sum_{\text{trace } \epsilon \text{ log}} \text{cost of optimal alignment for trace}}{\sum_{\text{trace } \epsilon \text{ log}} \text{cost of worst alignment for trace}}$$

2.1.3 Data-aware conformance checking

The lion's share of Process Mining research focuses on control-flow, i.e., the ordering of activities [18]. Therefore, the majority of the effort put on conformance-checking techniques is pointing towards the control-flow perspective, ignoring other perspectives such as data, resources and time [17]. This way, there can be a number of deviations that aren't caught during a conformance-checking task, e.g., activities that take longer than what is expected, activities that should not be executed by a certain resource or activities that fail to make specific changes.

Considering that focusing only on one perspective can lead to incomplete diagnosis, data-aware (also called multi-perspective) conformance-checking techniques were developed [17–19,31]. In [17] a technique is proposed that extends control-flow alignments to incorporate other perspectives by constructing an Integer Linear Programming (ILP) problem and consequently solving it. In [31], a different approach is proposed using Compliance Rules Graphs (CRG) [30] to declare a set of rules that the process execution must obey, each rule is bound to an activity, thus enabling to pinpoint the cause of a violation. More recently, in [28], Process Mining discovery and conformance approaches using *Directly follows Models* were proposed, given its intuitiveness and simplicity; however, these techniques focused on control-flow instead of data-flow. DFMs served as the baseline for the approach presented in this dissertation (in Chapter 3), being extended with conformance rules to accommodate data-aware conformance checking.

2.2 BPMN AND BPMN-E²

To model a business process, with Process Mining tasks in mind, one should consider which language to use (*representational bias*) ensuring that Process Mining techniques can be executed flawlessly, without compromising the understandability of the results [26,48]. Over the years, several process model notations were proposed, such as Petri nets, causal nets, process trees and BPMN, being this the standard notation for business process modeling, used by a variety of professionals in their everyday jobs (business analysts, product managers, technical designers, system architects and more) [26]. For this reason, it is of great value using BPMN to reduce *representational bias* for Process Mining, at least as a starting and ending point [25], with conversions to and from more viable notations being made "under the hood".

BPMN is a specification developed by the Business Process Management Initiative (BPMI). Its primary goal is to provide a notation that is readily understandable by all business users, from business analysts who create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and, finally, to the business people who will manage and monitor those processes. A Business Process Model, then, is a network of graphical objects, which are activities (i.e., work) and the flow controls that define their order of performance [51].

The BPMN notation provides a simple and understandable mechanism for creating business process models, being able to handle the complexity inherent to business processes. A BPMN diagram can be composed by a number of graphical elements, depending on the domain and the process being modeled. Nonetheless, there are six essential elements that together make the core of a BPMN model (see Figure 2.3). These are:

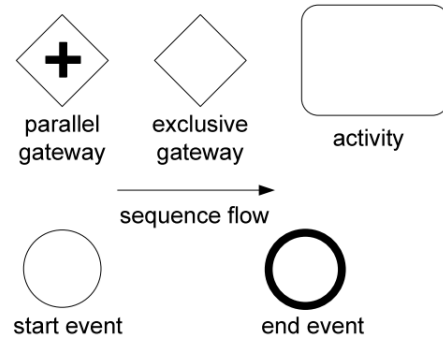


Figure 2.3: BPMN core elements [25].

a) *Start Event*, indicates where the process starts; b) *End Event*, indicates where the process ends; c) *Activity*, work that a company performs; d) *Sequence flow*, shows the order of activities, i.e., how the process should flow; e) *Parallel gateway*, indicates concurrency between activities; and f) *Exclusive gateway*, indicates exclusive decisions to be made. A list of other more specific and complex elements can be found in [38].

However, despite having a great support for process modeling, the BPMN notation has some drawbacks when dealing with particular domains, namely HACCP systems [40]. For this systems, two major issues were found:

1. Difficulty in representing specific context details, complete workflow activities, and the semantics of the path selection to be taken during a process instance. Temporal features, identification of quality control and monitoring points and the effects of an activity on the characteristics of a product can't be modeled from a visual nor from a machine-readable perspective.
2. Possible misleading conformance checking results when monitored and non-monitored activities are present in the same model.

To solve this limitations, an already mentioned extension (BPMN-E²) was thought and developed in [40] introducing several new human and machine readable elements that better represent the contextual information of HACCP processes. Note that, despite the priority given to HACCP systems, this notation can also be easily applied to other domains with the same level of richness and expressiveness. In Table 2.1, there can be found the new elements introduced by the BPMN-E² notation.

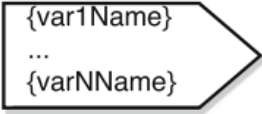
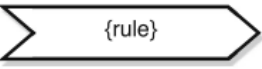

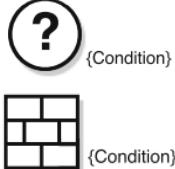
| Elements | Definition | Graphics |
|-------------------------|--|---|
| Monitoring Point | Represents the measurement of a variable or a set of variables in a specific point in the workflow. |  |
| Activity Effect | Represents the activity effect, i.e., how the activity affects a product or how it can change the product characteristics. |  |
| Activity duration | Represents an estimation of the expected execution time of an activity. |  |
| Advanced decision point | Represents a decision point that allows to make clear the reasons involved in a particular choice, connecting the possible choices and paths with the characteristics of the product and the measured variables. It also distinguishes between Normal (above) and Quality (below) decision points. |  |

Table 2.1: New elements introduced by the BPMN-E² notation [40].

These new elements allow for more detailed process models, considering new perspectives that can be advantageous for both Process Modeling and Process Mining tasks.

2.3 SATISFIABILITY MODULO THEORIES

Satisfiability Modulo Theories (SMT) is the problem of deciding the satisfiability of a first-order formula φ with respect to one or more decidable first-order background theory τ [13,42]. A

background theory constraints the interpretations of certain predicate and function symbols. Simply speaking, an SMT problem for φ and τ is the question of whether there is a model of τ that makes φ true.

An SMT solver is a program that implements the corresponding algorithms to automatically determine whether a given formula is satisfiable [15]. An SMT solver distinguishes [11]:

- *Underlying logic*, e.g., first-order, modal, temporal, second-order...;
- *Background theory*, the theory against which satisfiability is checked;
- *Input formulas*, the class of formulas the solver accepts as input;
- *Interface*, the set of functionalities provided by the solver.

For this work, SMT solvers serve an essential role in the definition of a conformance rule (see Chapter 3). Of all the existing background theories, this work's focus will lie upon *Integer arithmetic*, *Real arithmetic* and *Arrays*.

2.4 CLUSTER ANALYSIS

Clustering is a technique that involves sorting cases or variables according to their similarity on one or more dimensions, and producing groups that maximize within-group similarity and minimize between-group similarity [23]. It has been used in several fields including bioinformatics, industrial engineering, marketing, e-commerce and others [10], being used as a exploratory tool to help researchers and organisations to handle large amounts of data.

Clustering methods can be arranged in two different categories: *hierarchical* and *partitional*, each being composed of several individual algorithms with its own strengths and drawbacks. In this dissertation clustering algorithms will be used to implement the *event-based trace clustering for log reduction* technique presented in Section 4. In the following sections, we will introduce and focus on *K-Means* and *density-based spatial clustering of applications with noise* (DBSCAN) algorithms, since these were the ones used in this work. Nevertheless, this technique can be extended to allow for other clustering algorithms

2.4.1 Distance measures

To group multidimensional data, one must be able to quantify the similarities between each data point. *Distance* or *similarity* measures are therefore fundamental components in clustering algorithms [39].

The most used *distance measure* is the Euclidean distance, a special case of the Minkowski metric [39] (where $\alpha = 2$) defined as

$$d^\alpha(z_u, z_w) = \left(\sum_{j=1}^{N_d} (z_{u,j} - z_{w,j})^\alpha \right)^{1/\alpha} = \|z_u - z_w\|^\alpha \quad (1)$$

When $\alpha = 1$, the measure is referred to as the Manhattan distance [39]. Both the Euclidean and the Manhattan measures are appealing [52] however using them to cluster data of high dimensionality can prove ineffective because the distance between the patterns increases with increase in dimensionality. On the other hand, the cosine distance (or vector dot product) - sum of the product of each component from two vectors - is suitable for high dimensional data [39]. There are several more distance measures that can be used for different types of data. Interested readers are referred to [52].

2.4.2 Hierarchical clustering

Hierarchical clustering techniques are used to reveal nested structures of clusters within the data [23]. These techniques generate a cluster tree by splitting clusters into smaller ones (*divisive*) or merging clusters into larger ones (*agglomerative*) [23, 39]. Hierarchical clustering requires the researcher to select a distance metric, which is a unit of measurement for expressing the distances between cases. It also requires the researcher to select a way of defining the link between clusters [23]. The advantages of *hierarchical clustering* are the need not to specify the number of clusters *a priori* and its independence from the starting conditions [39]. However, these algorithms are computationally expensive (time and space), due to the usage and manipulation of the underlying cluster tree (or *dendrogram*). This usually makes *hierarchical clustering* inapt for larger datasets.

2.4.3 Non-hierarchical clustering

Non-hierarchical or *partitional clustering*, on the other hand, produces discrete clusters, by dividing the dataset into a specified number of clusters [23, 39]. These techniques often use an iterative algorithm that converge to an optimal value, trying to *minimize* a certain criteria function locally (over a cluster) or globally (over the entire dataset) [24, 39]. *Partitional clustering* advantages are the disadvantages of *Non-hierarchical clustering*, however it requires a pre-determined number of clusters that can largely impact the final results.

K-means, introduced in [32] is the most used *partitional clustering algorithm*. The aim of *K-means* is to minimize intra-cluster distance [39]. With this method, initial cluster centroids (values representing the average of each cluster on each variable) are manually or randomly assigned. The algorithm then assigns cases to the cluster whose center is nearest, based on

the *Euclidean distance* between them. Assigning the cases in this manner usually changes the cluster centroids, and thus objects are reassigned to clusters and the centroids are updated again. This process continues until no objects change their cluster memberships [23].

DBSCAN is a density based method which can identify arbitrary shaped clusters defined as dense regions separated by low dense regions. *DBSCAN* starts with an arbitrary data point of the dataset and checks for data points within a given radius (ϵ) [46]. A cluster is formed if there are enough data points to do so, otherwise, the data point is marked as *noise*. If a point is found to be a dense part of a cluster, its ϵ -neighbours are also part of that cluster. Hence, all points that are found within the ϵ -neighbourhood are added, as is their own ϵ -neighbourhood when they are also dense. This process continues until the density-connected cluster is completely found. Then, a new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise [45].

2.4.4 Clustering in Process Mining

Several clustering techniques have been used in the field of Process Mining, namely to perform *trace clustering*. In [14], *agglomerative hierarchical clustering* is used to automatically identify *process execution classes* based on selected case attributes. The main *clustering unit* of their work is a *sublog* (i.e., a set of cases). They start by dividing the original event log into smaller subsets and then merge them together based on their similarity until the defined number of clusters is reached. The returned clusters form *sublogs* that can then be individually analysed. Despite considering selected case attributes to initially slice the event log, this approach relies solely on the control-flow distance to compute *sublog similarity*.

In [44], *agglomerative hierarchical clustering*, *K-Means*, *Quality Threshold* and *Self-Organising Maps* were used to perform *trace clustering*. The concept of *trace profile* is introduced as a set of related items that define a trace from a specific *perspective*. Each trace profile can be then aggregated in a *feature vector* and fed to a cluster algorithm. Similarly to [14], this approach aims to return a set of more homogeneous *sublogs* based on the chosen *trace profiles*. The main advantage of this approach is the multi-perspective nature of the *profiles*. As an example, the authors introduce: the *activity profile* - specifying which activities were executed; the *originator profile* - counting how many events have been caused by each originator per trace; the *event profile* - counting how many events are annotated with a specific attribute; amongst others. In [43], the same authors compare several dimensionality reduction techniques to improve the performance of the previous mentioned *trace clustering* technique.

Despite the effort put in *trace clustering*, the motivation of the existent approaches relies on process discovery. It is known that large and complex event logs usually generate hard-to-read *spaghetti-like* models. With *trace clustering* the creation of several *sublogs* can be

leveraged to discover process models out of cases with similar behaviour, thus reducing the process complexity and improving the model readability and comprehensibility.

The approach proposed in this dissertation follows a different route, motivated by the developed *multi-perspective conformance checking* algorithm. Instead of returning several *sublogs* that add up to the original event log, we propose the use of *clustering techniques* to generate a new, downsized event log that maintains the original process' knowledge.

2.5 SUMMARY

In this chapter, several relevant topics and concepts were introduced. Their understanding is necessary to understand the motivations and context in which the proposal lays its foundations. The BPMN-E² notation was presented in the context of the BPMN notation. Furthermore, the current state-of-the-art data-aware conformance checking were introduced. Finally, the area of cluster analysis was , namely its application in the area of Process Mining. Knowing these topics, it is expected a smooth understanding of the upcoming chapters.

DATA-AWARE CONFORMANCE CHECKING

This chapter explains the conformance checking proposal advocated in this work, starting by highlighting its main design goals, detailing then the rational and mechanisms sustaining its development.

3.1 DESIGN GOALS

Considering that BPMN-E² focuses on a process' data flow, the developed conformance checking mechanism must primarily be focused on the data perspective of Process Mining and, thus, providing an efficient and viable way to pinpoint and warn about the following deviations:

- **Inconsistent activity effect**

Assuming the existence of the "Activity effect" element, it should be possible to verify if one or more data variables are being properly affected by an activity.

- **Inconsistent activity duration**

Assuming the existence of the "Activity duration" element, it should be possible to compare the observed activity duration with the expected duration.

- **Wrong path selection**

Assuming the existence of the "Advanced decision point" element, it should be possible to verify if a specific case took the right path according to the values of one or more variables. Moreover, it is also possible to distinguish between "quality" and "normal" decision points, thus enabling taking different measures regarding which type of decision point was broken.

Furthermore, the mechanism should be able to distinguish between monitored and non-monitored activities and act accordingly. Non-monitored activities do not produce event log

records, and so it is important to disregard these activities during conformance checking in order to reduce false negatives and provide more viable results.

To achieve this, a solution was devised considering two phases: 1) *conversion*, from BPMN-E² notation to a more suitable structure; and 2) *conformance checking*, replaying through the event log while checking with the previous structure for eventual non-conformities.

3.2 CONVERSION PHASE

To abstract the conformance checking algorithm from the initial model, there is the need for an intermediate representation of the rules to be followed when replaying the log.

One viable structure that can accurately store this information is a *Directly Follows Model* where the nodes represent the modelled activities and the edges represent a sequence flow between activities. Moreover, each edge can be annotated with the set of rules that must be satisfied when flowing from one activity to another. This annotated structure is here defined as *Directly Follows Rules Model* (DFRM). This way, the conformance checking phase itself is not compromised to a specific notation, providing only that a manual or automatic conversion mechanism to a DFRM is available.

3.2.1 *Directly follows Rules Model*

In [28], *Directly Follows Models* (DFMs) are syntactically described as directed graphs in which the nodes are either an activity, *start* or *end*. Therefore, the language of DFM consists of all traces that can be obtained when flowing from the *start* node to the *end* node.

Definition 1. (*Directly follows model - Syntax*).

Given an alphabet Σ such that $start \notin \Sigma$ and $end \notin \Sigma$, a directly follows rules model is a directed graph (N, E) , such that $N : \Sigma \cup \{start, end\}$ is a set of nodes and $E : N \times N$ is a set of edges.

Considering this, it is possible to extend a DFM by annotating each edge with a set of conformance rules that must be followed when flowing from one node to another.

Definition 2. (*Directly follows rules model - Syntax*).

Given an alphabet Σ such that $start \notin \Sigma$ and $end \notin \Sigma$, a directly follows rules model is a directed graph (N, E, R, A) , such that $N : \Sigma \cup \{start, end\}$ is a set of nodes, $E : N \times N$ is a set of edges, R is a set of rules and $A : E \times R$ is a set of associations between edges and rules.

These rules can then be checked during conformance checking tasks to detect possible non-conformities and deviations during process execution.

3.2.2 SMT Solvers as conformance rules

Satisfiability Modulo Theories (SMT) addresses the problem of deciding the satisfiability of a first-order formula with respect to some background theory. An SMT Solver is a tool for deciding the satisfiability of formulas in these theories [20].

Attending to its logical nature, the problem of verifying a conformance rule can be seen as an SMT problem considering, in this case, the following background theories: *Arithmetic*, *Real* and *Arrays*. Consequently, a conformance rule can be mapped into an SMT Solver instantiated with an initial set of *assertions*. Therefore, a conformance rule can be seen as an SMT Solver instantiated with an initial set of assertions that refer to the conditions of a BPMN-E² element (see Figure 3.1). This assertions are extracted directly from the extension elements being converted from their BPMN-E² representation to an equivalent *SMT-Solver compatible* expression (for this dissertation Z₃ and its python interface - Z₃Py - were used).

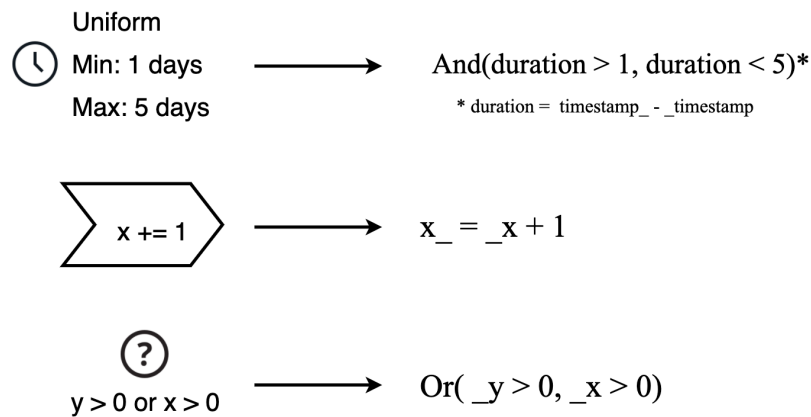


Figure 3.1: Example of conversions from different BPMN-E² extension elements. Underscores (_) are used to distinguish between attributes belonging to the source activity (prefix) and the target activity (suffix).

During conformance checking, these rules can be tested by adding a set of new assertions based on the event log data and solving the SMT problem. In case of satisfiability, the rule is also satisfied, conversely, in case of unsatisfiability, the rule is not satisfied. It is also important to store the type of the conformance rule in order to provide a more detailed conformance checking report. Therefore, for each new BPMN-E² element that generates a conformance rule, a corresponding type is assigned.

```

# Create conformance rule
temperature = Int('temperature')
s = Solver()
s.add(temperature > 7)

# Verify rule using log's temperature value
s.add(temperature == 9)
s.check() # satisfied

```

Listing 1: Process of rule verification

In the code snippet in Listing 1, the process of verifying a rule is demonstrated¹. The SMT Solver is initialised with an assertion (`temperature > 7`), this represents the conformance rule to check. Then an equality assertion is added (`temperature == 9`), this represents the actual value provided by the event log. Finally, the rule is checked. In this case, since the provided temperature value is bigger than 7, the conformance rule is satisfied.

3.2.3 Dealing with non-monitored activities

One of the biggest advantages of BPMN-E² notation is the identification and distinction of monitored and non-monitored activities, thus providing a way to graphically represent processes with partially monitored activities without influencing conformance-checking results. Considering, for instance, the non-monitored activity *A27* modelled in Figure 1.1, traditional conformance checking approaches would be expecting records of this activity execution in an event log, which would lead to the expected, yet incorrect, identification of a conformance error.

To overcome this drawback, non-monitored activities must be filtered during the conversion phase assuring that the inputs of non-monitored activities become connected to the corresponding outputs. The automation of this process allows to maintain the consistency of the diagram and to prevent the loss of information in the workflow, both of these problems were addressed in [40].

3.2.4 Associating rules to a DFRM

With DFRMs and SMT Solvers as conformance rules in mind, the conversion phase consists of parsing the BPMN-E² source file whilst converting the new elements into SMT Solver's assertions and associating them with the respective edge of the DFRM. This raises the question “*To what activities should conformance rules be associated with?*”. The answer is bound to the way event data is recorded:

¹ The example is written in Python using Z3 solver.

1. Events are recorded when the activity starts.

In this case, when flowing from an activity A to an activity B only extensions elements of A and of the sequence flow A→B should be included (see Figure 3.2).

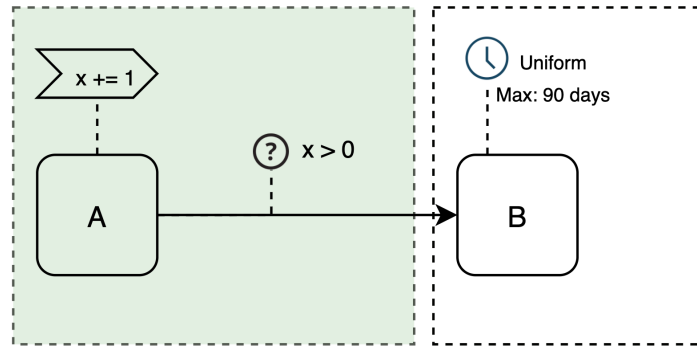


Figure 3.2: Rule association when event data is recorded at the start of an activity.

2. Events are recorded when the activity ends.

In this case, when flowing from an activity A to an activity B only extensions elements of B and of the sequence flow A→B should be included (see Figure 3.3).

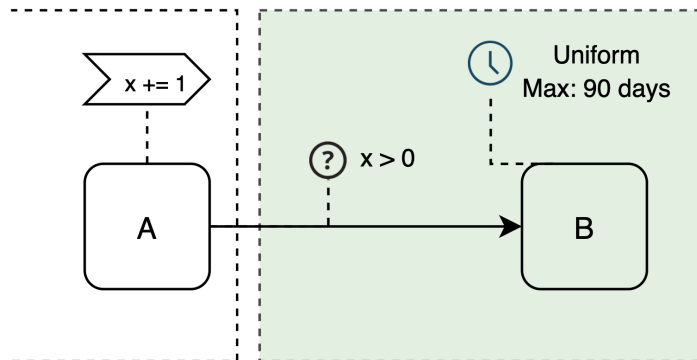


Figure 3.3: Rule association when event data is recorded at the end of an activity.

3. Events are recorded when the activity starts and ends.

This particular case differs from the others since every activity is recorded twice (in the beginning and at the end). This way, when flowing from an activity A to an activity B only extensions elements of the sequence flow A→B should be included. Analogously, when flowing from A:start to A:end and from B:start to B:end only extensions elements of A and B should be included, respectively (see Figure 3.4).

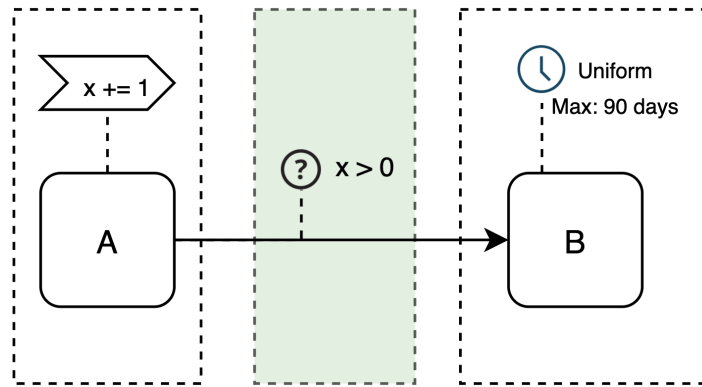


Figure 3.4: Rule association when event data is recorded at the start and at the end of an activity.

In the context of this dissertation, it was considered that event data is recorded at the end of the activities, since that is the most common recording method [50]. An example of a conversion process' input and output is illustrated in Figure 3.5.

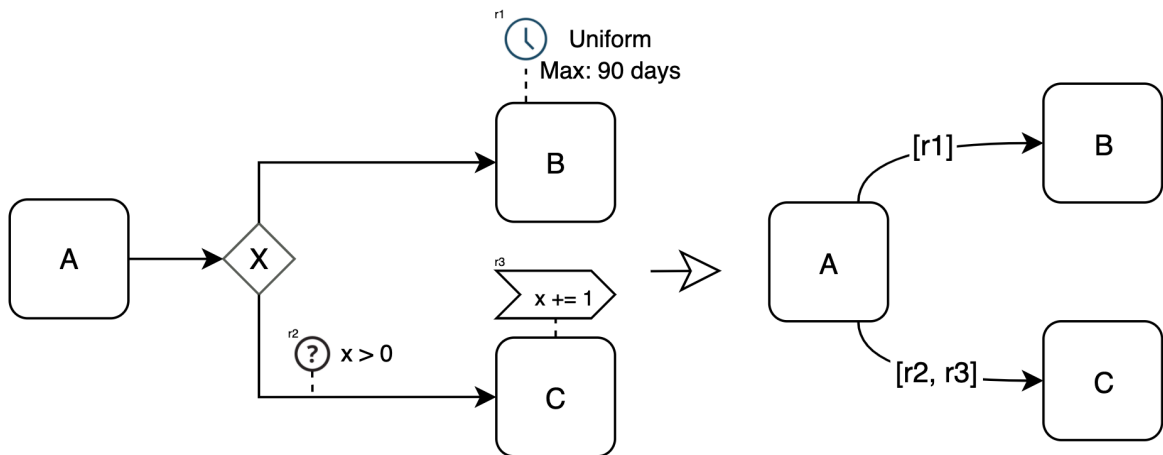


Figure 3.5: Excerpt of a BPMN-E² model, extended with advanced decision points, activity durations and activity effects (left), and the respective DFRM graph after conversion (right).

3.3 CONFORMANCE CHECKING PHASE

The second phase consists of the actual conformance checking algorithm. This algorithm will receive an event log and a previously generated DFRM as inputs, and will either produce a fitness value or a detailed report concerning the data-flow of the process as output.

The event log is parsed case-by-case, activity-by-activity keeping track of the most recent attribute updates. Activities can create and alter these attributes. By comparing changes in their values, it is possible to: 1) extract activity's duration; 2) extract activity's effect; and 3) extract the path that was taken. This data is then fed to the previously generated DFRM,

being used to check the satisfiability of the respective set of conformance rules. All detected non-conformities are recorded, storing the conformance rules that were broken, the cases they occurred in, and the activities and the attributes responsible for them. In this way, it is possible to produce a report regarding [40]:

- *Time* - stating the correspondence between the theoretical time constraints and the real time taken by the activities (measured in the event log timestamps);
- *Activity effects* - stating the expected activity effect over a process instance with the real effect carried out;
- *Quality points* - stating the fulfillment of checkpoints on the workflow (checking whether an instance has followed the correct path).

Alongside these reports, it is also possible to detect control-flow deviations when the sequence flow observed in the event log does not exist in the DFRM. However, there are better and more complete methods to tackle control-flow conformance-checking (namely, *alignments*). This is why, a combination of state-of-the-art *alignment-based* methods with the proposed approach is advised to provide a richer analysis regarding both control and data flow perspectives.

There are four quality criteria (*fitness*, *precision*, *generalisation* and *simplicity*) that can be used to assess the quality of a given model [9, 37, 50]. Here, a way to compute the *fitness* of a model is proposed, since it measures the proportion of the event log's valid behaviour according to the model [50]. Firstly, the fitness of an individual trace τ is considered to be the percentage of rules that were satisfied. This way, assuming a BPMN-E² model, the fitness of a given trace can be computed simply using Eq. (2)

$$f(\tau) = \begin{cases} \frac{\textit{satisfied}}{\textit{total}} & , \textit{if total} > 0 \\ 1 & , \textit{if total} = 0 \end{cases} \quad (2)$$

where *satisfied* is the number of conformance rules that were satisfied and *total* is the total number of conformance rules that were tested. This formula can be extended to support weighting deviations according to their importance and severity by using Eq. (3) instead.

$$f(\tau) = \begin{cases} \frac{\sum_{\zeta \in \textit{vrules}} \textit{weight}(\zeta)}{\sum_{\zeta \in \textit{trules}} \textit{weight}(\zeta)} & , \textit{if trules} \neq [] \\ 1 & , \textit{if trules} = [] \end{cases} \quad (3)$$

where *vrules* is the list of conformance rules that were satisfied, *trules* is the list of conformance rules that were tested and *weight* is a function that retrieves the weight for a particular conformance rule.

At the event log level, fitness can be computed by averaging the fitness values of each log trace. For an event log l and a model m , the *fitness* is computed using Eq. (4):

$$f(l, m) = \frac{\sum_{\tau \in l} f(\tau, m)}{\text{length}(l)} \quad (4)$$

Note, in Eq. (3), that if the trace leads to no conformance rule verification, it is considered as if there are no data-flow deviations. However, it can be the case that the trace does not follow the correct control-flow in the first place, leading to certain rules not being tested. With this in mind, interested readers are encouraged to complement the proposed approach with state-of-the-art “traditional” conformance checking techniques, such as *token replay* and *alignments*. This “hybrid” approach will provide insights on both control and data-flow that can be leveraged to attain a more complete conformance analysis. As an example, an overall fitness can be computed by combining the results from both approaches, as defined in Eq.(5):

$$\text{fitness}(l, m) = w_t \cdot f_t + w_{cr} \cdot f_{cr} \quad (5)$$

where w_t and f_t are the weight and the fitness values of a state-of-the-art “traditional” conformance checking approach, and w_{cr} and f_{cr} are the weight and the fitness values of the proposed approach.

3.4 SUMMARY

In this chapter, the data-aware conformance checking algorithm was explained. The rationale behind the proposal was presented, as well as its different building blocks. In essence, a new structure - DFRM - was introduced. They leverage the new BPMN-E² elements by mapping them into conformance rules (as abstractions of SMT Solvers). These rules are then verified at runtime while maintaining a record of which conformance rules are met and which ones are not. Finally, a way to compute *fitness* based on the obtained results was also discussed.

The development of this algorithm, entailed efforts of designing a way to reduce its execution time. However, the one variable that weighted the most on this matter was the size of the log, meaning that bigger logs would require more processing time. With this in mind, an event-based trace clustering algorithm for log reduction was developed.

EVENT-BASED TRACE CLUSTERING FOR LOG REDUCTION

This chapter introduces the event-based trace clustering technique used in this work to downsize large event logs into a smaller one. Section 4.1 points the motivations for the development of this technique. Sections 4.2 and 4.3 explain in detail how the technique functions.

4.1 MOTIVATION

One of the main drawbacks of multi-perspective conformance checking techniques compared to the more common control-flow approaches is that considerably larger amounts of traces must be analyzed at runtime, which unavoidably leads to longer execution times for bigger event logs. As an example, let's assume an event log with a thousand different cases that span across ten trace variants, control-flow approaches only need to compute the fitness for these ten variants and map it to the according cases. On the other hand, however, adding the data perspective leads, in the worst case scenario, to a number of trace variants equal to the total number of cases. Therefore, data-flow approaches oftentimes need to compute the fitness for every single case in the event log.

With this in mind, and acknowledging that there is no “workaround” to this reality at the algorithm level (the conformance checking task will always compute fitness for all individual event log cases), a way to downsize the event log while retaining the most relevant cases (data wise) was thought of. This can be done by sampling cases with distinct data-flow behaviour and excluding those with similar behaviour.

In this scenario, given its importance in pattern recognition [39], cluster analysis can be leveraged to group cases based on the values of different event attributes and their changes during a process execution.

| case:concept:name | concept:name | time:timestamp | amount | expense | paymentAmount | totalPaymentAmount | dismissal |
|-------------------|--------------|---------------------------|--------|---------|---------------|--------------------|-----------|
| A10005 | Create Fine | 2007-03-20 00:00:00+01:00 | 36.0 | - | - | 0.0 | NIL |
| A10005 | Payment | 2007-03-21 00:00:00+01:00 | - | - | 36.0 | 36.0 | - |
| N28738 | Create Fine | 2000-09-13 00:00:00+02:00 | 31.0 | - | - | 0.0 | NIL |
| N28738 | Send Fine | 2000-10-30 00:00:00+01:00 | - | 6.71 | - | - | - |
| N28738 | Payment | 2000-12-05 00:00:00+01:00 | - | - | 38.01 | 38.01 | - |

Table 4.1: Excerpt of the event log used in Section 6. Used here to illustrate the vectorization of event log traces.

| case | Create Fine | | | Send Fine | | Payment |
|--------|-------------|---------------|-----------|-----------|---------------|---------------|
| | amount | paymentAmount | dismissal | expense | paymentAmount | paymentAmount |
| A10005 | 36.0 | 0.0 | o (NIL) | - | - | 36.0 |
| N28738 | 31.0 | 0.0 | o (NIL) | 6.71 | 36.0 | 38.01 |

Table 4.2: Vectorized traces extracted from Table 4.1, using the attributes *amount*, *dismissal*, *paymentAmount* and *expense*.

4.2 VECTORIZING A TRACE

A clustering algorithm operates over a number of data points called *feature vectors* [39]. In turn, a *feature vector* is composed of several *features* (or *attributes*). To be able to execute clustering algorithms over event log traces, they must be transformed into a corresponding *feature vector*, with its *features* being the event's attributes.

A vectorized trace can then be defined as an object whose items are the attributes of all the log's events and its values are the corresponding attribute's data for that particular trace. In order to vectorize a trace, the following steps are taken:

1. Assess which attributes should be present in the *feature vector*. Optionally, *timestamps* can be leveraged to compute an activity duration, thus taking time perspective into consideration during clustering.
2. Create a *feature vector* for each trace's event containing the chosen attributes' values. If the particular event does not contain one or more of the selected attributes, they can simply be omitted.
3. Append all of the previous generated event *feature vectors* to form the final *feature vector*.

The process of trace vectorization is illustrated in Tables 4.1 and 4.2.

4.3 CLUSTERING AND SAMPLING

The similarity between the different *feature vectors* is computed based on a *distance measure*, used to evaluate how close two *vectors* are from each other. There are several *distance measure functions* that can be used to assign a level of similarity between to *data points*, such as,

Euclidean distance, city block distance, Minkowski distance, Canberra distance, cosine distance, amongst others [52]. Onward we will use the *Euclidean distance* (described in Eq 1), since it is the most used *distance measure*, however keep in mind that others can yield better results depending on the underlying problem.

We will not delve into the inner works of clustering algorithms. Interested reader are referred to [24]. Instead we assume their output as a group of clusters, each one representing an *hard partition* of the original data with close proximity in regards to a number of *features*. The underlying motif of using clustering analysis to perform *event log* reduction lies in considering that all the traces belonging to a cluster provide the same amount of process knowledge as its neighbours and, therefore, can be removed from the original event log while keeping an accurate representation of how the process was conducted - this is hereby defined as *trace sampling*. An high level view of this process can be seen in Figure 4.1.

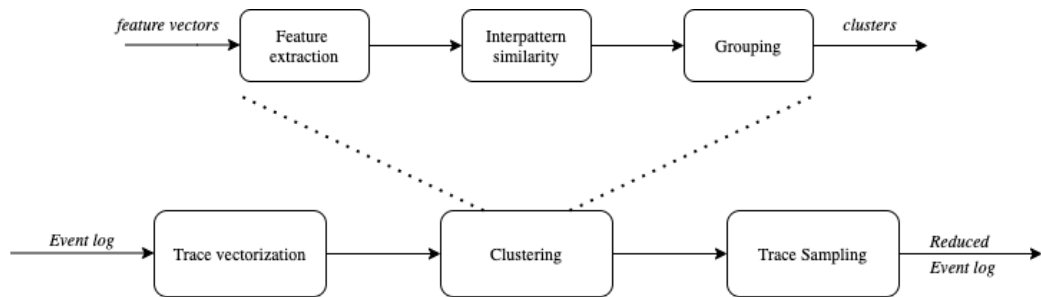


Figure 4.1: Stages of *Event-based trace clustering for log reduction*. Clustering stages were taken from [24].

It is natural that the clusters have different sizes, containing less (or more) cases than the others. To assure the reduced event log's distribution of event behaviour remains unaltered from the original's, the amount of sampled cases must be proportional to the cluster's size. This way, the reduced event log keeps, not only the original's data-flow behaviour, but also the proportion in which this behaviour occurs. Take, as an example, an event log with 10 different cases, where 8 of them follow a specific data-flow and the remaining 2 follow a different data-flow. Consider now a 50% reduction is necessary, it is important that the final event log maintains this ratio of 8:2, by sampling 4 cases from the first cluster and 1 case from the second cluster.

The main downside of the event log reduction is the lost of process information due to the exclusion of several traces from the original event log. Even though the most representative cases are kept in the reduced version there can oftentimes exist traces that behave "oddly" compared to the rest (the so called *outliers*). These *outliers* pose two drawbacks: a) it can happen that they are assigned to clusters that do not reflect their actual behaviour and end up being excluded; b) *outliers* are known to negatively impact certain clustering algorithms disabling them from correctly identify clusters [21]. With this in mind, one should evaluate the process data available and judge if the benefits of having a reduced event log make of

for the loss of trace information. As a rule of thumb, log reduction can prove useful when individual traces can be neglected by generalisation, such as, discovering a process model from an event log or computing conformance checking fitness and precision. If the context calls for a mandatory analysis on each trace to pinpoint deviations and its causes, a reduced event log would naturally limit the analysis by omitting several cases. Nevertheless, even in those cases, the reduced version can be used as a starting point to provide a general picture of the problem at hands that could later be extended if a more thorough analysis is needed.

This technique was mainly developed as a solution to the long execution times experienced during the previously proposed conformance checking technique, however since it operates at the log level, one can use it to downsize any event log and apply it to other data-aware Process Mining techniques, namely conformance checking, process discovery and process enhancement.

4.4 SUMMARY

In this chapter, the event-based trace clustering for log reduction algorithm was explained. Framed in the context of this dissertation's purposal and inspired by other trace clustering techniques, its main goal was the transformation of existing event logs by sampling the cases that better generalise the business process. The proposed technique relies on three stages: *trace vectorization* - pre-process the event log, based on user-selected criteria, to feed it to *state-of-the-art* clustering algorithms ; *clustering* - group the previously created trace vectors into heterogeneous clusters; *sampling* - select a percentage of cases from each of the computed clusters. It is expected that each cluster contains cases that behave similarly in regards to the selected attributes and, therefore, can be seen as redundant for certain Process Mining tasks.

Although it was developed in the context of the data-aware conformance checking algorithm proposed in the previous chapter, this technique was designed to work independent of any Process Mining technique and therefore can, and should be, leveraged as a tool for event log processing.

IMPLEMENTATION

This chapter discusses the development of the techniques introduced in Chapter 3 and 4. Nowadays, both open-source (e.g., ProM and Apromore) and commercial (e.g., Disco, Celonis, ProcessGold, etc.) tools are available for executing Process Mining tasks. However, these tools are either restricting the use of custom algorithms (commercial tools) or relying on standalone programs (open-source and commercial tools) that difficult the usage of Process Mining in a more experimental setting. With this in mind, the library Process Mining for Python (PM4Py) was built, lowering the barrier for algorithmic development and allowing for easy integration of Process Mining algorithms with state-of-the-art Python packages [6, 12]. Considering this increasing number of Python libraries that support *Process Mining*, *Process Modeling* and *Data Science*, Python was chosen to build both the conformance checking and the trace clustering algorithms, thus taking advantage of the existing libraries and further contributing to its ecosystem by providing utilities for the BPMN-E² notation and the proposed conformance checking approach. Section 5.1 explains the code design and architecture and Section 5.2 lists the library main functionalities and highlights important development decisions made in the conversion and conformance algorithms.

5.1 ARCHITECTURE

The developed library is structured following the class diagram in Figure 5.1, with the developed library being organised in five main packages:

- **Objects**

Contains all the classes and utilities needed to define the constructs introduced in Section 3.

- **Conformance**

Contains all the classes and utilities needed to execute the conformance checking task defined in Section 3.

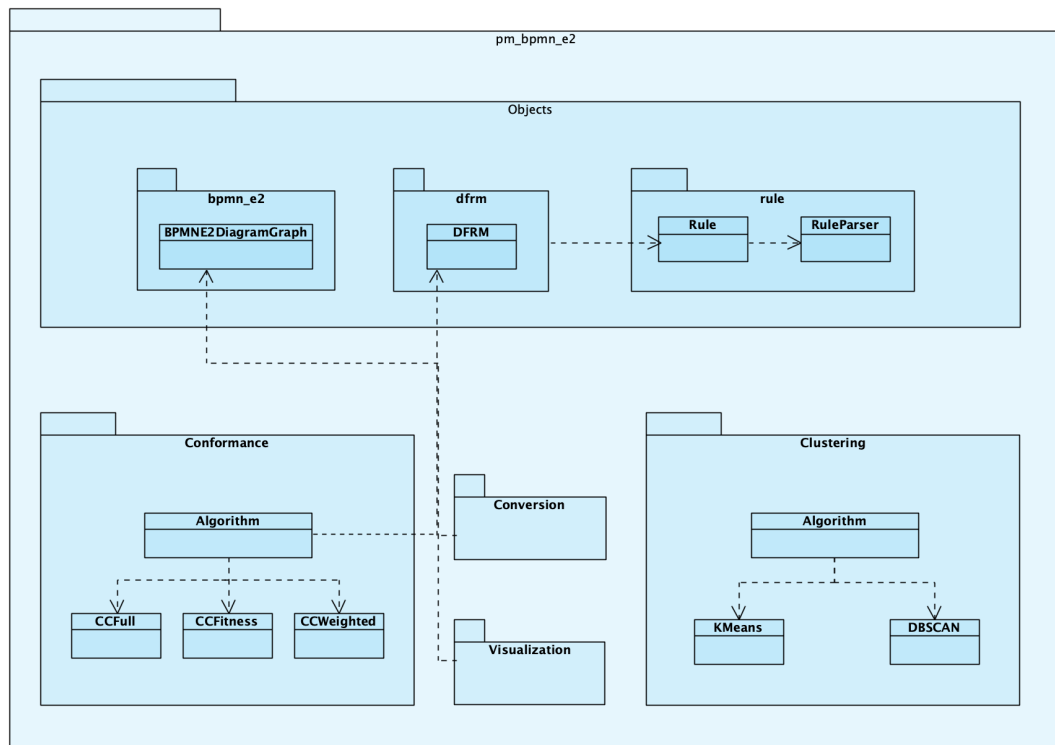


Figure 5.1: Class diagram showing of the PM_BPMN_E2 library.

- **Clustering**

Contains all the classes and utilities needed to execute the event-based trace clustering algorithm defined in Section 4.

- **Conversion**

Contains utilities for converting a BPMN-E² diagram into its respective DFRM.

- **Visualization**

Contains all the utilities needed for visualising both BPMN-E² diagrams and DFRMs, using *Graphviz* [4].

At the class level, it is possible to identify several relevant classes, them being:

- **BPMNE2DiagramGraph**

Defines a BPMN-E² diagram. Extends *PM4Py*'s BPMN class by providing support to define and import the new BPMN-E² extension elements.

- **DFRM**

Defines a *Directly Follows Rule Model*. It uses a *NetworkX* [5] directed graph as the underlying main structure and provides several methods to create and modify it by adding and removing *nodes*, *edges* and *rules*.

- **Rule**

Defines a conformance rule. Relies on Z3Py (Z3's Python API) as the underlying SMT Solver and provides an interface to add assertions and check if the rule is met given a certain criteria.

- **RuleParser**

Wrapper class around an ANTLR4 grammar responsible for the interpretation and conversion from the BPMN-E2 extension elements notation to Z3's input format.

- **Conformance.algorithm**

Implements the conformance checker algorithm explained in Section 3. It follows the *strategy behavioural design pattern* allowing to choose between different conformance checking strategies. It also leverages PM4Py's event log utilities for importing event logs from XES file format. The machine learning library *scikit-learn* [8] is used to execute the different clustering algorithms.

- **Clustering.algorithm**

Implements the clustering algorithm explained in Section 4. It follows the *strategy behavioural design pattern* allowing to choose between KMeans and DBSCAN clustering algorithms. It also leverages PM4Py's event log utilities for importing event logs from *eXtensible Event Stream* (XES) file format and to convert event logs into pandas dataframes.

5.2 MAIN FUNCTIONALITIES

The developed library has the following main functionalities:

- Import BPMN-E² diagrams;
- Convert BPMN-E² diagrams to their respective DFRMs;
- Visualise BPMN-E² diagrams and DFRMs using Graphviz;
- Execute the proposed data-aware conformance checking task;
- Produce pandas dataframes based on conformance checking results;
- Produce HTML reports regarding detected non-conformities (Figure 6.1);
- Execute the proposed event based log reduction task.

This section highlights the main development and algorithmic choices for conversion, conformance and clustering techniques.

5.2.1 Conversion

The conversion algorithm introduced in Chapter 3 transforms a BPMN-E² structure into a DFRM structure. To better understand how the conversion is done it is important to have a general understanding of how this two structures are built.

BPMN-E²

Since the BPMN-E² notation presents itself as an extension to the known BPMN notation, its implementation is built on top of an already developed BPMN Python library (bpmn-Python [1]). As can be seen in Figure 5.2, `BPMNE2DiagramGraph` class extends `bpmn-Python:BPMNDiagramGraph` class by including two new instance variables that store the new elements proposed by the new notation: `extension_elements` and `associations`. It also extends the class method `load_diagram_from_xml()` to import the new elements and adds two new class methods `view` and `save` to print the graph and to save it as *png*, respectively.

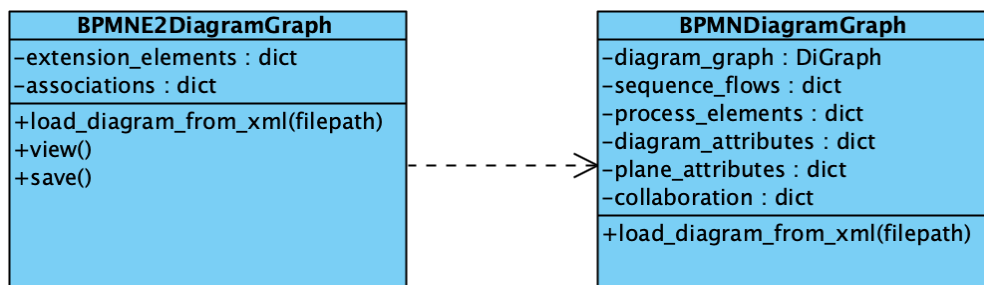


Figure 5.2: `BPMNE2DiagramGraph` class diagram .

DFRM

The DFRM class on the other hand, is composed of only one instance variable - `graph` - that represents the DFRM itself (see Figure 5.3). It is also composed of several class methods that enable the creation and manipulation of the *graph structure*. Methods `add_nodes`, `add_edges()` and `add_rules()` are self-explanatory and enable the creation of the main DFRM's elements. Method `verify_rules()` is used to verify all the rules of a given edge. Finally, methods `view` and `save` are used to print the graph and to save it as *png*, respectively.

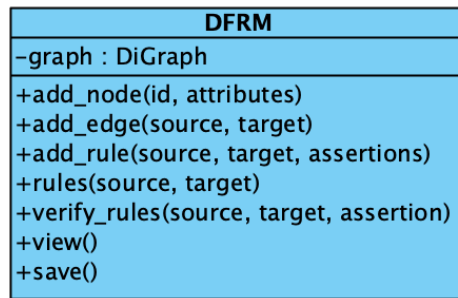


Figure 5.3: DFRM class diagram.

With both these structures cleared out, it is now possible to explain how the conversion algorithm was thought and developed.

```

1 Function bpmn_e2_to_dfrm(bpmn_e2):
2   dfrm ← new DFRM()
3   mapping ← gateway_mapping(bpmn_e2)
4   add_nodes(dfrm, bpmn_e2)
5   add_edges(dfrm, bpmn_e2, mapping)
6   add_rules(dfrm, bpmn_e2, mapping)
7   remove_non_monitored(dfrm)
8   return dfrm

```

As can be seen, there the algorithm is divided in five main steps:

1. Create a gateway mapping. This auxiliary structure maps a gateway's input to its output. It is intended to allow for correctly creating edges and assigning *decision* rules.
2. Add nodes. Copies *non-gateway* elements from BPMN-E² to DFRM.

```

1 Function add_nodes(dfrm, bpmn_e2):
2   foreach node in bpmn_e2.nodes do
3     if node.type is not gateway then
4       |   add node to dfrm

```

3. Add edges. Connects DFRM nodes based on BPMN-E² sequence flows. Since DFRM excludes *gateway* elements, a gateway mapping is needed to correctly connect nodes.

```

1 Function add_edges(dfrm, sequence_flows, mapping):
2   foreach flow in sequence_flows do
3     edge_sources ← sources from mapping
4     edge_targets ← targets from mapping
5     edges ← connect edge_sources to edge_targets
6     add edges to dfrm

```

4. Add rules. Assigns conformance rules to the respective DFRM edges. For each BPMN-E² element, a rule object is initialised based on its attributes. Then, the rule is added to the respective edge. The edge is obtained by finding both the *non-gateway* source and target nodes the rule element was bound to.

```

1 Function add_rules(dfrm, bpmn_e2, mapping):
2   foreach element in bpmn_e2.elements do
3     rule ← create rule from element
4     rule_target ← find bpmn_e2's element the rule is bound to
5     predecessors ← find rule_target's predecessors elements
6     add rule to dfrm by binding it to the edges (rule_target, predecessors)

```

5. Remove non-monitored activities. Finally, exclude all non-monitored activities by removing their respective nodes and rules while redirecting its edges from its predecessor nodes to its successor nodes. This step is optional.

```

1 Def remove_non_monitored(dfrm):
2   foreach node in dfrm.nodes do
3     if node is non-monitored then
4       remove node from dfrm
5       remove node's connected edges
6     add new edges by connecting node's predecessors to its successors

```

5.2.2 Conformance Checking

The conformance checking algorithm follows the procedures mentioned in Section 3 strictly. It starts by converting the BPMN-E2 and initialising both the results and attributes structures. It then runs through every event log's case and: 1) verifies the appropriate conformance rules, 2) stores the obtained results and 3) updates the attributes with the more recent values.

```

1 Function conformance(event_log, bpmn_e2):
2   dfrm ← bpmn_e2_to_dfrm(bpmn_e2)
3   results ← []
4   attributes ← initialize attributes from bpmn_e2 monitored variables
5   foreach case in event_log do
6     foreach event in case do
7       event_results = verify_rules(dfrm, attributes, event)
8       results.append(event_results)
9       update attributes with event
10  return results

```

The process of verifying a rule is straightforward. Considering that *attributes* stores the most recent event's attributes and *event* contains the attributes of the event currently being parsed, the first step is to fetch from the DFRM the conformance rules that belong to the transition from the previous to the current event. Then, each rule is verified by comparing the changes in the events' attributes.

```

1 Function verify_rules(dfrm, attributes, event):
2   source ← case id from attributes
3   target ← case id from event
4   results ← []
5   rules ← dfrm.rules(source, target)
6   foreach rule in rules do
7     assertion ← z3 assertion from source and target
8     rule_result = rule.verify(assertion)
9     results.append(rule_result)
10  return results

```

5.2.3 Event-based trace clustering for log reduction

The log reduction algorithm proposed in Section 4 is composed of three main steps. The first one is intended to prepare the event log data for clustering. This step starts by converting the *PM4Py's event log structure* to a *pandas Dataframe* for further and better processing. It continues by removing the columns that were not specified by the user for clustering, next categorical types are converted to numeric and a new column "duration" is added (only if the user wants to take activity duration into consideration). Finally, each trace is vectorized (as discussed in Section 4) and its data normalized. The second phase consists in applying the clustering algorithm chosen by the user. For each resulting cluster N case indices are sampled based on the parameters given by the user. These sampled indices are then used on

the last phase to point which cases are kept in the dataframe and which ones are not. The dataframe is then converted in the *PM4Py's event log structure*.

```

1 Function trace_clustering(event_log, cluster_by):
2   df ← dataframe from event_log
3   df_copy ← maintain a copy from original dataframe
4   keep df columns that are in cluster_by
5   convert df's categorical types to numeric
6   add duration column to df
7   vectorize trace
8   normalize df data
9   clusters ← apply state-of-the-art clustering algorithm
10  selected_indexes ← []
11  foreach cluster in clusters do
12    selected_cases ← sample n cases from each cluster
13    append selected_cases indexes to selected_indexes
14  reduced_df ← filter df_copy on selected_indexes
15  reduced_log ← convert reduced_df to PM4Py's Event Log Structure
16  return reduced_log

```

5.3 SUMMARY

In this chapter, the implementation of the algorithms proposed in Chapters 3 and 4 were discussed. The developed library was written using Python programming language due to its increasing support of areas like data science and process mining. It recycles the event log representation of the library *PM4Py*, extends the BPMN diagram representation of the library *bpmn-python* to accommodate the new elements of the BPMN-E² notation and defines the DFRMs and conformance rules rules objects as discussed in the previous chapters. Most important, it provides a faithful materialization of both the proposed data-aware conformance checking and the event log reduction algorithms.

In the following chapter, the developed library will be used on a real-life process, guiding the reader on how to leverage its functionalities to conduct a Process Mining analysis. The performance of both algorithms will also be evaluated.

PROOF OF CONCEPT

The main goal of the developed conformance checking task is to provide detailed insights on inconsistencies found regarding three main aspects: a) activity effects; b) activity duration; and c) path selection. To achieve this, the results of verifying every conformance rules are recorded alongside contextual data regarding the moment the rule was being verified, including the case, the activities and the event log values being processed. This data can then be processed in multiple ways to meet the analytical goals of the users. Using the developed library, an out-of-the-box HTML report similar to the one in Figure 6.1 can be generated, containing several sections providing information and visualisations regarding the process model and the event log, and fitness values (at case, rule and type level). It is also possible, and encouraged to output the results using dataframes enabling a more customised and powerful data analysis meeting the user's business needs.



Figure 6.1: Example of a conformance checking HTML report.

In this chapter, the proposed conformance checking technique is tested on a real-life scenario and evaluated using synthetic data. It starts, in Section 6.1, by providing a real-life

use case analysis and showing how the developed library can be leveraged to perform multi-perspective conformance analysis. In Section 6.2, time complexity and performance are assessed using synthetically generated data. The event log reduction technique introduced in Chapter 4 is also evaluated in this section.

6.1 REAL-LIFE USE CASE

At the moment of the writing, there was no real-life BPMN-E² use case that could be leveraged to provide a scenario where both a BPMN-E² and the conformance checking task could prove useful. Nevertheless, this was seen as an opportunity to find a real-life situation where the proposal of this dissertation could be applied to gather significant insights.

For this, a real-life event log taken from an information system of the Italian police [34] was used.

6.1.1 Event log

As stated in [34], the event log respects to a road traffic fine management process regarding the creation, payment and appeal of fines. The event log is rich in attributes that can be leveraged to support a data-aware conformance analysis. After a thorough analysis we verify that the log is composed of 561.471 events recorded across 145,800 event traces, which were recorded between January 2000 and June 2013.

| case:conceptname | concept:name | time:timestamp | amount | expense | paymentAmount | totalPaymentAmount | dismissal |
|------------------|----------------------------------|---------------------------|--------|---------|---------------|--------------------|-----------|
| A10004 | Create Fine | 2007-03-20 00:00:00+01:00 | 36.0 | - | - | 0.0 | NIL |
| A10004 | Send Fine | 2007-07-17 00:00:00+02:00 | - | 13.0 | - | - | - |
| A10004 | Insert Fine Notification | 2007-07-24 00:00:00+02:00 | - | - | - | - | - |
| A10004 | Add penalty | 2007-09-22 00:00:00+02:00 | 74.0 | - | - | - | - |
| A10004 | Send for Credit Collection | 2009-03-30 00:00:00+02:00 | - | - | - | - | - |
| A10005 | Create Fine | 2007-03-20 00:00:00+01:00 | 36.0 | - | - | 0.0 | NIL |
| A10005 | Payment | 2007-03-21 00:00:00+01:00 | - | - | 36.0 | 36.0 | - |
| A16149 | Create Fine | 2007-10-24 00:00:00+02:00 | 36.0 | - | - | 0.0 | NIL |
| A16149 | Send Fine | 2008-01-18 00:00:00+01:00 | - | 13.0 | - | - | - |
| A16149 | Insert Fine Notification | 2008-01-31 00:00:00+01:00 | - | - | - | - | - |
| A16149 | Insert Date Appeal to Prefecture | 2008-02-12 00:00:00+01:00 | - | - | - | - | - |
| A16149 | Add penalty | 2008-03-31 00:00:00+02:00 | 74.0 | - | - | - | - |
| A16149 | Send Appeal to Prefecture | 2008-03-31 00:00:00+02:00 | - | - | - | - | # |

Table 6.1: Excerpt of the road traffic fine management process' event log.

As seen in Table 6.1, one can identify the following attributes:

- *totalPaymentAmount* - Total amount already payed by the offender;
- *paymentAmount* - Amount payed by the offender during the "Payment" event;
- *amount* - Monetary value of the fine;
- *dismissal* - Flag that indicates possible cause for a fine dismissal. "NIL" if the fine is not dismissed, "G" if it is dismissed by the prefecture and "#" if it is dismissed by a judge;

- *expenses* - Monetary value of extraordinary expenses such as the cost of sending the fine via mail.

6.1.2 BPMN-E2 Process Model

The business process model was manually designed considering the petri net illustrated in [34].

The general process flow is as follows. When a road traffic offence occurs, a fine is created. The offender can then pay the fine partially or in full at several stages of the process. The fine management is closed when the offender pays the full amount. If the process is still open, a fine notification will be sent to the offender's residency, after which he can choose to appeal of the decision (to a judge and/or to the prefecture). If the fine is not paid before 180 days a penalty is added to the current fine's amount. If the fine is still not paid, it will eventually end by handing over the case for credit collection.

This process can then be enhanced with several BPMN-E² elements to provide a data-flow perspective at the activity level (see Table 6.2), resulting in the final BPMN-E² model, as seen in Figure 6.2.

Table 6.2: Rules extracted from process' description.

| Activity | Rule type | Description |
|---|-----------------|---|
| Send Fine | <i>Duration</i> | Fine notification should be send until 60 days after fine creation. |
| Appeal to Judge | <i>Duration</i> | Offender has 60 days to appeal to judge after being notified. |
| Add Penalty | <i>Effect</i> | When a penalty is added the fine's payment amount must increase |
| <i>Any</i> → Insert for credit collection | <i>Decision</i> | The fine going to credit collection, implies that the offender did not pay the fine's full amount |
| Send Appeal to Prefecture → end | <i>Decision</i> | If the appeal to prefecture succeeds, the fine is dismissed with flag "G" |
| Appeal to Judge → end | <i>Decision</i> | If the appeal to the judge succeeds, the fine is dismissed with flag "#" |
| <i>Any</i> → end | <i>Decision</i> | The process should only end when dismissed or when the fine is fully paid |

In addition, BPMN-E² formally specifies that is mandatory to define a *Monitoring Point* for all monitored activities. Therefore, all activities were extended with a *Monitoring point*. However, most of these elements were left out from Figure 6.2 for comprehensibility purposes. The resulting diagram, constitutes a centralized and easy-to-understand source of knowledge related to the process being analyzed.

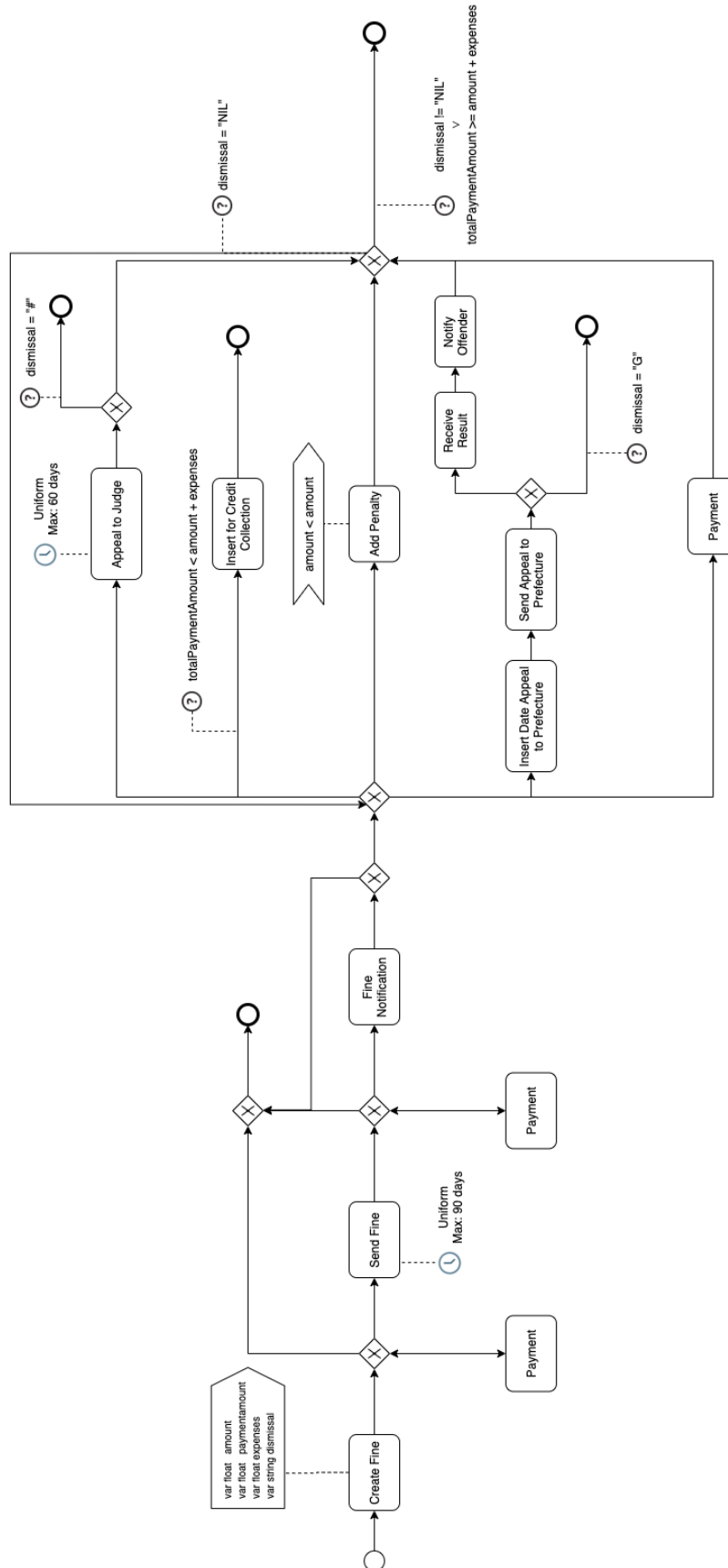


Figure 6.2: BPMN-E² model of the road traffic fine management process.

6.1.3 Analyzing the results

Initially, the conformance checking task was used to resolve an overall fitness value to assess how well the event log fit the process model. In average, 3.68 conformance rules were tested per case, reaching a total of 554.291 tested rules. Each case took, on average, 1.29 ms to be processed. A fitness value of 0.813 was reached assuming the same weight for every conformance rule. However, it was decided that the process *end condition* - a process should only end when dismissed or when the fine is fully paid - had to weight more than the others given the severity of its possible violation. Therefore, considering a 5 times increase in this conformance rule weight, a new fitness value of 0.81 was reached. The insignificant change in fitness can be seen as a sign that this rule was not broken that many times. In fact, there were only 6986 (4.6%) deviations recorded for this particular conformance rule. Despite the low number of occurrences, it still added up to 329.069,54 EUR in financial injury. It is also relevant to note that the activity "Add Penalty" failed to increase the fine's amount 13 times, although not much, this deviations possible lead to more financial losses.

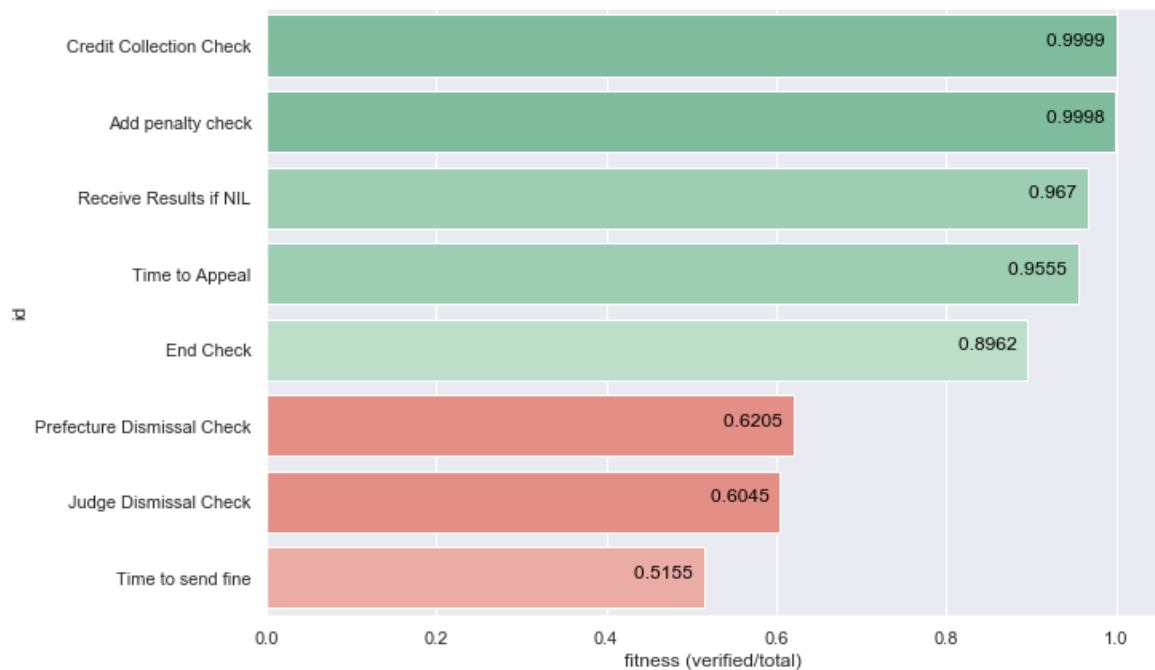


Figure 6.3: Fitness values for each conformance rule. The results are colour-coded, shades of green indicate satisfactory results while shades of red indicate less than ideal results (with a threshold of 0.7).

Shifting the perspective to the rule level, it was possible to compute a fitness value for each conformance rule. These can be seen in Figure 6.3. It is concluded that most of the recorded deviations are directly related to wrongly set dismissal values and delays in sending fine's

notifications. More precisely, 38% of times the fine was dismissed by the prefecture with a wrong dismissal value, 39% the fine was dismissed by the judge with a wrong dismissal value and 48% the fine was sent after the fixed limit to send fine's notifications.

The analysis that follows was driven by a set of questions, to which an answer was sought. They are:

What is the average number of deviations per case?

The average number of deviations per case, is relatively low, set at 0.39 *deviations per case*. More precisely, there are 54.812 cases with at least one deviation, corresponding to 36.5% of the total events. Analogously, the remaining cases have no data-flow deviations (73.5%). The maximum number of deviations recorded in one individual case was 3, however, only 10 cases reflected this behaviour. This conclusions were supported by Figure 6.4.

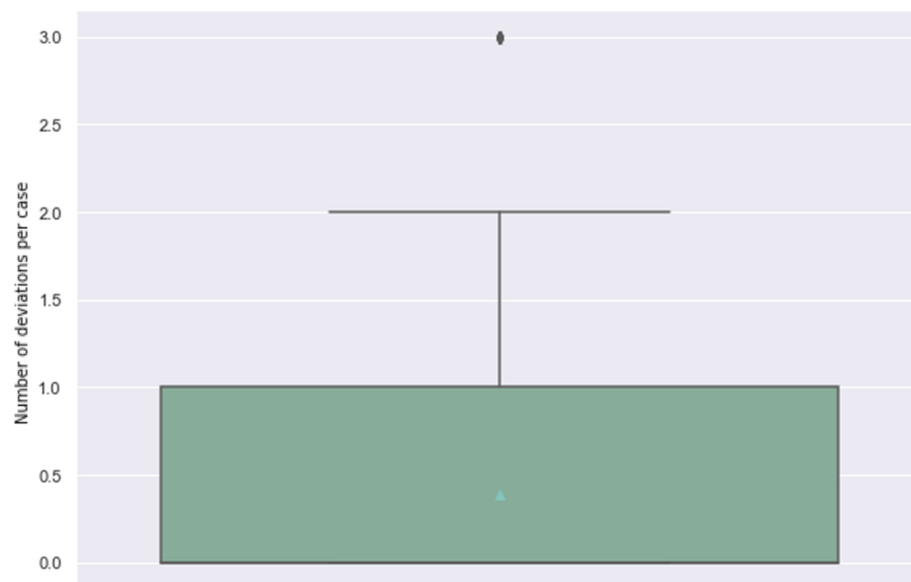


Figure 6.4: Boxplot of the number of deviations per case.

What is the percentage of deviations per conformance rule type?

From the detected conformance rule deviations, it was possible to identify that all the rules were at least broke once. Nevertheless, a clear amount of deviations were directly related to activity duration, in fact, 86% of deviations were of this type. 14% were of type "decision" and less than 1% were of type "effect". This shows, that activity effects are taking place in a correct manner whilst activity duration's being violated more often and should be targeted for a more detailed analysis. This conclusions were supported by Figure 6.5.

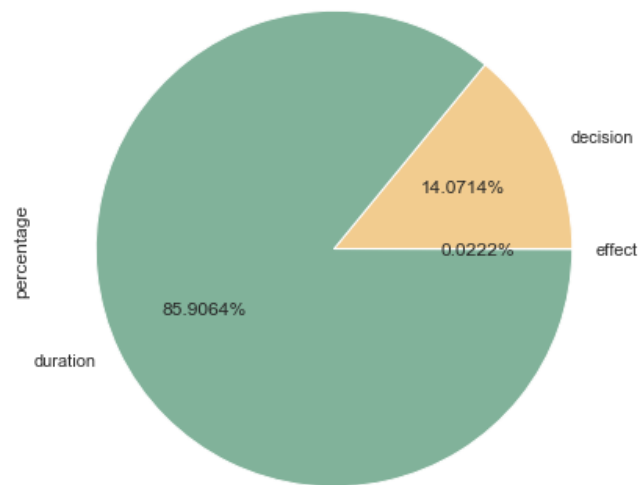


Figure 6.5: Pie chart highlighting deviations per rule type.

What is the average activity duration when the activity duration typed rules fail?

For this particular scenario, two rules were defined: *Time to Send fine (AD₁)* and *Time to Appeal (AD₂)*. After an initial analysis, it was found that *AD₁* amount to, approximately, 99% of the this deviation type. This points out irregular behaviour regarding the process of mailing the fine notification to the offender. The Italian law fixes the maximum time to send the fine's notification to 90 days. However, when this law is broken, the average amount of time until mailing the fine is of 123 days, one month above the defined limit.

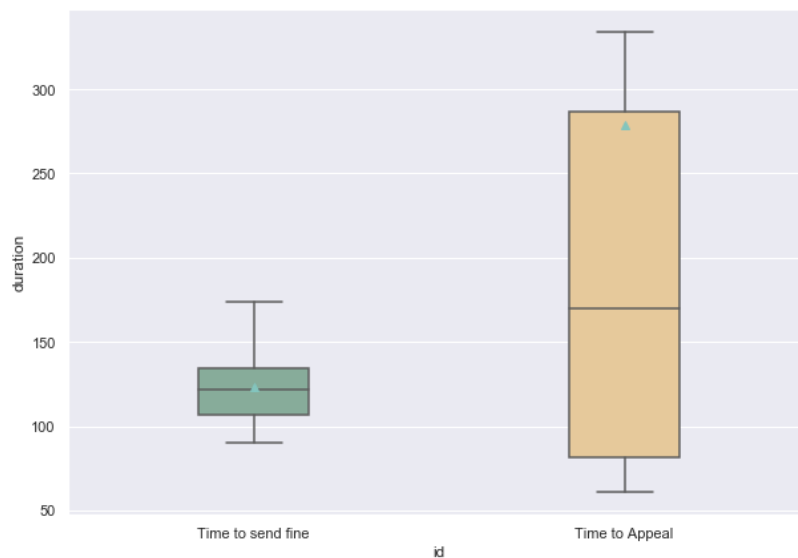


Figure 6.6: Boxplot of duration for each duration-typed conformance rule.

In respect to rule *AD2*, despite being broken considerably less times, the delays were also significantly larger. The maximum time to appeal to a judge was of 60 days. When this limit is not abide, it takes in average 279 days to appeal to a judge, almost 7 months above the defined limit. This conclusions were supported by Figure 6.6.

6.2 EVALUATION USING SYNTHETIC DATA

In this section, several synthetic event logs were generated to evaluate the behaviour of the proposed conformance checking and the event log reduction techniques.

6.2.1 *Experimental setup*

The conformance checking experiments were mainly focused in its time performance and scalability, specifically:

1. *How is performance affected by increasing the number of conformance rules in the model?*
2. *How is performance affected by increasing the number of cases in the event log?*
3. *How is performance affected by increasing the number of events per case in the event log?*

With this in mind, four BPMN-E² diagrams were created. All of them are composed of five activities (A, B, C, D, E) and five monitoring groups. The first diagram (referred to as AD) contains two extra activity duration elements, the second diagram (AE) contains two extra activity effect elements, the third diagram (DP) contains two extra decision point elements, finally, the forth diagram (ALL) contains all three elements from the previous diagrams. Figure 6.7 illustrates the last diagrams used mentioned (ALL), the others are exactly the same diagram but keeping only one type of extension elements.

The event logs, on the other hand, were designed to evaluate the scalability of the approach. It was assured that every event log was fully compliant regarding the control-flow perspective. In turn, each event log was generated with a combination of the following characteristics: number of cases (1k, 10k and 100k) and number of events per case (2 and 4). This results in 6 different event logs with increasing number of cases and events.

The following tests were executed in a machine with an 2,5 GHz Quad-Core Intel Core i7 CPU and 16 GB of RAM.

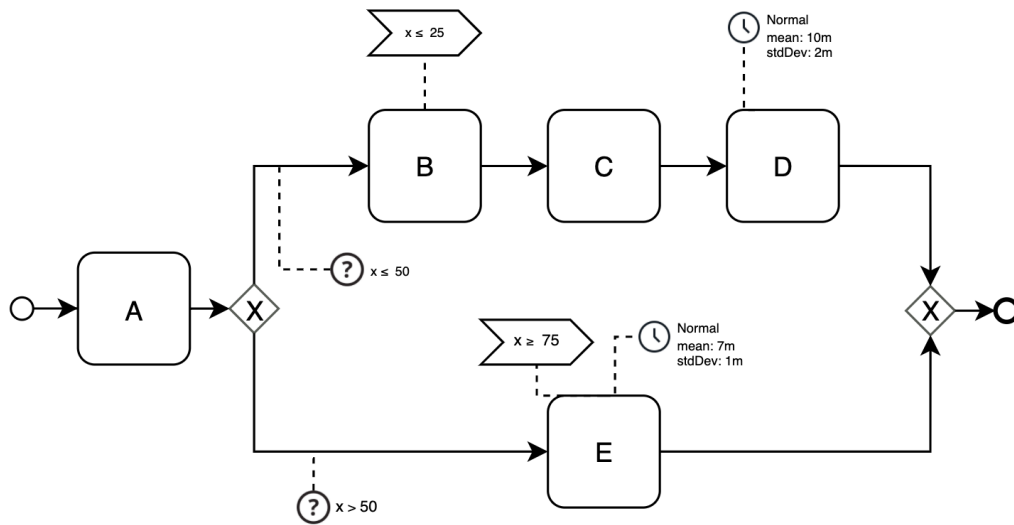


Figure 6.7: Diagram ALL used during evaluation. Note that for each path, despite the distinct number of activities the number of rules to test was kept.

6.2.2 Conformance Checking evaluation

To test the conformance checking approach, each event log was ran against each process model, to a total of 24 tests. Each test was executed five times, the test results were then averaged to obtain a final result, as shown in Table 6.3

| Model | Cases | Events per case | Execution time (s) |
|-------|-------|-----------------|--------------------|
| AD | 1k | 2 | 0.3142 |
| | | 4 | 0.3676 |
| | 10k | 2 | 2.9093 |
| | | 4 | 3.3608 |
| | 100k | 2 | 28.9304 |
| | | 4 | 33.3462 |
| AE | 1k | 2 | 0.3520 |
| | | 4 | 0.3868 |
| | 10k | 2 | 3.4969 |
| | | 4 | 3.6441 |
| | 100k | 2 | 35.0104 |
| | | 4 | 36.5146 |
| DP | 1k | 2 | 0.3372 |
| | | 4 | 0.3668 |
| | 10k | 2 | 3.2929 |
| | | 4 | 3.4433 |
| | 100k | 2 | 33.3938 |
| | | 4 | 33.6114 |
| ALL | 1k | 2 | 1.0125 |
| | | 4 | 1.0740 |
| | 10k | 2 | 9.5141 |
| | | 4 | 10.0135 |
| | 100k | 2 | 95.4211 |
| | | 4 | 99.3699 |

Table 6.3: Evaluation results.

Analysing the obtained results it is possible to assess the following in regards to the overall performance. By isolating the event log, it is noticeable that verifying the same amount of rules takes approximately the same time regardless of their type. This was expected since every rule is built over a same SMT Solver. Furthermore, the conformance checking task scales linearly regarding the number of rules being tested per case. This can be backed up by the obtained results, considering a linear increase in the task duration for bigger logs that comprise, naturally, more rules to be verified. The number of events per case, and consequently, the total number of events, proved to be irrelevant to the overall performance since the execution time is hardly affected by any change in their value if the number of rules per case is kept the same.



Figure 6.8: Evaluation heatmap. Execution time's correlations are highlighted.

Considering both the results on Table 6.3 and the *heatmap* on Figure 6.8, it is evident that the execution time is mainly affected by the number of rules that are to be tested. The proposed conformance checking task runs in $\mathcal{O}(N)$, with N being the number of rules to test. This number is indirectly impacted by the number of cases (\mathcal{C}) and the average number of rules per case (\mathcal{R}). This way, the previous time complexity can be re-written into $\mathcal{O}(\mathcal{C} * \mathcal{R})$.

The solution's performance will then decrease for larger event logs and more *rule-dense* BPMN-E² diagrams.

6.2.3 Event log reduction evaluation

The event log reduction experiments were conducted to assess how well did it perform. For this, each clustering technique will be used to perform increasingly heavier log reductions while verifying how it impacts both overall representability of the original event log and the conformance checking task. To evaluate the goodness of clustering results, *clustering validation* will be used, since it has long been recognized as one of the vital issues essential to the success of clustering applications [29]. There are two main ways to perform clustering validation: a) external clustering validation; and b) internal clustering validation. The former uses external criteria to validate the results, e.g. cluster labels, whilst the latter uses features inherent to the data itself [29, 52]. Commonly, it is hard to find real datasets that have prior cluster information [29], making internal validation the only way to validate such problems.

In this particular scenario, only external cluster validation techniques were used, by comparing fitness values obtained from the conformance checking task run over the original event log with fitness values computed using its downsized versions. This way, it can be assessed how different clustering algorithms and parameters impact the event log by cross-referencing their fitness values with the original unaltered one.

The technique was applied to the event log introduced in Section 6.1. Five different percentual downsizes (50%, 40%, 30%, 20%, 10%) were applied for both K-means and DBSCAN clustering algorithms. Each downsized log was then fed into the developed conformance checking algorithm. The error is computed simply by comparing the fitness value obtained using the reduced event log with the fitness value obtained from the original event log, using the absolute value of the difference between them. This way, it is possible to measure how the results “drift” when different reduction parameters are used.

Since this technique relies heavily on how well the data is clustered, relevant parameters were tweaked for each clustering algorithm. Different *number of clusters* and *eps* values were

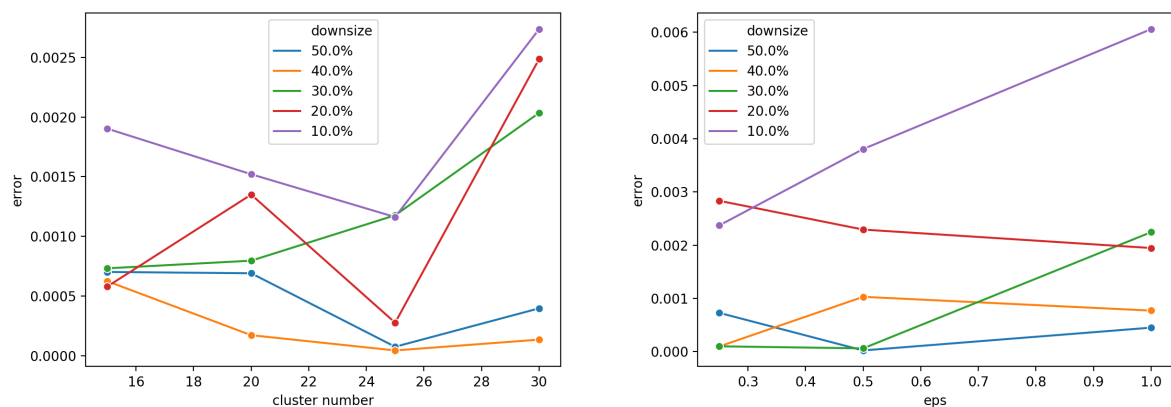


Figure 6.9: Conformance Checking errors after applying the event log reduction technique, using K-means (left) and DBSCAN (right) clustering algorithms.

Analysing the graphs in Figure 6.9, it is noticeable that the difference in fitness value for this particular scenario is minimal. In fact, the maximum error value does not exceed 0.0006, which is greatly satisfactory. In general, the error increases for heavier downsizes with K-means performing better overall regardless of the insignificant difference between the two clustering algorithms. Nevertheless, there are cases where heavier downsizes yield better results. In fact, K-means algorithm performs at its highest on a 40% reduction with 25 clusters. On the other hand, DBSCAN algorithm excels on both 30% and 50% reductions with an *eps* value of 0.5.

Considering that the conformance checking algorithms time complexity is $\mathcal{O}(\mathcal{C} * \mathcal{R})$, with \mathcal{C} being the total number of cases, i.e. the event log size, and \mathcal{R} the average number of rules per case, reducing the event log is bound to reduce the algorithm's execution time. In fact, it is expected that a given percentage reduction of the number of cases will result in the same percentage reduction of the execution time. For this particular case, it would be possible to reduce the original execution time of approximately 3 *minutes* to 18 *seconds* without any significant change in the final result, by applying a 90% event log reduction.

6.3 SUMMARY

In this chapter, the developed techniques were tested against real-life and synthetic process models and event logs. Using a real-life problem, it was possible to highlight the expressiveness and *ease of modeling* gained by using the BPMN-E2 notation. Moreover, a process analysis was conducted using the developed approach, exposing the potential of using the developed python library in conjunction with other data science libraries, such as *pandas* and *seaborn*. The event log reduction technique was also tested using the real-life scenario's event log. The results proved to be very satisfactory, however clustering techniques are highly affected by the context in which they are used and different logs could yield different results. Using synthetic event logs, the overall performance was measured. The data-aware conformance checking algorithm runs as expected, being directly influenced by the number of conformance rules to check and the overall event log size.

CONCLUSIONS AND FUTURE WORK

In this dissertation a new conformance checking approach is proposed and developed. It aims to take advantage of the new elements introduced with the new process model notation - BPMN-E² - an extension of the well known BPMN notation.

The extension elements of BPMN-E² provide an added layer of expressiveness to the BPMN notation that were previously only present in natural language documents that complemented the process models. The approach relies on SMT Solvers to define a set of conformance rules that must be followed during process execution. Therefore, there is one initial phase that constitutes the conversion from BPMN-E² to a valid conformance rule that is later tested during the conformance checking phase.

An event log reduction algorithm was also proposed and developed following the trends of. These use state-of-the-art clustering algorithms to identify certain behaviour patterns on an event log and, mainly, to discover groups of less cluttered and easier to read process models. In this dissertation, however, clustering algorithms are used with the purpose of identifying homogeneous process behaviour between cases in order to downsize the event log by sampling the cases that better represent the process execution and removing those that prove to be redundant.

Both of the discussed approaches were developed using Python, due to the vast amount of data mining related libraries and, especially, to the Process Mining library PM4Py. The solution was then used to analyse real event data related to a *Fine Management System* highlighting the benefits of the approach. Synthetic event logs were also used to evaluate the performance of the conformance checking and the event log reduction algorithms.

There are still several ways to improve the currently developed features, such as: a) improving the current visualization module to enable a better graphical representation and an interactive manipulation a BPMN-E² model; b) enabling the creation of custom conformance rules that don't fall under the built-in ones; c) developing a better API to create independent DFRM's that can be used in conjunction with any other type of model notation; and d) allowing online conformance checking by extending support for streaming event-data. Furthermore, this dissertation's work can be seen as the gateway to start using BPMN-E² in the context of Process Mining, and, therefore it is important to mention that it

focused only on one of the three types of Process Mining - conformance checking. This said, there is still interesting and useful work to be done in respect to this new BPMN-E² notation and its usage in Process Mining techniques. Possible future work routes could include the development of a *process discovery* technique to build BPMN-E² models starting from an attribute rich event log; the development of a *process enhancement* technique to add the BPMN-E² extension elements to an already existing BPMN model; or even, the adaptation of the current *conformance checking* technique to analyse streams of events in real time.

BIBLIOGRAPHY

- [1] bpmn-python. <https://github.com/KrzyHonk/bpmn-python>.
- [2] Celonis. <https://www.celonis.com/>.
- [3] Disco. <https://fluxicon.com/disco/>.
- [4] Graphviz. <https://graphviz.org/>.
- [5] Networkx. <https://networkx.org/>.
- [6] Pm4py. <https://pm4py.fit.fraunhofer.de/>.
- [7] Prom. <https://promtools.org/>.
- [8] Scikit-learn. <https://scikit-learn.org/>.
- [9] A Adriansyah, B. F. Van Dongen, and W. M.P. Van Der Aalst. Towards robust conformance checking. In *Lecture Notes in Business Information Processing*, volume 66 LNBIP, pages 122–133, 2011.
- [10] Pavlo D. Antonenko, Serkan Toy, and Dale S. Niederhauser. Using cluster analysis for data mining in educational technology research. *Educational Technology Research and Development*, 60(3):383–398, 2012.
- [11] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard Version 2.6. Technical report, University of Iowa, 2017.
- [12] Alessandro Berti, Sebastiaan J. van Zelst, and Wil van der Aalst. Process Mining for Python (PM4Py): Bridging the Gap Between Process- and Data Science. *CEUR Workshop Proceedings*, 2374:13–16, may 2019.
- [13] Armin Biere, Marijn Heule, Hans Van Maaren, Toby Walsch, Clark Barrett, Roberto Sebastiani, Sanjit A Seshia, and Cesare Tinelli. Handbook of Satisfiability Satisfiability Modulo Theories. Technical report, OS Press, 2008.
- [14] Yukun Cao. *Attribute-Driven Hierarchical Clustering of Event Data in Process Mining Master Thesis*. PhD thesis, RWTH Aachen University, 2018.
- [15] Franck Cassez and Anthony M Sloane. ScalaSMT: Satisfiability Modulo Theory in Scala (tool paper). In *SCALA 2017 - Proceedings of the 8th ACM SIGPLAN International Symposium on Scala, co-located with SPLASH 2017*, pages 51–55, 2017.

- [16] Massimiliano de Leoni, Jorge Munoz-Gama, Josep Carmona, and Wil M.P. van der Aalst. Decomposing alignment-based conformance checking of data-aware process models. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8841, pages 3–20, 2014.
- [17] Massimiliano De Leoni and Wil M.P. Van Der Aalst. Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8094 LNCS:113–129, 2013.
- [18] Massimiliano De Leoni and Wil M.P. Van Der Aalst. Data-aware process mining: Discovering decisions in processes using alignments. *Proceedings of the ACM Symposium on Applied Computing*, pages 1454–1461, 2013.
- [19] Massimiliano De Leoni, Wil M.P. Van Der Aalst, and Boudewijn F. Van Dongen. Data- and resource-aware conformance checking of business processes. *Lecture Notes in Business Information Processing*, 117 LNBIP:48–59, 2012.
- [20] Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *Lecture Notes in Computer Science*, pages 337–340. Springer-Verlag, 2008.
- [21] Guojun Gan and Michael Kwok Po Ng. K-Means Clustering With Outlier Removal. *Pattern Recognition Letters*, 90:8–14, 2017.
- [22] Fritz H. Grupe and M. Mehdi Owrang. Data base mining: Discovering new knowledge and competitive advantage. *Information Systems Management*, 12(4):26–31, 1995.
- [23] David B. Henry, Patrick H. Tolan, and Deborah Gorman-Smith. Cluster analysis in family psychology research. *Journal of Family Psychology*, 19(1):121–132, 2005.
- [24] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.
- [25] Anna A. Kalenkova, Massimiliano De Leoni, and Wil M.P. Van Der Aalst. Discovering, analyzing and enhancing BPMN models using ProM. *CEUR Workshop Proceedings*, 1295:36–40, 2014.
- [26] Anna A. Kalenkova, Wil M.P. van der Aalst, Irina A. Lomazova, and Vladimir A. Rubin. Process mining using BPMN: relating event logs and process models. *Software and Systems Modeling*, 16(4):1019–1048, 2017.
- [27] Milan Kubina, Michal Varmus, and Irena Kubinova. Use of Big Data for Competitive Advantage of Company. *Procedia Economics and Finance*, 26:561–565, 2015.

- [28] Sander J.J. Leemans, Erik Poppe, and Moe T. Wynn. Directly follows-based process mining: Exploration & a case study. *Proceedings - 2019 International Conference on Process Mining, ICPM 2019*, pages 25–32, 2019.
- [29] Yanchi Liu, Zhongmou Li, Hui Xiong, Xuedong Gao, and Junjie Wu. Understanding of internal clustering validation measures. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 911–916, 2010.
- [30] Linh Thao Ly, Stefanie Rinderle-Ma, and Peter Dadam. Design and verification of instantiable compliance rule graphs in process-aware information systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6051 LNCS:9–23, 2010.
- [31] Linh Thao Ly, Stefanie Rinderle-Ma, David Knuplesch, and Peter Dadam. Monitoring business process compliance using compliance rule graphs. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7044 LNCS(PART 1):82–99, 2011.
- [32] J MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–296, 1967.
- [33] Felix Mannhardt, Massimiliano de Leoni, Hajo A. Reijers, and Wil M.P. van der Aalst. Balanced multi-perspective checking of process conformance. *Computing*, 98(4):407–437, apr 2016.
- [34] Felix Mannhardt, Massimiliano de Leoni, Hajo A. Reijers, and Wil M.P. van der Aalst. Balanced multi-perspective checking of process conformance. *Computing*, 98(4):407–437, apr 2016.
- [35] R. S. Mans, M. H. Schonenberg, M. Song, W. M.P. Van Der Aalst, and P. J.M. Bakker. Application of process mining in healthcare - A case study in a Dutch Hospital. *Communications in Computer and Information Science*, 25 CCIS:425–438, 2008.
- [36] Ronny S Mans, Helen Schonenberg, Giorgio Leonardi, and Silvia Panzarasa. Process mining techniques: An application to stroke care Process Mining in Logistics View project Operational support for business processes View project. *Studies in health technology and informatics*, 136:573–8, 2008.
- [37] Jorge Munoz-Gama. *Conformance checking and diagnosis in process mining : comparing observed and modeled processes*. PhD thesis, Universitat Politècnica de Catalunya – BarcelonaTech, 2014.

- [38] Object Management Group (OMG). Business Process Model and Notation (BPMN) Version 2.0.2. Technical Report December, Object Management Group, 2013.
- [39] Mahamed G.H. Omran, Andries P. Engelbrecht, and Ayed Salman. An overview of clustering methods. *Intelligent Data Analysis*, 11(6):583–605, 2007.
- [40] Mateo Ramos-Merino, Juan M. Santos-Gago, Luis M. Álvarez-Sabucedo, Victor M. Alonso-Roris, and Javier Sanz-Valero. BPMN-E2: a BPMN extension for an enhanced workflow description. *Software and Systems Modeling*, 18(4):2399–2419, aug 2019.
- [41] A Rozinat, Ivo S.M. de Jong, C W Günther, and W. M.P. van der Aalst. Process mining applied to the test process of wafer scanners in ASML. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 39(4):474–479, 2009.
- [42] Roberto Sebastiani. Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation*, 3(3-4):141–224, 2007.
- [43] M. Song, H. Yang, S. H. Siadat, and M. Pechenizkiy. A comparative study of dimensionality reduction techniques to enhance trace clustering performances. *Expert Systems with Applications*, 40(9):3722–3737, 2013.
- [44] Minseok Song, Christian W. Günther, and Wil M.P. Van Der Aalst. Trace clustering in process mining. *Lecture Notes in Business Information Processing*, 17 LNBIP:109–120, 2009.
- [45] N Suthar, I Jeet Rajput, and V Kumar Gupta. A Technical Survey on DBSCAN Clustering Algorithm. *Ijser.Org*, 4(5):1775–1781, 2013.
- [46] Karuna Kant Tiwari. DBSCAN : An Assessment of Density Based Clustering and It ' s Approaches. *International Journal of Scientific Research & Engineering Trends*, 2(5):109–113, 2016.
- [47] W. M.P. van der Aalst, H. A. Reijers, A. J.M.M. Weijters, B. F. van Dongen, A. K. Alves de Medeiros, M. Song, and H. M.W. Verbeek. Business process mining: An industrial application. *Information Systems*, 32(5):713–732, 2007.
- [48] Wil Van Der Aalst, Arya Adriansyah, Ana Karla Alves De Medeiros, et al. Process mining manifesto. In *Lecture Notes in Business Information Processing*, volume 99 LNBIP, pages 169–194, 2012.
- [49] Wil Van der Aalst, Arya Adriansyah, and Boudewijn Van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.
- [50] Wil van der Aalst and Wil van der Aalst. Data Science in Action. In *Process Mining*, pages 3–23. Springer Berlin Heidelberg, 2016.

- [51] Stephen White. Introduction to BPMN. *IBM Cooperation*, 2:1–11, 2004.
- [52] D. S. Wilks. Cluster Analysis. *International Geophysics*, 100:603–616, 2011.

