Combining event logs with supplementary events

Stevandy, C.

Award date: 2015

Link to publication
Combining Event Logs with Supplementary Events

Master Thesis

Christian Stevandy

Committees:
Prof.dr.ir. H.A. Reijers
Felix Mannhardt MSc
dr.ir. H. Rik Eshuis

Final version

Eindhoven, 31 August 2015
Abstract

Process mining techniques aim to extract valuable knowledge of business processes based on event logs. Current process mining techniques work for a single complete event log. However, events related to an execution of a business process are often stored in multiple databases of the information systems without any information on how they are related to each other. As a consequence, information related to process execution is scattered in multiple event logs. Previous study presents a method to extend an incomplete event log with events from supplementary sources. However, the method is unable to correlate an event that correspond to an optional activity.

In this master thesis, we present two approaches to combine incomplete event log with supplementary events. Our approaches have been evaluated through two case studies. The evaluation outcome shows that our approaches are capable of extending event logs with supplementary events that correspond to an optional activity.

Keywords: process mining, event log, optional activity, DPN-net, multi-perspective conformance checking, event correlation.
Contents

Contents

vi

List of Figures

vii

List of Tables

ix

1 Introduction

1.1 Motivation ................................................. 1
1.2 Problem Statement .......................................... 2
1.3 Research Question and Goal ................................. 2
1.4 Research Scope .............................................. 2
1.5 Research Method and Outline .............................. 3

2 Preliminaries

2.1 Process-Aware Information Systems ....................... 5
2.1.1 Process Model ........................................... 5
2.1.2 Event Log .............................................. 8
2.2 Process Mining ............................................. 9
2.2.1 Conformance Checking ................................. 10

3 Analysis

3.1 Correlating Optional Activities ........................... 13
3.2 Root Cause Analysis ...................................... 15

4 Design

4.1 Approach 1: Heuristics Based Logs Extension .......... 17
4.1.1 Algorithm ............................................. 18
4.1.2 Function ............................................. 21
4.1.3 Computational Complexity ........................... 24
4.2 Approach 2: Alignment Based Logs Extension .......... 24
4.2.1 Algorithm ............................................. 24
4.2.2 Function ............................................. 29
4.2.3 Computational Complexity ........................... 33

5 Implementation

5.1 Overview .................................................. 36
5.2 User Guide ................................................. 36
5.3 Additonal Plugins ......................................... 37

6 Evaluation

6.1 Case 1: Road Data Fine Management .................... 38
6.1.1 Experiment Data ....................................... 38
6.1.2 Evaluation Method .................................... 40
## CONTENTS

6.1.3 Evaluation: Approach 1 ......................................................... 41
6.1.4 Evaluation: Approach 2 ......................................................... 42
6.2 Case 2: WW-uitkering UWV ....................................................... 46
   6.2.1 Experiment Data .............................................................. 46
   6.2.2 Evaluation Method ......................................................... 48
   6.2.3 Evaluation: Approach 1 ..................................................... 48
   6.2.4 Evaluation: Approach 2 ..................................................... 48
6.3 Summary ................................................................. 51

7 Conclusions ................................................................. 52
  7.1 Conclusions ................................................................. 52
  7.2 Reflection ................................................................. 52
  7.3 Limitations and Future Works ............................................ 53

Bibliography ................................................................. 55
# List of Figures

1.1 Research Scope .................................................. 3
2.1 DPN-net of the road traffic fine management process [1] ............ 6
2.2 A fragment of event logs [2] ...................................... 8
2.3 The three basic types of process mining are explained in terms of input and output: (a) discovery, (b) conformance checking, and (c) enhancement. [3] ............ 9
2.4 Trace Variants ...................................................... 11
2.5 Process model with an optional activity Exercise .................. 11
3.1 Process model with an optional activity Exercise .................. 14
3.2 Process model with a repeatable optional activity Exercise ........ 14
3.3 Overview of the current approach[4] ................................ 15
3.4 Root cause analysis of current problem. ........................ 16
3.5 Overview of the second approach. ............................... 16
4.1 DPN-net example .................................................. 19
4.2 Main Event Log .................................................... 20
4.3 Supplementary Events ............................................ 20
4.4 Possible Extended Traces ........................................ 20
4.5 Filter Supplementary Events by Time ............................ 21
4.6 Extended Event Log $L_E$ ......................................... 21
4.7 DPN-net example 2 ................................................ 26
4.8 Main Event Log .................................................... 26
4.9 Supplementary Events ............................................ 26
4.10 Process trace example 2 .......................................... 27
4.11 Possible Extended Traces ........................................ 27
4.12 Extended Trace Exercise 25 Feb 15 ............................. 28
4.13 Extended Log ...................................................... 28
5.1 ProM Overview .................................................... 35
5.2 Implemented Plug-ins ............................................ 36
6.1 Major Road Process Fine Management Trace Variants ................ 39
6.2 DPN-net of the road traffic fine management process [1] .......... 40
6.3 extended trace has the same cost as before ........................ 44
6.4 Multi Payments Extension ....................................... 45
6.5 Main Event Log .................................................... 46
6.6 Supplementary Events ............................................ 46
6.7 DPN-net of Unemployment Benefits Claim Process ................. 47
6.8 Alignment Before Extension ..................................... 49
6.9 Optional Subprocess ............................................. 50
6.10 Extended Log Sequential Case ................................. 50
LIST OF FIGURES

6.11 Extended Log Parallel Case .................................................. 51
## List of Tables

2.1 Guards of the road traffic fine management DPN-net [1] ................................. 6  
2.2 Control-flow perspective alignment of a log trace the road fines management DPN-net[1] ................................................................. 10  
3.1 Alignment Results ................................................................................. 15  
6.1 Guards of the road traffic fine management DPN-net [1] ................................. 39  
6.2 Send Fine Extension Results (perfectly conformed traces) ................................. 41  
6.3 Send Fine Extension Results (all traces) .......................................................... 41  
6.4 Send for Credit Collection extension results(perfectly conformed traces) .......... 42  
6.5 Send for Credit Collection extension results(all traces) ........................................ 42  
6.6 Payment extension results (perfectly conformed traces) ........................................ 42  
6.7 Payment extension results (all traces) ............................................................ 42  
6.8 Send Fine Extension Results (Approach 2) ....................................................... 43  
6.9 Send Fine Extension Results (Approach 1) ....................................................... 43  
6.10 Send for Credit Collection Extension Results (Approach 2) ......................... 43  
6.11 Send for Credit Collection Extension Results (Approach 1) ......................... 43  
6.12 Single Payment Extension Results (Approach 2) ............................................ 44  
6.13 Single Payment Extension Results (Approach 1) ............................................ 44  
6.14 Multiple Payment Extension Results ............................................................ 45  
6.15 Guards of the unemployment benefits claim process DPN-net ....................... 48
Chapter 1

Introduction

This master thesis is the result of the graduation project for the Business Information Systems master at Eindhoven University of Technology (TU/e). The project is carried out within the Architecture of Information Systems (AIS) group of the Mathematics and Computer Science department of TU/e. The AIS research group investigates methods, techniques and tools for the design and analysis of process-aware information systems. The AIS group makes research results accessible by providing (open-source) software. Notable examples are ProM (process mining and process analysis) and YAWL (workflow management).

Process mining techniques currently work for single complete events. However events related to an execution of a business process are often stored in multiple databases of the information systems without any information on how they are related to each other. This thesis analyses the problem of combining event logs with supplementary events by using process model as domain knowledge. This resulted two approaches to combine an event log with supplementary event as plug-ins within ProM framework.

This chapter introduces the context of this thesis. Section 1.1 introduces the motivation behind this thesis. Section 1.2 explains the problem description. Section 1.3 describes research question and goal. Section 1.4 discusses the research scope in this thesis. Finally, the research method and outline are described in Section 1.5.

1.1 Motivation

Nowadays organisations all over the world generate more and more data. In 2012, more than 200 million emails were sent every minute [5]. This phenomenon is generally referred to as big data. Does this phenomenon give any benefit to organisations? Today, it is possible to translate big data into useful knowledge. This knowledge is valuable for improving decision making and performance in organisations [6]. With the right experts and the right tools, big data may generate powerful knowledge that can give insights for organisations [6]. One way to generate knowledge from big data is by using process mining.

Process mining techniques allow for extracting knowledge from event logs [2]. An event log is a file that stores information about the execution of business processes in organisations. An event log contains traces that record sequences of events and each trace represents a record of an executed process instance.

Thanks to the widespread use of distributed and parallel computing systems, organisations often use multiple IT systems to support their business operations. Information about the execution of a business process in many cases is scattered across multiple event logs. Often, those event logs
are missing important information on how they are related to each other. Hence, it is difficult to apply existing process mining techniques, as these techniques are not applicable for multiple event logs because they require single event log [2]. Both works [4] and [7] propose techniques that create a single event log out of multiple unlinked sources.

In [7], an artificial immune system algorithm is used to merge separate log file into one log file. Whereas this work addresses the issue of merging multiple event logs, it does not use any domain knowledge from domain experts. In order to extract useful knowledge that valuable for organisations, the right domain experts are required [8]. Therefore, a technique that utilizes knowledge from these experts is more desirable. The approach presented in [4] extends a main event log with events from supplementary data sources by using alignment from a process model. However, this approach is incapable of correlating any of the events related to an optional activity.

In a business process, the process can end without an execution of an optional activity. As a consequent, it is difficult to link an optional activity with alignment because it may not present in the main process execution. Accordingly, this master thesis will provide an approach to support domain experts correlating unlinked optional supplementary events that correspond to a main event log.

1.2 Problem Statement

As discussed in the previous section, information about the execution of a business process is scattered across multiple event logs and current existing process mining techniques work on a single event log. Currently, there is no approach that capable of combining an event log with supplementary events that capable of correlating an event that correspond to an optional activity.

1.3 Research Question and Goal

The previous section describes the problem statement in this thesis. Consequently, we formulate the following research goal and general research question of this project:

**Research Goal**: Develop an approach to combine an event log with supplementary events that capable of correlating an event that correspond to an optional activity.

**Research Question**: How do we combine an event log with supplementary events?

**Sub Questions**:
1. Why is the approach in [4] unable to correlate supplementary events that correspond to an optional activity?
2. How do we correlate an optional activity?

1.4 Research Scope

The goal of this thesis is to develop an approach to combine an event log with supplementary events that capable of correlating an event that correspond to an optional activity. The research scope is illustrated in Figure 1.1

The research will not propose a new process mining technique, a new event log format, or a new process model language. This thesis will focus on finding an approach to combine an event log with unlinked supplementary events.
1.5 Research Method and Outline

In order to solve the research problem and reach the research goal, the following steps based on the regulative cycle [9] are used in this thesis:

1. **Literature study**
   Chapter 2 provides background information related to this research.

2. **Problem statement**
   Chapter 3 explains the problem statement in more detail.

3. **Analysis**
   Chapter 3 discusses the root causes of the problem.

4. **Design**
   Chapter 4 presents the approaches used to solve the problem.

5. **Implementation**
   Chapter 5 explains how to use prototypes of the implemented approaches in chapter 4 within ProM.

6. **Evaluation**
   Chapter 6 describes an evaluation of the implemented approach
   Chapter 7 concludes the research
Chapter 2

Preliminaries

This chapter introduces basic knowledge that is relevant to this thesis. Section 2.1 introduces relevant basic concepts in process-aware information systems: process model and event logs. Section 2.2 introduces the concept of process mining in general.

2.1 Process-Aware Information Systems

Back in 1970s and 1980s, information systems were dominated by data-driven approaches. The main concern of IT was on recording and retrieving information that represented as data. Consequently, this led to the development of data modeling and data-centric information systems, but the development of business process modeling was unheard at that time. It was only after the business process reengineering (BPR) concept arose in the early 1990s [10], interest in business process was increased. As a result, information systems were driven to put more emphasize on a process. A PAIS or Process-Aware Information System is defined as a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process model [11]. WorkFlow Management (WFM) Systems and Business Process Management (BPM) systems are example of PAISs.

The ultimate goal of a PAIS is to enable organizational changes by simply modifying the process models involved. PAIS enables not only process automation but also analysing and controlling executed processes. Every event or activity that happens in an organisation is recorded in PAISs as event logs. The analysis of the executed processes is analysed through process mining technique from event logs and process models.

This section introduced the components of PAIS. Section 2.1.1 introduces the concepts of process modeling in general and in particular Petri net with data (DPN-net), a process modeling language used in this thesis. Section 2.1.2 introduces the concepts of event logs in general.

2.1.1 Process Model

A picture is worth a thousand words, so does a process model. Process model is a graphical representation of a business process in an organisation. A process model has three important roles in the context of PAIS [12]:

**Insight.** A process model may give stakeholders insight into the a business process. It can be used to discuss requirements, design decisions, and to validate assumptions.
Analysis. A process model can be used to analyse business processes or the information system itself. For example, the performance of a system (e.g., response times), the performance of processes (e.g., simulation, queueing theory), and verification (e.g., soundness).

Enactment. A process model can be used as a basis to generate information systems.

There are several existing process modeling languages such as Petri-Net, BPMN, UML, BPEL, YAWL, etc. In this thesis, DPN-net is used as process modeling language because it captures control-flow and data behaviour that suitable for this thesis. A DPN-net is a process modeling language that extends a Petri net [13] with data-dependent guards and write operations on a set of given variables [14].

![DPN-net of the road traffic fine management process](image)

**Figure 2.1: DPN-net of the road traffic fine management process [1]**

<table>
<thead>
<tr>
<th>Transition(s)</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send for Credit Collection</td>
<td>ToPay' = Amount + Expenses - Payment ∧ ToPay' &gt; 0</td>
</tr>
<tr>
<td>Inv2, Inv3</td>
<td>Payment = Amount + Expenses</td>
</tr>
<tr>
<td>Inv1</td>
<td>(Dismissal != NIL) ∨ (Payment = Amount ∧ Points = 0)</td>
</tr>
<tr>
<td>Receive Result, Inv5</td>
<td>Dismissal = NIL</td>
</tr>
<tr>
<td>Inv4</td>
<td>Dismissal = #</td>
</tr>
<tr>
<td>Inv6</td>
<td>Dismissal = G</td>
</tr>
</tbody>
</table>

**Table 2.1: Guards of the road traffic fine management DPN-net [1]**

Figure 2.1 shows an example of a DPN-net and Table 2.1 shows an example of guards in DPN-net. A transition is described with a rectangular shape. The black coloured transitions e.g. Inv1, Inv2, etc are commonly referred to invisible transitions. The purpose of an invisible transitions is mainly only for routing and therefore they are not recorded in event logs. A variables is described as an oval such as Amount, Points, etc. Finally, a place is represented as a circle such as Start and End. The definition of DPN-net is adopted from [1] and [4] as follows:

**Definition 1 (DPN-net).** A DPN-net $N = (P,T,F,V,Val,W,G)$ is defined as:

- a set of places $P$;
– a set of transitions $T$;
– a flow relation $F \subseteq (P \times T) \cup (T \times P)$;
– a set $V$ of variables;
– a set $U$ of variable values;
– a function $Val: V \rightarrow 2^U$ that defines the values admissible for each variable $v \in V$, i.e. $Val(v)$ is the domain of variable $v$;
– a write function $W: T \rightarrow 2^V$ that labels each transition with a set of write operations, i.e. with the set of variables whose values needs to be written/updated;
– a guard function $G: T \rightarrow Formulas(V \cup \{v' \mid v \in V\})$ that associates each transition with a different guard.

A Petri net with data (DPN-net) is a directed bipartite graph containing places and transitions in which transitions can write variables. Transitions can only fire if all their input places have all necessary tokens and their guard is satisfied. A guard uses a formula with relational operators ($<$, $>$, $=$) and logical operators ($\lor$, $\land$, $\neg$). $Formulas(X)$ refers to guard formulas defined over a set $X$ of variables.

The preset of a transition $t$ is the set of its input places: $\bullet t = \{p \in P \mid (p,t) \in F\}$. The postset of $t$ is the set of its output places: $\star t = \{p \in P \mid (p,t) \in F\}$. When a variable $v \in V$ appears in a guard $G(t)$, it refers to the value just before the occurrence of $t$. If $v \in W(t)$, it can also appear as $v'$ and refer to the value after the occurrence of $t$.

Given a set $X$, $\mathbb{B}(X)$ denotes the set of all multi-sets over a set of $X$. In addition, given a multi-set $M \in \mathbb{B}(X)$, for each $x \in X M(x) \in \mathbb{N}$ is used to indicate the number of duplicates of element $x$ present in $M$.

**Definition 2 (State of a DPN-net).** Let $N = (P,T,F,V,U,Val,W,G)$ be a DPN-net. The set of possible states of $N$ is formed by all pairs $(M,A)$ where $M \in \mathbb{B}(P)$, i.e. a multi-set of the places in $P$. For any state:

- $M$: marking of Petri net $(P,T,F)$. The marking assigns to each place $p \in P$ a number of tokens.
- $A$ is function that associates a value with each variable, i.e. $A:V \rightarrow U \cup \{\bot\}$, with $A(v) \in Val(v) \cup \{\bot\}$.
- $\bot$: special value assigned to variables that have not been initialized.
- Two special markings $M_I,M_F$ the initial and final marking. The initial state of a DPN-net is $(M_I,A_I)$ with $A_I(v) = \bot$ undefined for each $v \in V$. A non-empty set of final states exists and includes every state $(M,A)$ with $M = M_F$.
- Zero or more transitions of a DPN-net may be able to fire.

**Definition 3 (Transition Firings).** Let $N = (P,T,F,V,U,Val,W,G)$ be a DPN-net. A transition firing $s$ is a pair $(t,w) \in T \times (V \rightarrow U)$, where $\rightarrow$ denotes partial functions, i.e. the function’s domain is a subset of $V$. $S_N$ denotes the set of possible transition firings. Additional functions to easily access the components of a transition firing $s = (t,w)$ are introduced as follows:

- $\#_{\text{act}}(s) = t$.
- $\#_{\text{vars}}(s) = w$,
- $\#_{\text{vars}}(s,v) = w(v)$ if $v \in \text{dom}(\#_{\text{vars}}(s))$,
- $\#_{\text{vars}}(s,v) = \bot$ if $v \not\in \text{dom}(\#_{\text{vars}}(s))$. Where function $\text{dom}(w)$ is a domain of $w$. 

Combining Event Logs with Supplementary Events
The transition *Payment* in Figure 2.1 is an example of an optional activity. An optional activity is a transition that generates a self-loop or it is in a choice which involved an invisible transition as one of its choice.

**Definition 4 (Self-loop).** Let $N = (P, T, F, V, U, Val, W, G)$ be a DPN-net and $p \in P, t \in T$. A self-loop is a couple $(p, t)$ such that $t$ is both an input and an output of $p$.

### 2.1.2 Event Log

In general, an event log is a set of traces or process instance that contains a sequence of events which are recorded in PAISs. Figure 2.2 shows an example of an event log. In Figure 2.2, the column Case id refers to a single trace or process id (e.g., 1, 2). The column Activity refers to an events such as *Register request*, *Examine thoroughly*, and *Check ticket*. Each case contains sequences of events or ordered events, for example in the first case (Case id = 1), *Register request* occurs before *Examine thoroughly*. Each event contains additional information such as timestamps, resource, and cost. In the context of this thesis, these properties are referred to as attributes. The notion of attributes is similar to the notion of variables that is described in Section 2.1.1.

<table>
<thead>
<tr>
<th>Case id</th>
<th>Event id</th>
<th>Properties</th>
<th>Timestamp</th>
<th>Activity</th>
<th>Resource</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35654423</td>
<td>Register request</td>
<td>30-12-2010:11.02</td>
<td>Pete</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td></td>
<td>35654424</td>
<td>Examine thoroughly</td>
<td>31-12-2010:10.06</td>
<td>Sue</td>
<td>400</td>
<td></td>
</tr>
<tr>
<td></td>
<td>35654425</td>
<td>Check ticket</td>
<td>05-01-2011:15.12</td>
<td>Mike</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>35654426</td>
<td>Decide</td>
<td>06-01-2011:11.18</td>
<td>Sara</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td></td>
<td>35654427</td>
<td>Reject request</td>
<td>07-01-2011:14.24</td>
<td>Pete</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>35654483</td>
<td>Register request</td>
<td>30-12-2010:11.32</td>
<td>Mike</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td></td>
<td>35654485</td>
<td>Check ticket</td>
<td>30-12-2010:12.12</td>
<td>Mike</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>35654487</td>
<td>Examine casually</td>
<td>30-12-2010:14.16</td>
<td>Pete</td>
<td>400</td>
<td></td>
</tr>
<tr>
<td></td>
<td>35654488</td>
<td>Decide</td>
<td>05-01-2011:11.22</td>
<td>Sara</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td></td>
<td>35654489</td>
<td>Pay compensation</td>
<td>08-01-2011:12.05</td>
<td>Ellen</td>
<td>200</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.2: A fragment of event logs [2]

In short, an event log can be described in the following summary:

- An event log is a set of traces or process instance.
- A trace or process instance is a sequence of events.
- An event can have many attributes.

Several formats exist for event logs such as CSV, XML, MXML, XES. Currently, the XES format is the standard event log format used in ProM. Therefore, the format used in this thesis is XES. The explanation of XES is beyond the context of this thesis. Please refer to [http://www.xes-standard.org/](http://www.xes-standard.org/) to get more information.

Adopted from [1] and [4], an event and event logs are formalized as follow:

**Definition 5 (Event & Event Logs).** Let $N = (P, T, F, V, U, Val, W, G)$ be a DPN-net and let $S_N$ the set of possible transition firings of $N$. An event $s_L \in S_N$ is the recording of transition firing.
A log trace $\sigma_L \in S_N^*$ is a sequence of events. An event log $L$ over $S_N$ is a multi-set of traces: $L \in \mathbb{B}(S_N)$.

Each event contains data for example timestamps that can be accessed by additional function $\#_{\text{time}}(s_L) = \#_{\text{vars}}(s_L, \text{time})$.

## 2.2 Process Mining

Process mining is state of art techniques to extract knowledge about a business process based on an event log and a process model. Process mining aims to discover, analyse, monitor and improve process. It provides techniques and tools to discover models that describe processes, organisations, and products [15]. It can also be use to audit business processes.

![Diagram of process mining techniques](image)

Figure 2.3: The three basic types of process mining are explained in terms of input and output: (a) discovery, (b) conformance checking, and (c) enhancement. [3]

The basic usage of process mining is summarized in three classes of process mining techniques as follows (Figure 2.3) [3]:

- **Discovery**: process discovery is a process mining technique that constructs a process model from an event log.

- **Conformance Checking**: conformance checking is a technique to align a process model with an event log. Conformance checking checks whether reality in form of an event log conforms to a process model.

- **Enhancement**: enhancement is a technique to improve an existing process model by using information about the actual process that recorded in an event log. For example, timestamps in an event log can be used to extend the model in order to find bottlenecks, throughput times, service levels, and frequencies.

This thesis is only related to **conformance checking**. **Discovery** and **Enhancement** of process mining are outside the context of this thesis. Section 2.2.1 describes **conformance checking** in multiple perspectives.
2.2.1 Conformance Checking

Conformance checking can have multiple perspectives such as control-flow, data, organizational, and time. The control-flow perspective focuses on the sequence of activities in the model. The main objective of this perspective is to find all possible paths or sequences. The data perspective focuses on the attributes or variables. The organizational perspective focuses on resources in the log, i.e. people, systems, roles, or departments. The time perspective is related to when and how many times an event occurs. Conformance checking measures a comparison between a process model and an event log. We call that comparison as an alignment.

Alignment works by comparing activities observed in a trace (moves in log) to executions of activities in a process model (moves in model). Table 2.2 shows an alignment in control-flow perspective for a trace in the road fines management DPN-net[1]. The left side of the table shows moves in an event log trace while the right side shows a possible run in the process model, we also call it as a process trace or process projection. Table 2.2 shows that there are some differences between the moves in the event-log trace and the moves in process. In the event-log trace, Send Fine and Inv3 do not exist as executed activities such as moves in model (the right side). The alignment calculates deviations between each move in log trace and move in model in term of cost. The more aligned a trace and a process model the less cost it has. An optimal alignment is an alignment which has the lowest cost.

<table>
<thead>
<tr>
<th>Event-log trace</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Fine</td>
<td>Create Fine</td>
</tr>
<tr>
<td>≫</td>
<td>Send Fine</td>
</tr>
<tr>
<td>Notification</td>
<td>Notification</td>
</tr>
<tr>
<td>Payment</td>
<td>Payment</td>
</tr>
<tr>
<td>≫</td>
<td>Inv3</td>
</tr>
</tbody>
</table>

Table 2.2: Control-flow perspective alignment of a log trace the road fines management DPN-net[1]

The alignment, cost function, and optimal alignment definition are adopted from [1] and [4] as follow:

Definition 6 (Alignment). Let \( N = (P,T,F,V,U,\text{Val},W,G) \) be a DPN-net with initial marking \( M_I \) and final marking \( M_F \). Let \( S_N^P \) be a DPN-net \( \{ \gamma \} \) and \( M_F \). Let \( S_N^P = S_N \cup \{ \gamma \} \). A legal move in an alignment is represented by a pair \( (s_L, s_M) \in (S_N^P \times S_N) \) such that:

- \( (s_L, s_M) \) is a move in log if \( s_L \in S_N \) and \( s_M = \gamma \),
- \( (s_L, s_M) \) is a move in model if \( s_L = \gamma \) and \( s_M \in S_N \)
- \( (s_L, s_M) \) is a move in both with correct write operations if \( s_L \in S_N, s_M \in S_N \) and \( \#_{\text{act}}(s_L) = \#_{\text{vars}}(s_L, v) = \#_{\text{vars}}(s_M, v) \),
- \( (s_L, s_M) \) is a move in both with incorrect write operations if \( s_L \in S_N, s_M \in S_N \) and \( \#_{\text{act}}(s_L) = \#_{\text{vars}}(s_M, v) \),
- All other moves are considered as illegal.

Definition 7 (Cost Function & Optimal Alignment). Let \( N \) and \( \sigma_L \) be a DPN-net and log trace, respectively. Assuming \( \mathcal{A}_N \) as the set of all legal alignment moves, a cost function \( K \) assigns a non-negative cost to each legal move: \( \mathcal{A}_N \rightarrow \mathbb{R}_+^+ \). The cost of an alignment \( \gamma \) between \( \sigma_L \) and \( N \) is computed as the sum of the cost of all constituent moves: \( K(\gamma) = \Sigma_{(s_L, s_M)\in \gamma} K(s_L, s_M) \). Alignment \( \gamma \) is an optimal alignment if, for any complete alignment \( \gamma' \) of \( N \) and \( \sigma_L \), \( K(\gamma) \leq K(\gamma') \).

An event log may contain traces with similar path or route. A trace variant is a collection of traces which have the same control-flow perspective. Figure 2.4 shows 56,482 traces have the same variant (Create Fine, Send Fine, Insert Fine Notification, Add Penalty, Send for Credit...
The difference between each trace within the 56,482 traces is information or data regarding an individual recorded event such as attributes and timestamp. As a consequent, if we rely on conformance checking with only one perspective such control-flow perspective, we cannot distinguish traces with the same variants such as the 56,482 traces in Figure 2.4. Therefore, a multi-perspective conformance checking: control-flow and data perspective (e.g., [1]) is chosen.

A trace conforms to the reference process model, if it has correct control-flow perspective and data perspective. Correct control-flow perspective means the process model is able to produce a path or route that recorded in a trace in of an event log and the process starts at initial marking(s) and ends at final marking(s). By using the same example in section 3.1, Figure 2.5 shows an example of a process model DPN-net with initial marking at the place start. The black coloured transition is an invisible transition that is not recorded in event logs. The possible recorded event logs from the model is either \((\text{Practice Exam, Exam})\) or \((\text{Practice Exam, Exercise, Exam})\). If, for example, there is a trace \((\text{Exam, Exercise, Practice Exam})\), it does not have correct control-flow perspective because there is no path exists that able to replay the trace.

A correct data perspective means a trace in an event log does not violate any guard in the process model. In Figure 2.5, the transition Exercise has a guard \((\text{Result} == \text{"failed"})\). The Result is called an attribute or variable and "failed" is its value. If for example there is a trace \((\text{Practice Exam, Exam})\) and with the attribute Result = "failed" at the end of Practice Exam, it does not have correct data perspective because it violates the guard \((\text{Result} == \text{"succeed"})\) at the invisible transition. This example has a correct control-flow perspective but a wrong data perspective. As a consequence it does not conform to the reference process model.
CHAPTER 2. PRELIMINARIES

The following definition are adopted from [1] and [4]:

**Definition 8 (Fitness & Conformance).** Let $\sigma_L \in S^*_N$ be a log trace and let $N$ be a DPN-net. Let $\gamma_O \in A_N^*$ be an optimal alignment of $\sigma_L$ and $N$, and $\gamma_E \in A_N^*$ be an optimal alignment of the empty trace and $N$. The fitness level of $\sigma_L$ and $N$ is defined as follows:

$$F(\sigma_L, N) = 1 - \frac{K(\gamma_O)}{K(\gamma_E) + \sum_{s \in \sigma_L} K((s, \emptyset))}$$

A trace $\sigma_L$ has correct **conformance** to $N$ if $F(\sigma_L, N) = 1$. 

12 Combining Event Logs with Supplementary Events
Chapter 3

Analysis

Several decades ago, a database was a new concept where information systems are designed to store data only for a specific function in an organization. For example, a purchasing system would store and use a purchasing file only. Each specific function in an organization used data files dedicated to that system. This approach is called the traditional approach to data management[16]. Nowadays, most organizations use modern integrated database in multiple information systems (i.e., DBMS). In workflow systems, the information of an executed process is usually fragmented in multiple information systems [17]. As a consequence, information related to process execution is scattered in multiple event logs. Extending an event log with an additional event is quite a challenge. It is difficult to find correlating attribute between events because currently there is no standard guideline on how to correlating process events exists. A process may have multiple identifiers such as both ClientID and TransactionID which causes many ways to define event correlation [18]. However, even though single identifier exists, modern business process may happen concurrently [19], e.g. a client with specific ID may be involved in multiple business processes in the same time, thus ID attribute is not always reliable in correlating events. Therefore, it is assumed that there is no attribute that allows event logs to be trivially correlated.

The approach in [4] extends event logs with supplementary events with a help of conformance checking by using a process model. However, this approach has a limitation with regard to optional activities. This chapter will explain the problem with the approach in [4] with correlating an event log to optional activities and analyse what is the root cause. It also explain what approaches can be used to solve the problem.

Section 3.1 explains the problem with optional activities in the approach in [4]. Section 3.2 describes the root cause and proposes solutions to solve the problem.

3.1 Correlating Optional Activities

In process modeling, an exclusive choice pattern is a node in the process where a path is chosen from several possible paths based on a decision or data[20]. In DPN-net, an optional activity is a self-loop node transition or a transition in an exclusive choice pattern that involved with an invisible transition as one of the choice. Figure 3.1 shows an example of fictional process. The process begins when student does the Practice Exam. If the student fails he needs to do the Exercise before doing the Exam. The Exercise is an optional activity, if a student does not need to do the Exercise, the process continues with the invisible transition, we called it as Inv and it will not be recorded in event logs. The possible routes of this process are [Practice Exam, Exercise, Exam] and [Practice Exam, Inv, Exam]. We call an event log that used as a basis for extension as a main event log. It is assumed in this scenario that the main event log only records information
CHAPTER 3. ANALYSIS

about the executed activities Practice Exam and Exam, while information regarding the executed
optional activity Exercise is logged in different system (different log).

Figure 3.1: Process model with an optional activity Exercise

Given the process model in Figure 3.1, a main event log \( L = \langle (\text{Practice Exam}, \{\text{Result}=\text{failed}\}), \text{Exam} \rangle, \langle (\text{Practice Exam}, \{\text{Result}=\text{succeed}\}), \text{Exam} \rangle \rangle \), and a multi set of supplementary events \( S = \{\text{Exercise}\} \) and it is known that the Exercise is only executed when the result is failed \( \{\text{Result} = \text{failed}\} \). The current approach cannot correlate the optional activity Exercise. As the activity Exercise is optional in the process model, the extension algorithm used in [4] will always prefer not to execute the Exercise activity to build a trace with an optimal alignment.

For example, the trace \( \langle (\text{Practice Exam}, \{\text{Result}=\text{failed}\}), \text{Exam} \rangle \) will be aligned to the same process trace \( \langle (\text{Practice Exam}, \{\text{Result}=\text{failed}\}), \text{Exam} \rangle \rangle \) instead of the desired process trace \( \langle (\text{Practice Exam}, \{\text{Result}=\text{failed}\}), \text{Exercise}, \text{Exam} \rangle \rangle \).

Figure 3.2: Process model with a repeatable optional activity Exercise.

A similar problem exists for DPN-nets that contain a repeatable optional activity as shown in
Figure 3.2. The approach in [4] cannot correlate the main event logs \( L \) to the repeatable optional
activity Exercise.
3.2 Root Cause Analysis

Figure 3.3 shows the overview of the approach in [4]. First, the algorithm computes an alignment with the lowest cost (optimal alignment) of each trace in main event log. Then, the process projection or process moves in model of the alignment is used as the basis for extending main event logs with supplementary events. The alignment does not contain an optional activity because the algorithm in [4] prefers the shortest alignment with the lowest cost. Therefore, the extended main log does not contain optional activity. The following example will explain the problem in [4] in more detail:

- Main event log: 
  \[(\text{Practice Exam}, \{\text{Result}=\text{failed}\}),\text{Exam}\]
- Supplementary event: \(S = [\text{Exercise}^5]\)
- Process model: Process model with an optional activity. (Figure 3.1)

<table>
<thead>
<tr>
<th>Desired Alignment</th>
<th>Optimal Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Event-log Trace</strong></td>
<td><strong>Process</strong></td>
</tr>
<tr>
<td>Practice Exam</td>
<td>Practice Exam</td>
</tr>
<tr>
<td>(\rightarrow)</td>
<td>Exercise</td>
</tr>
<tr>
<td>Exam</td>
<td>Exam</td>
</tr>
<tr>
<td><strong>Cost</strong></td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.1: Alignment Results

The alignment from this example process is shown in Table 3.1. A move in log \((s_L, \rightarrow)\) or move in model \((\rightarrow, s_M)\) are assumed to have cost 1. The left side shows the desired of alignment and the right side shows the optimal alignment. In the left side of Table 3.1, the state \((\rightarrow,\text{Exercise})\) is a move in log with a cost of 1. The total cost of the alignment in the left side is 1. Whereas in the right side of Table 3.1, the state \((\rightarrow,\text{Inv})\) is a move in model with a cost of 0. The move in model involving an invisible transition is normally has no cost or ignored because an invisible transition is not recorded in an event log. Therefore the right side of Table 3.1 is the optimal alignment since it has lower cost (0 vs 1). The alignment algorithm itself relies on \(A^*\) algorithm which search for an optimal alignment [21] [22]. Therefore, an optimal alignment is always preferred. Even if we give invisible transition the same cost as a regular transition that will cause both alignment in the left and right side have the same cost, the optimal alignment will be selected arbitrarily. This arbitrarily decision is caused by the alignment algorithm itself which relies on the \(A^*\) algorithm. Therefore, this is not a reliable way to extend an optional activity.

Figure 3.4 summarized the root causes of the problem. From the root cause analysis, the first cause of the problem is originated by the use of an alignment. Consequently, the first approach proposes a solution without using an alignment. The second and the third causes are originated by the optimal alignment. Therefore, the second approach proposes another way of using an alignment. The proposed approaches are summarized as follows:
CHAPTER 3. ANALYSIS

Incapable of correlating optional activities

Why is it incapable of correlating optional activities?

Approach 1:
Ignore alignment
Heuristic Based Log Extension

The result from alignment does not contain any move with optional activities

Why does the result not contain any move with optional activities?

A move without optional activity is preferred because it has lower cost

Why does a move with optional activity have a higher cost?

Optional activity is missing in the aligned trace

Approach 2:
Modify trace before alignment to make move with optional activity included in the set

Figure 3.4: Root cause analysis of current problem.

• Approach 1: extend a main event log by using heuristics directly without any alignment.
• Approach 2: extend a main event log before measures alignment as illustrated in Figure 3.5.

Figure 3.5: Overview of the second approach.
Chapter 4

Design

Heuristic is a problem solving approach that solves problem by using simple logic, rules, and experiences. The heuristic itself is originally come from Greek word *heuriskein* which means to discover. The proposed solutions in this thesis are using heuristics. Consequently, this approach seeks for discovering the correlation between supplementary event and main event logs by using some guidelines as a decision maker for extending logs.

*One*, a robot may not injure a human being, or through inaction, allow a human being to come to harm; *Two*, a robot must obey the orders given it by human beings, except where such orders would conflict with the First Law; *Three*, a robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

Three Laws of Robotics - Isaac Asimov

The three laws of robotics is an example of heuristics method. Studies show that our mind uses heuristics to cope with the complexity inherent in most decisions we make, and they perform well in most situations [23]. In the early 1970s, the work [24] explained how human commonly uses heuristics under an uncertain situation, such as predicting a probability of an supplementary event belong to a trace. As currently there is no solution for correlating supplementary events to a main event log using a process model with optional activities currently known, heuristics are an efficient procedure in an attempt to find a good solution [25]. Furthermore, as there is no attribute that could correlate event trivially exists, a greedy algorithm, an example of a heuristic is used to correlate an supplementary event.

This chapter discusses the heuristic approach to extend process logs with additional supplementary events. Section 4.1 introduces the first approach that ignore the alignment. Section 4.2 introduces the second approach, using the alignment as heuristics.

4.1 Approach 1: Heuristics Based Logs Extension

This section describes the first approach to extend an event log with supplementary events that are extracted from additional sources. Beside the event name itself, the event log has information that can be used in an heuristic such as *timestamps* and *attributes*. By looking at the event log and a process model, the first approach uses the following heuristics:
**One,** a trace is extended only if it does not conform to the reference process model;

**Two,** an extended trace must have correct time order;

**Three,** an extended trace must conform to the reference process model.

Section 4.1.1 discusses the first approach. Section 4.1.2 discusses the functions in Section 4.1.1 in more details. Section 4.1.3 discusses the computational complexity of the algorithm.

### 4.1.1 Algorithm

The input of the algorithm is:

1. a DPN-net $N = (P,T,F,V,U,Val,W,G)$ with a given initial and final marking;
2. an event log $L \in B(S_N^*)$ that records the firing of transitions.
3. a multiset of supplementary events $E \in B(S_N^*)$.

The output of the algorithm is an extended event log $L_E$.

**Algorithm 1** `extendLogHeuristics`

**Input:** DPN-net $(N)$, Event Log $(L)$, Supplementary Events $(S)$.

**Output:** Extended Event Log $(L_E)$.

1. foreach $\sigma_L \in L$ do
2.   $\text{foundCandidate} \leftarrow \text{false}$
3.   if $\neg \text{ISCONFORM}(\sigma_L, N)$ then
4.     $P_\sigma \leftarrow \text{GETPOSSIBLEEXTENSION}(\sigma_L, N, S)$
5.     while $P_\sigma\.\text{hasNext}$ and $\neg \text{foundCandidate}$ do
6.       $\text{P}_\sigma,i \leftarrow P_\sigma\.\text{next}$
7.       $C_1 \leftarrow \text{GETCANDIDATEBYTIME}(\text{P}_\sigma,i,S)$
8.       $C_2 \leftarrow \text{GETCANDIDATEBYDATA}(\sigma_L,N,C_1,i)$
9.       if $C_2 \neq \emptyset$ then
10.      $e \leftarrow \text{SELECTCANDIDATE}(C_2)$
11.     $\sigma_e \leftarrow \sigma_L\.\text{add}(e,i)$
12.     $L_E \leftarrow L_E \cup \{\sigma_e\}$
13.     $\text{foundCandidate} \leftarrow \text{true}$
14.   end if
15. end while
16. end if
17. if $\neg \text{foundCandidate}$ then
18.   $L_E \leftarrow L_E \cup \{\sigma_L\}$
19. end if
20. end for
21. return $L_E$

$N$ : DPN-net.
$L$ : main event log.
$S$ : supplementary events.
$L_E$ : extended event log.
$\sigma_L$ : log trace.
$P_\sigma$ : possible extended traces that conform to model in control-flow perspective.
$\sigma_P$ : possible extended log trace.
Algorithm 1 extends log traces in $\sigma_L \in L$ by adding supplementary events in $S$ to $\sigma_L$ in possible positions that are allowed to conform to process model. First, line 3 checks whether trace $\sigma_L$ conforms with the model control-flow perspective and data perspective. If a trace does not conform, the function ISCONFORM will return false and the trace will be extended. Line 4 gives possible extension that are allowed by the model in the control-flow perspective. Next, by using the possible trace $\sigma_M$ in line 7, the supplementary events are filtered by time and data in line 7 and 8. The function GETCANDIDATEBYTIME filters supplementary events that with correct time order as event candidates $C_1$ with the trace possible extended trace $\sigma_P$ and the function GETCANDIDATEBYDATA filters event candidates $C_2$ that conform with the data perspective of the model. The functions GETCANDIDATEBYTIME and GETCANDIDATEBYDATA guarantee the event candidates $C_2$ will satisfy time order and conform with the model if it used in possible trace $\sigma_M$. Then, line 10 the function SELECTCANDIDATE selects an event that will be added in line 11. Finally, the algorithm returns the extended event log $L_E$ in line 21.

In this algorithm, the first event candidate is selected by the function SELECTCANDIDATE. Although there is unlinked data between main event log and supplementary events, both event log and supplementary events are usually ordered by the time when a process is started. Therefore, there is a higher probability that the first event in supplementary events is related with the first trace in main event log. This approach depends on the conformance of a process model. It requires an event log and a model with a perfect conformance.

In the following, the functions in Algorithm 1 will be illustrated in example used in Section 3.2. Let DPN-net in Figure 4.1, event log in Figure 4.2, and supplementary events in Figure 4.3 be an input for Algorithm 1:

- DPN-net $N$: process model example Figure 4.1.
- Event Log $L$: $\langle (\text{Practice Exam}, \{ Result=\text{failed}\}), \text{Exam} \rangle, \langle (\text{Practice Exam}, \{ Result=\text{succeed}\}), \text{Exam} \rangle$
- Supplementary events $S$: $\langle \text{Exercise} \rangle$.

First, the first trace (trace number 1) is examined by the function ISCONFORM. The function ISCONFORM checks the control-flow perspective and the data perspective. According to the model in Figure 4.1, it is possible to run the first trace in the model (Practice Exam, $INV$, Exam). $INV$ is an invisible transition and it is not recorded in the event log. Consequently the control-flow perspective of the first trace is correct. However by the data perspective, the first trace is wrong.
because the invisible transition $INV$ should only be executed when the attribute $Result = "succeed"$. Hence, the first trace does not conform to the model. Accordingly, Algorithm 1 will try to extend the first trace.

After checking conformance, the function GETPOSSIBLEEXTENSION will get all possible traces that are allowed in the control-flow perspective. The function GETPOSSIBLEEXTENSION works by greedily adding supplementary event to every position in the trace as illustrated in Figure 4.4. The function returns only possible traces that are allowed by the control-flow perspective. Figure 4.4 shows there are three possible traces which are (Exercise,Practice Exam,Exam), (Practice Exam,Exercise,Exam), and (Practice Exam,Exam,Exercise). The check mark in the figure indicates a trace that conforms to the control-flow perspective while the cross mark indicates a trace that does not conform to control-flow perspective. The only possible trace for extension is (Practice Exam,Exercise,Exam).

Next, the function GETCANDIDATEBYTIME will filter supplementary events by time order which means it searches for supplementary event $Exercise$ that happened after $Practice Exam$ and before $Exam$. Figure 4.5 illustrates the function GETCANDIDATEBYTIME. The only can-

Figure 4.2: Main Event Log

Figure 4.3: Supplementary Events
candidate $C_1$ that has correct time order is the Exercise with Timestamp : 15 Mar 15. Then, the event candidates $C_1$ is filtered by the function GETCANDIDATEBYDATA. The function GETCANDIDATEBYDATA filters candidates by greedily adding each event in $C_1$ and checking whether the added trace is allowed by the data perspective. The only candidate in this example is the Exercise with Timestamp : 15 Mar 15.

Then, the function SELECTCANDIDATE selects the first candidate in $C_2$ and the selected event $(e)$ is added to the trace $\sigma_L$ by the function ADDSUPPLEMENTARYEVENT as a extended trace $\sigma_e$. Finally, the extended trace $\sigma_e$ is added to the extended event log $L_E$. Figure 4.6 shows the output of Algorithm 1 for this example. The second trace is not extended, because it already conforms to the process model.

**4.1.2 Function**

This section describes the functions in Algorithm 1 in more details. The functions that will be described are ISCONFORM, GETPOSSIBLEEXTENSION, GETCANDIDATEBYTIME and GETCANDIDATEBYDATA.
CHAPTER 4. DESIGN

ISCONFORM

Function ISCONFORM checks a log trace $\sigma_L$ conformance in the control-flow perspective and the data perspective. Function CHECKCONTROLFLOW checks whether the $\sigma_L$ can be replayed in process model $N$. Function CHECKDATA checks if the $\sigma_L$ satisfies all guards in process model $N$. The pseudo-code of both functions are available in Algorithm 3 and 4.

Algorithm 2 ISCONFORM

1: function ISCONFORM($N, \sigma_L$)
2:     return CHECKCONTROLFLOW($N, \sigma_L$) and CHECKDATA($N, \sigma_L$)
3: end function

The control-flow perspective is checked by the function CHECKCONTROLFLOW in Algorithm 3. First the log trace $\sigma_L$ is replayed with the function REPLAYLOG in line 2, then the function FINALMARKINGREACHABLE in line 3 checks whether final marking in process model $N$ is reachable. The process model $N$ is assumed to contain both initial and final markings. The control-flow perspective is correct, if all events in $\sigma_L$ are able to be replayed in the model and the final markings are reachable. Both the function REPLAYLOG and FINALMARKINGREACHABLE are implemented by using DataAwareReplayer package within ProM.

Algorithm 3 CHECKCONTROLFLOW

1: function CHECKCONTROLFLOW($N, \sigma_L$)
2:     finalMarking ← REPLAYLOG($N, \sigma_L$)
3:     if FINALMARKINGREACHABLE(finalMarking, $N$) then
4:         return true
5:     else
6:         return false
7:     end if
8: end function

In the function CHECKDATA in Algorithm 4, the data or variables are updated for every event $e$ in log trace $\sigma_L$ by the function UPDATEDATA. The function CHECKDATA returns false if there any data that violates a guard in the process model $N$. Both the functions UPDATESTATE and ISGUARDTRUE are implemented DataAwareReplayer package within ProM. In the function ISGUARDTRUE, the event $e$ is mapped to transition(s) in $N$ by accessing the transition in firing sequence $\#act(s) = t$.

Algorithm 4 CHECKDATA

1: function CHECKDATA($N, \sigma_L$)
2:     foreach $e \in \sigma_L$ do
3:         data ← UPDATEDATA($e$)
4:         if ¬ISGUARDTRUE($e, data, N$) then
5:             return false
6:         end if
7:     end for
8:     return true
9: end function
Algorithm 5 GETPOSSIBLEEXTENSION

1: function GETPOSSIBLEEXTENSION(\(\sigma_L, N, S\))
2: \(P_\sigma \leftarrow \{}\)
3: \(s_e \leftarrow S.first\)
4: for \((i = 0; i \leq \sigma_L.size; i++)\) do
5: \(\sigma_P \leftarrow \sigma_L.add(s_e, i)\)
6: if CHECKCONTROLFLOW\((N, \sigma_P)\) then
7: \(P_\sigma \leftarrow P_\sigma \cup \{(\sigma_P, i)\}\)
8: end if
9: end for
10: return \(P_\sigma\)
11: end function

GETPOSSIBLEEXTENSION

Function GETPOSSIBLEEXTENSION generates possible new traces by looking only at the control-flow perspective. First, a supplementary event \(s_e\) is chosen from supplementary events \(S\) in line 3. In the control-flow perspective any event in \(S\) is assumed to refers to the same activity, therefore it does not matter which event is taken from \(S\). However, it makes the first approach can only extend a single type of supplementary events. Then, line 4 to 9 greedily adding event \(s_e\) to log trace \(\sigma_L\). Line 6 assures only a control-flow conforming trace \(\sigma_P\) is returned.

Algorithm 6 GETCANDIDATEBYTIME

1: function GETCANDIDATEBYTIME(\(\sigma_P, S\))
2: \(C_1 \leftarrow \{}\)
3: foreach \(e \in S\) do
4: if CHECKTIME\((\sigma_P, e)\) then
5: \(C_1 \leftarrow C_1 \cup \{e\}\)
6: end if
7: end for
8: return \(C_1\)
9: end function

\(\sigma_P\) is a log trace that already contains arbitrary supplementary event. Function GETCANDIDATEBYTIME filters supplementary events \(S\) by checking the time stamp of an event \(e\) in its position within \(\sigma_P\) with its previous and next events in \(\sigma_P\). If the time stamp is correct i.e., time stamp of previous event \(\leq\) time stamp of \(e\) \(\leq\) time stamp of next event, then that event \(e\) is a candidate in \(C_1\).

GETCANDIDATEBYDATA

First, an event \(e\) from event candidates \(C_1\) is added to a trace \(\sigma_L\). Then, function CHECKDATA in Algorithm 4 verifies the conformance of the replaced log trace \(\sigma_R\) in the data perspective. If \(\sigma_R\) conforms to the process model \(N\) then the event \(e\) is added as a candidate to event candidates \(C_2\). The function GETCANDIDATEBYDATA assures that each event \(e\) in the event candidates \(C_2\) conforms to the process model \(N\) in data perspective.
Algorithm 7 \textsc{GetCandidateByData}

\begin{algorithm}[H]
1: function \textsc{GetCandidateByData}(\sigma_L, N, C_1, i) \\
2: \hspace{1em} C_2 \leftarrow \{\} \\
3: \hspace{1em} \textbf{foreach} \ e \in C_1 \ \textbf{do} \\
4: \hspace{2em} \sigma_R \leftarrow \sigma_L.add(e, i) \\
5: \hspace{2em} \textbf{if} \ \textsc{CheckData}(N, \sigma_R) \ \textbf{then} \\
6: \hspace{3em} C_2 \leftarrow C_2 \cup \{e\} \\
7: \hspace{2em} \textbf{end if} \\
8: \hspace{1em} \textbf{end for} \\
9: \hspace{1em} \textbf{return} C_2 \\
10: \textbf{end function}
\end{algorithm}

4.1.3 Computational Complexity

In worst case, function \textsc{GetPossibleExtension} will be repeated as many times as the maximum number of events a trace \sigma_L in an event log \( L \) and the internal \textit{while} loop is repeated as many times as the size of \( P_r \) which is the number of events in \( \sigma_P \). Function \textsc{GetCandidateByTime} and \textsc{GetCandidateByData} are repeated as many times as the number of supplementary events. Therefore, approximately the worst case complexity is \( O(|L| \times (|\sigma_L|_{\text{max}} + |\sigma_L|_{\text{max}} + (|S| + |S|^2))) \) with \(|\sigma_L|_{\text{max}} \) is the maximum size of trace \( \sigma_L \) in \( L \). However, this is rarely happened because the heuristics and process model limit the number of repetition.

4.2 Approach 2: Alignment Based Logs Extension

Approach 2 extends log by adding supplementary events in the most conform way by using alignment. This section describes the second approach to extend an event log with supplementary events by using an alignment. In a similar way as in Section 4.1, the extended log has to conform to the model with a correct time order. The second approach has the same input and output as Algorithm 1. The second approach uses the following heuristics:

- **One**, a trace is extended only if its alignment cost is bigger than zero;
- **Two**, an extended trace must have correct time order;
- **Three**, an extended trace must have lower or the same cost compared to before.

A trace is only extended if it does not perfectly conform to a process model. If a trace has a zero cost of its alignment then the trace has a perfect conformance where a supplementary event does not need to be added. The reason an extended trace must have a lower or equal cost than before is because of the following reasons. Firstly, if an incorrect event is added to a trace then its extended trace will have a higher cost and if the event added event is correct then the extended will have a lower cost. Secondly, for a trace that is extended by an optional activity, the extended trace may have the same cost as before because of the optional activity self-loop property that is difficult to be detected by an alignment. As for these reasons, an extended trace must have a lower or equal cost than before.

Section 4.2.1 discusses the second approach. Section 4.2.2 discusses the functions in Section 4.2.1 in more details. Section 4.2.3 discusses the computational complexity of the algorithm.

4.2.1 Algorithm

First, Algorithm 8 aligns a log trace \( \sigma_L \) and calculates its cost in line 3 to 4. Function \textsc{Align} and \textsc{GetProcessesTrace} are implemented by using another work such as the one in [1]. Only
a trace with a cost higher than 0 is extended. Next, the function GETPROCESSTRACE in line 6 gets a process trace (or process projection) $\sigma_P$ that is used by the alignment $\gamma$. Then, the function ISINPROCESSTRACE checks whether the supplementary event $e$ is in the process trace $\sigma_P$. If the event $e$ is in the process trace $\sigma_P$ and not in the $E_{\text{added}}$ then the function EXTENDTRACE in line 11 adds event $e$ to log trace $\sigma_L$ in the same position as process trace $\sigma_P$. Sequentially, line 11 to 13 check if the extended trace cost ($\text{extended\_cost}$) is lower or equal the cost before extension ($\text{current\_cost}$). Line 10 to 14 makes sure no supplementary event in $\sigma_P$ is added twice.

In case the supplementary event $e$ is not in the process trace $\sigma_P$ then the supplementary event $e$ is an optional activity. The function GETPOSSIBLETRACES in line 19 finds all possible extensions of the log trace $\sigma_L$ with event $e$ by greedily adding event $e$ into all position in $\sigma_L$. The greedy event addition is similar as the function GETPOSSIBLEEXTENSION in Algorithm 1. Function SELECTTRACE selects one of the possible extended trace that has the lowest cost. If there are two possible traces which cost are same, the function SELECTTRACE will select the first one.

**Algorithm 8** extendLogAlignment

**Input**: DPN-net($N$), Event Log ($L$), Supplementary Events ($S$).

**Output**: Extended Event Log ($L_E$).

1: foreach $\sigma_L \in L$ do
2:     $E_{\text{added}} \leftarrow \{\}$
3:     $\gamma \leftarrow \text{ALIGN}(N, \sigma_L)$
4:     $\text{current\_cost} \leftarrow K(\gamma)$
5:     if $\text{current\_cost} \neq 0$ then
6:         $\sigma_P \leftarrow \text{GETPROCESSTRACE}(\gamma)$
7:         foreach $e \in S$ do
8:             if $\text{current\_cost} \neq 0$ then
9:                 if ISINPROCESSTRACE($e, \sigma_P$) then
10:                    if $e \notin E_{\text{added}}$ then
11:                        $(\sigma_e, \text{isAdded}) \leftarrow \text{EXTENDTRACE}(N, e, \sigma_L, \sigma_P, \text{current\_cost})$
12:                        if $\text{isAdded}$ then
13:                            $S \leftarrow S \setminus \{e\}$
14:                            $\sigma_L \leftarrow \sigma_e$
15:                            $E_{\text{added}} \leftarrow E_{\text{added}} \cup \{e\}$
16:                        end if
17:                     end if
18:                 else
19:                     $P_o \leftarrow \text{GETPOSSIBLETRACES}(\sigma_L, N, e, \text{current\_cost})$
20:                     if $P_o \neq \emptyset$ then
21:                         $(\sigma_L, \text{extended\_cost}) \leftarrow \text{SELECTTRACE}(P_o)$
22:                         $S \leftarrow S \setminus \{e\}$
23:                         $\text{current\_cost} \leftarrow \text{extended\_cost}$
24:                     end if
25:                 end if
26:             end if
27:         end for
28:     $L_E \leftarrow L_E \cup \{\sigma_L\}$
29: end if
30: end for
31: return $L_E$

$\gamma, \gamma_e$ : alignment before extension, alignment after extension.
CHAPTER 4. DESIGN

$E_{added}$: list of events that already added.
current_cost, extended_cost: cost before extension, cost after extension.
$\sigma_L, \sigma_P, \sigma_e$: log trace, process trace, extended trace.
$P_\sigma$: possible extended traces that has better conformance than original trace.
e: selected supplementary event.

In the following example, Algorithm 8 will be illustrated. Let a DPN-net in Figure 4.7, an event log in Figure 4.8, and supplementary events in Figure 4.9 be an input for Algorithm 8:
- DPN-net $N$: process model example Figure 4.7.
- Event Log $L$: $[\langle \text{Practice Exam}, \{\text{Result}=\text{failed}\}\rangle, \text{Exam}\rangle, \langle \text{Practice Exam}, \{\text{Result}=\text{succeed}\}\rangle, \text{Exam}\rangle$.
- Supplementary events $S$: [Exercise$^3$].

![Figure 4.7: DPN-net example 2](image)

![Figure 4.8: Main Event Log](image)

![Figure 4.9: Supplementary Events](image)

First, the first trace (trace number 1) is aligned with the process model $N$ by the function ALIGN. Then the current trace cost is obtained from the function $K$. The result of PracticeExam is failed but the guard in the transition Exam requires the result to be succeed, therefore the first trace violates the data perspective in the guard in the transition Exam. If it is assumed that move
in model and move in log cost 1, then the first trace will have total cost of 1. Followed by GETPROCESSTRACE which gives the process trace used in the alignment, the process trace for this example is illustrated in Figure 4.10. Subsequently, Algorithm 8 checks each supplementary event $e$ whether it is in the process trace through the function ISINPROCESSTRACE. As shown in Figure 4.10, the supplementary event $Exercise$ is not in the process trace, therefore the algorithm continues with the function GETPOSSIBLETRACES.

The function GETPOSSIBLETRACES returns the possible extended traces $P_\sigma$ in Figure 4.11. Inside the function GETPOSSIBLETRACES in Algorithm 10, the extended trace cost is checked. The first supplementary event $Exercise$ with a timestamp 17 Aug 15 is not extended because it has incompatible timestamp that makes the extended cost higher. After checking the first supplementary event, the algorithm checks the next $Exercise$ with a timestamp 25 Feb 15. The extended trace in this case is shown in Figure 4.12. It costs 1 and it has the same cost with the original trace $\sigma_L$. The $Exercise$ Result is 5 and it still violates the guard on transition $Exam$ which requires Exercise Result $> 5$. Since the extended cost is equal to the cost before extension, the algorithm 10 returns the extended trace in Figure 4.12 as possible extended traces. Then function SELECTTRACE selects the possible extended traces with the lowest cost. There is only one possible trace returned from GETPOSSIBLETRACES, therefore the current extended trace in Figure 4.11 is selected.
Eventually, the last supplementary event, *Exercise* with a timestamp 15 Mar 15 undergoes the same thing. The cost for this extension is 0, because it satisfies the guard on *Exam*; hence, the trace is extended with the last *Exercise*. Afterwards, the algorithm continues with the second trace. The remains supplementary event is the *Exercise* with a timestamp 17 Aug 14 which has incompatible timestamp for the second trace. Therefore, the second trace is not extended. Finally, the algorithm returns extended log result in Figure 4.13.

Algorithm 8 extends a trace with multiple supplementary events. A variant of Algorithm 8 which only extends a trace with a single supplementary event is shown in Algorithm 9. The function GETCANDIDATEBYTIME work in a similar way as in Algorithm 6 in the first approach in Section 4.1.1. While the function SELECTTRACE works the same way as in Algorithm 8. The function $K_C$ returns a *control-flow perspective* cost.

First the algorithm get a cost of the current trace $\sigma_L$, then the function GETPOSSIBLEEXTENSIONSINGLE generates possible new traces by looking only in the *control-flow perspective*. Then the function GETCANDIDATEBYTIME filters supplementary events that have a correct time order as event candidates $C_1$ with the trace possible extended trace $\sigma_P$. Next, the function EXTENDTRACESINGLE extends the trace $\sigma_L$ by comparing the current cost and the extended cost inside the function EXTENDTRACESINGLE and returns only the best possible extended trace $\sigma$. Finally, the function SELECTTRACE selects an extended trace which has the lowest cost.
**Algorithm 9** extendLogAlignmentSingle

**Input**: DPN-net\((N)\), Event Log \((L)\), Supplementary Events \((S)\).

**Output**: Extended Event Log \((L_E)\).

1: foreach \(\sigma_L \in L\) do
2:  \(\text{foundCandidate} \leftarrow \text{false}\)
3:  \(\gamma \leftarrow \text{ALIGN}(N, \sigma_L)\)
4:  \(\text{current\_cost} \leftarrow K(\gamma)\)
5:  if \(\text{current\_cost} \neq 0\) then
6:      \(\text{current\_control\_cost} \leftarrow K_C(\gamma)\)
7:      \(P_\sigma \leftarrow \text{GETPOSSIBLEEXTENSIONSINGLE}(\sigma_L, N, S, \text{current\_control\_cost})\)
8:      \(P_e \leftarrow \{\}\)
9:      foreach \((\sigma_P, i) \in P_\sigma\) do
10:         \(C_1 \leftarrow \text{GETCANDIDATEBYTIME}(\sigma_P, S)\)
11:         \(E_\sigma \leftarrow \text{EXTENDTRACESINGLE}(\sigma_L, N, \sigma_P, i, C_1, \text{current\_cost})\)
12:         \(P_e \leftarrow P_e \cup \{E_\sigma\}\)
13:   end for
14:   if \(P_e \neq \emptyset\) then
15:      \((\sigma_e, e) \leftarrow \text{SELECTTRACE}(P_e)\)
16:      \(S \leftarrow S \setminus \{e\}\)
17:      \(L_E \leftarrow L_E \cup \{\sigma_e\}\)
18:      \(\text{foundCandidate} \leftarrow \text{true}\)
19:   end if
20: end if
21: if \(!\text{foundCandidate}\) then
22:   \(L_E \leftarrow L_E \cup \{\sigma_L\}\)
23: end if
24: end for
25: return \(L_E\)

\(\gamma\): alignment.

\(\text{current\_cost}, \text{current\_control\_cost}\): cost before extension, current control-flow cost before extension.

\(P_\sigma\): possible extended traces that have better or equal cost with the current\_cost.

\(C_1\): event candidates with a correct time order.

\(P_e\): extended traces candidates.

\(E_\sigma\): \((\sigma_e, \text{extended\_cost}, e)\)

\(\sigma_L, \sigma_P, \sigma_e\): log trace, process trace, extended trace.

### 4.2.2 Function

This section gives the detail of functions in Algorithm 8 and 9. The functions that will be discussed are GETPOSSIBLETRACES, SELECTTRACE, EXTENDTRACE, GETPOSSIBLEEXTENSIONSINGLE, and EXTENDTRACESINGLE.
CHAPTER 4. DESIGN

GETPOSSIBLETRACES

Algorithm 10 GETPOSSIBLETRACES

1: function GETPOSSIBLETRACES($\sigma_L, N, e, \text{current\_cost}$)
2: \hspace{1em} $P_\sigma \leftarrow \{\}$
3: \hspace{1em} for ($i = 0; i \leq |\sigma_L|; i++) do
4: \hspace{2em} $\sigma_P \leftarrow \sigma_L.\text{add}(e, i)$
5: \hspace{2em} $\gamma_e \leftarrow \text{ALIGN}(N, \sigma_L)$
6: \hspace{2em} extended\_cost \leftarrow K(\gamma_e)$
7: \hspace{2em} if (extended\_cost $\leq$ current\_cost) then
8: \hspace{3em} $P_\sigma \leftarrow (\sigma_P, \text{extended\_cost})$
9: \hspace{2em} end if
10: \hspace{1em} end for
11: \hspace{1em} return $P_\sigma$
12: end function

Function GETPOSSIBLETRACES in Algorithm 10 uses the alignment to decide whether the extended trace $\sigma_P$ is conforming better than log trace $\sigma_L$. Function ALIGN, $K$ are part of alignment in [1] and beyond the scope of this thesis.

SELECTTRACE

Algorithm 11 SELECTTRACE

1: function SELECTTRACE($P_\sigma$)
2: \hspace{1em} current\_cost \leftarrow 99999
3: \hspace{1em} $\sigma_S \leftarrow \{\}$
4: \hspace{1em} while $P_\sigma.\text{hasNext}$ do
5: \hspace{2em} $(\sigma_P, \text{extended\_cost}) \leftarrow P_\sigma.\text{first}$
6: \hspace{2em} if (extended\_cost $< current\_cost$) then
7: \hspace{3em} current\_cost \leftarrow extended\_cost
8: \hspace{3em} $\sigma_S \leftarrow \sigma_P$
9: \hspace{2em} end if
10: \hspace{1em} end while
11: \hspace{1em} return ($\sigma_S, current\_cost$)
12: end function

The function SELECTTRACE, returns a trace with the lowest cost in $P_\sigma$.

EXTENDTRACE

The function EXTENDTRACE works by adding the supplementary event $e$ in the current trace $\sigma_L$. First, the index of $e$ ($e_{index}$) is obtained from the process trace $\sigma_P$ in line 4. If $e_{index}$ is located at the first or the last index of $\sigma_P$ then $e$ is added to the first or last index of $\sigma_L$ (line 6 and 8), else the function searches the right index of $e$ in $\sigma_L$ from $\sigma_P$. Line 13 gets the current observed event $e_L$ from the index $i$ in $\sigma_L$. Then line 15 gets the index of $e_L$ ($e_L_{index}$) in $\sigma_P$. Next, if ($e_L_{index}$) $>$ ($e_{index}$) then the position of $e$ is already passed by $e_L$ in $\sigma_P$, therefore we say it is found and we can get the right position of $e$ in $\sigma_L$ by looking the last index of $i$ before the position of $e$ in the log trace is found. Finally, line 31 checks whether the extended trace has a better cost than before.
Algorithm 12 EXTENDTRACE

1: function EXTENDTRACE\((N, e, \sigma_L, \sigma_P, \text{current\_cost})\)
2: \(\sigma_e \leftarrow \{\}\)
3: \(\text{isAdded} \leftarrow \text{false}\)
4: \(e\_\text{index} \leftarrow \sigma_P.get\_index(e)\)
5: \(\text{if } e\_\text{index} = 0 \text{ then}\)
6: \(\sigma_e \leftarrow \sigma_L.add(e, 0)\)
7: \(\text{else if } e\_\text{index} \geq |\sigma_L| - 1 \text{ then}\)
8: \(\sigma_e \leftarrow \sigma_L.add(e, |\sigma_L| - 1)\)
9: \(\text{else}\)
10: \(i \leftarrow 0\)
11: \(\text{isFound} \leftarrow \text{false}\)
12: \(\text{while } ((i < |\sigma_L|) \text{ and } (\neg \text{isFound})) \text{ do}\)
13: \(e_L \leftarrow \sigma_L.get(i)\)
14: \(\text{if } e_L \in \sigma_P \text{ then}\)
15: \(e_L\_\text{index} \leftarrow \sigma_P.get\_index(e_L)\)
16: \(\text{if } e_L\_\text{index} > e\_\text{index} \text{ then}\)
17: \(\text{if } i = 0 \text{ then}\)
18: \(\text{isFound} \leftarrow \text{true}\)
19: \(\sigma_e \leftarrow \sigma_L.add(e, 0)\)
20: \(\text{else}\)
21: \(\text{isFound} \leftarrow \text{true}\)
22: \(\sigma_e \leftarrow \sigma_L.add(e, i - 1)\)
23: \(\text{end if}\)
24: \(\text{end if}\)
25: \(i \leftarrow i + 1\)
26: \(\text{end while}\)
27: \(\gamma_e \leftarrow \text{ALIGN}(N, \sigma_e)\)
28: \(\text{extended\_cost} \leftarrow K(\gamma_e)\)
29: \(\text{if } (\text{extended\_cost} \leq \text{current\_cost}) \text{ then}\)
30: \(\text{isAdded} \leftarrow \text{true}\)
31: \(\text{end if}\)
32: \(\text{return } (\sigma_e, \text{isAdded})\)
33: \(\text{end function}\)
GETPOSSIBLEEXTENSION SINGLE

Algorithm 13 GETPOSSIBLEEXTENSION SINGLE

```plaintext
1: function GETPOSSIBLEEXTENSION SINGLE(σ_L, N, S, current_control_cost)
2:     P_σ ← {} 
3:     s_e ← S.first 
4:     for (i = 0; i ≤ σ_L.size; i++) do 
5:         σ_P ← σ_L.add(s_e, i) 
6:         γ_e ← ALIGN(N, σ_P) 
7:         extended_control_cost ← K_C(γ_e) 
8:         if (extended_control_cost ≤ current_control_cost) then 
9:             P_σ ← P_σ ∪ {(σ_P, i)} 
10:          end if 
11:     end for 
12:     return P_σ 
13: end function
```

The function GETPOSSIBLEEXTENSION SINGLE works in a similar way as Algorithm 5 by using a control-flow cost to check if a possible extended trace σ_P is more conform to the process model N. If the extended_control_cost is ≤ than the current_control_cost then σ_P is added as one of possible traces.

EXTENDTRACESINGLE

Algorithm 14 EXTENDTRACESINGLE

```plaintext
1: function EXTENDTRACESINGLE(σ_L, N, σ_P, i, C_1, current_cost)
2:     selected_event ← {} 
3:     best_σ_e ← {} 
4:     best_cost ← 99999 
5:     foreach e ∈ C_1 do 
6:         σ_e ← σ_L.add(e, i) 
7:         γ_e ← ALIGN(N, σ_e) 
8:         extended_cost ← K(γ_e) 
9:         if (extended_cost ≤ current_control_cost) then 
10:             if (extended_cost ≤ current_control_cost) then 
11:                 best_σ_e ← σ_e 
12:                 best_cost ← extended_cost 
13:                 selected_event ← e 
14:             end if 
15:         end if 
16:     end for 
17:     E_σ ∪ {(best_σ_e, best_cost, selected_event)} 
18:     return E_σ 
19: end function
```

The function EXTENDTRACESINGLE selects a supplementary event in candidates C_1 that gives the extended trace σ_e the lowest possible cost.
4.2.3 Computational Complexity

This algorithm is computationally more expensive than Algorithm 1 considering the computation of an alignment. In the worst case, the function GETPOSSIBLETRACES repeats as many times as the number of traces in \( L \) and the function GETPOSSIBLETRACES will be repeated as many times as the number of event in supplementary events \( S \). Function SELECTTRACE in the worst case will be repeated as many times as the maximum size of \( \sigma_L \) in event log \( L \). Approximately the worst case for the second approach is \( O(|L| \times |S| \times (|\sigma_L|_{\text{max}} + |\sigma_L|_{\text{max}} \times \text{alignment})) \). Fortunately, the function ISINPROCESSTRACE that uses process trace reduces the amount of computation of an alignment that need to be done in the function GETPOSSIBLETRACE. The algorithm variant in Algorithm 9 approximately the worst case for this algorithm is \( O(|L| \times (\text{alignment} + |\sigma_L|_{\text{max}} \times \text{alignment} + |\sigma_L|_{\text{max}} \times |S| \times \text{alignment} + |\sigma_L|_{\text{max}})) \).
Chapter 5

Implementation

ProM (Process Mining framework) is an extensible framework that supports a wide variety of process mining techniques [26]. ProM is an Open Source framework and provides independent platform that implemented in Java. ProM provides researchers and practitioners a tool to implement new algorithm in form of plug-ins. ProM also provides user interface and ready to use implementations (e.g. mining, alignment) and a function to read or write files. ProM supports multiple formats such as MXML, XES, Petri net, and Petri net with data. Figure 5.1 shows an overview of ProM. The Files in Figure 5.1 refer to multiple formats that supported by ProM. ProM Packages are collections one or more plug-ins. The implemented plug-ins are developed and released in the form of packages. More information about ProM and the most recent research in process mining area can be found at http://www.processmining.org/.

This chapter discusses the implementation of approaches that are discussed in Chapter 4. The approaches are implemented as plug-ins within ProM. Section 5.1 gives an overview of the implemented approaches. Section 5.2 provides an user guide how to use the plug-ins. Section 5.3 introduces additional plug-ins that are implemented in this thesis.
CHAPTER 5. IMPLEMENTATION

5.1 Overview

The approaches are implemented as plug-ins in Java within the ProM framework. The plug-ins may be re-used in the future. In order to implement the approaches, the following tools are required: Eclipse, Java, Subclipse, and ApacheIvyDE. The following packages are used as libraries in implementation: ApacheUtils, BasicUtils, EfficientStorage, Log, LogEnhancement, DataAwareReplayer, DataPetriNets, PetriNets, Widgets, XESAlignmentExtension (https://svn.win.tue.nl/trac/prom/browser/Packages).

5.2 User Guide

Currently the plug-ins are not officially released, but they are available for checkout under SVN [Packages/ChristianStevandy/Trunk] and the sources are also available at https://svn.win.tue.nl/trac/prom/browser/Packages/ChristianStevandy/Trunk. The plug-ins are called Log Extension as showed in Figure 5.2. The first approach in section 4.1 is labelled with Heuristics Based while the plug-in for the second approach in section 4.2 is labelled Alignment Based.

![Figure 5.2: Implemented Plug-ins](image)

The plug-in consists of the following steps:

1. Mapping transition name in process model to event name in log.

2. Mapping variable name in process model to variable name in event log.
3. **Additional Configuration for Alignment Based.** The alignment based has additional configuration: *Extend Single Event* option to extend only single event and *Use ID Attribute* to select an ID attribute.

4. Select final marking.

5.3 **Additional Plugins**

Additional plug-ins implemented in this thesis:

- Compare Event Logs (Relaxed) : plug-in to evaluate the extended log result.
- Compare Event Logs (Strict) : plug-in to evaluate the extended log result.
- Remove Event Plugin : plug-in to remove an event in a trace for debugging purpose.
- Select Trace by Index : plug-in to select traces in an event log for debugging purpose.
- Filter Unmapped Supplementary Events : plug-in to filter supplementary events that are not available in the model.
Chapter 6

Evaluation

This chapter discusses the results of approaches presented in chapter 4 through two case studies. The first case study is an experiment with a single type of supplementary event. The second case study, multiple types of event are missing in the main event log. The experiment is done in Windows 8.1 64 bit, i7-3770 and 16gb RAM. Section 6.1 discusses the first case study. Section 6.2 discusses the second case study. The chapter is structured as follows: section 6.1 and section 6.2 discuss the case studies, and section 6.3 summarises the evaluation.

6.1 Case 1: Road Data Fine Management

This case study uses a data from a real-life case about the process to manage road-traffic fines [27]. Section 6.1.1 describes the data that used for experiment. Section 6.1.2 discusses the evaluation method used in this case study. Section 6.1.3 and 6.1.4 discuss the evaluation result for the first and the second approach.

6.1.1 Experiment Data

The event log used in this case study contains 561,470 events and grouped in 150,370 traces out of which 76,613 traces are perfectly conformed to the process model. The event log is acquired from a single information system, hence we need to remove an event from the original event log in order to make an event log and supplementary events as input. Therefore we assumed the following scenarios:

- Removed Send Fine. The Send Fine is assumed to be recorded in different system. First Send Fine is removed from original log and it is used as a supplementary event. Then the log which Send Fine is removed becomes the main event log.

- Removed Send for Credit Collection. The same as above, Send for Credit Collection as an supplementary event. The Send for Credit Collection is an optional activity that is involved in a choice with an invisible transition.

- Removed Payment. As above, Payment as an supplementary event. The Payment is an optional activity in a self-loop pattern that can be repeated multiple times in the process.

The traces variants of the event log is shown in Figure 6.1. The Send Fine, Send for Credit Collection, and Payment are in the majority of the trace variants. About 37.56% of the traces in the event log contain Send for Credit Collection, about 37.17% (30.84+6.33)% of the traces in the event log contain Payment, and about 57.45% of the traces in the event log (37.56+13.56+6.33)%
contain *Send Fine*. The Removed Send Fine and Removed Send for Credit Collection scenarios are the same scenarios that are used in the approach in [4]. Those scenarios evaluate validity and performance of the proposed approach with regard to the approach in [4]. The Removed Send for Credit Collection and Removed Payment scenario evaluate the performance of the proposed approach to extend optional activities.

*Figure 6.1: Major Road Process Fine Management Trace Variants*

This case study uses the same process model explained in Section 2.1.1. Figure 6.2 shows the process model. The process begins by creating a fine with the variables *Amount* and *Points* record how much money need to be paid. Along the process, the offender can pay parts or full amount of the fine as seen in the transition Payment on the process model. The amount of payment is recorded in the *Payment* variable. Each time a fine is sent to the offender, an additional cost that need to be paid is recorded in the variable *Expenses*. Furthermore, if the fine is not paid timely, a penalty is added to the fine. The offender can appeal through a judge or the prefecture and the result is recorded in the variable *Dismissal*. The process ends if the fine is paid or the appeal is accepted. The process may end if the fine is paid partially, and the remaining amount that needs to be paid is recorded in variable *ToPay* at transition Send for Credit Collection. The process model has several guards as shown in 6.1 [1].

<table>
<thead>
<tr>
<th>Transition(s)</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send for Credit Collection</td>
<td><em>ToPay</em> = Amount + Expenses - Payment ∧ <em>ToPay</em> &gt; 0</td>
</tr>
<tr>
<td>Inv2, Inv3</td>
<td>Payment = Amount + Expenses</td>
</tr>
<tr>
<td>Inv1</td>
<td>(Dismissal != NIL) ∨ (Payment = Amount ∧ Points = 0)</td>
</tr>
<tr>
<td>Receive Result, Inv5</td>
<td>Dismissal = NIL</td>
</tr>
<tr>
<td>Inv4</td>
<td>Dismissal = #</td>
</tr>
<tr>
<td>Inv6</td>
<td>Dismissal = G</td>
</tr>
</tbody>
</table>

*Table 6.1: Guards of the road traffic fine management DPN-net [1]*
6.1.2 Evaluation Method

As we known the original complete log (before an event is removed as an supplementary event), the evaluation is done by binary classification. The classification are defined as follows:

True Positive (TP) : supplementary event is added and the original trace also contains the event.
False Positive (FP) : supplementary event is added but the original trace does not contain the event.
False Negative (FN) : supplementary event is not added but the original trace contains the event.
True Negative (TN) : both extended trace and original trace does not contain supplementary event.

The evaluation uses the following measurement that can be derived from binary classification:

Precision : the ratio of the number of correctly extended traces to the total number of extended traces.
Recall : the ratio of the number of correct extended traces to the total number of traces that originally contain supplementary event.
F1 Score : accuracy by measuring the harmonic mean of precision and recall.

Two types of evaluation are used, relaxed and strict evaluation. The relaxed evaluates the extended log based on control-flow perspective only where the attributes in extended event is ignored. The strict evaluates both control-flow and data perspective. In strict evaluation, any difference in a attribute in an extended event (i.e. difference in time stamp) will be categorized as False Positive.
6.1.3 Evaluation: Approach 1

Scenario: Removed Send Fine

The experiment to extend the main event log with Send Fine in the 76,613 perfectly conformed traces took 58 s and it took 2,937 s for all traces. Table 6.2 shows the results of experiment with perfectly conformed traces and Table 6.3 shows the results of experiment with all traces. In Table 6.2, in the relaxed evaluation, there are 29,249 traces that are correctly extended and 3,929 that are not extended. Zero false positive means there is no Send Fine added to a trace that originally does not contain Send Fine. However in strict evaluation where the attributes in an event are compared, there are 10,459 added Send Fine which have different attribute from the original one. Mostly it caused by the time stamp differences.

<table>
<thead>
<tr>
<th></th>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>29,249</td>
<td>18,790</td>
</tr>
<tr>
<td>FP</td>
<td>0</td>
<td>10,459</td>
</tr>
<tr>
<td>FN</td>
<td>3,929</td>
<td>3,929</td>
</tr>
<tr>
<td>TN</td>
<td>43,435</td>
<td>43,435</td>
</tr>
<tr>
<td>Precision</td>
<td>1.00</td>
<td>0.64</td>
</tr>
<tr>
<td>Recall</td>
<td>0.88</td>
<td>0.83</td>
</tr>
<tr>
<td>F1 Score</td>
<td>0.94</td>
<td>0.72</td>
</tr>
</tbody>
</table>

Table 6.2: Send Fine Extension Results (perfectly conformed traces)

<table>
<thead>
<tr>
<th></th>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>70,884</td>
<td>22,502</td>
</tr>
<tr>
<td>FP</td>
<td>48</td>
<td>48,430</td>
</tr>
<tr>
<td>FN</td>
<td>33,103</td>
<td>33,103</td>
</tr>
<tr>
<td>TN</td>
<td>46,335</td>
<td>46,335</td>
</tr>
<tr>
<td>Precision</td>
<td>0.9993</td>
<td>0.32</td>
</tr>
<tr>
<td>Recall</td>
<td>0.68</td>
<td>0.40</td>
</tr>
<tr>
<td>F1 Score</td>
<td>0.81</td>
<td>0.36</td>
</tr>
</tbody>
</table>

Table 6.3: Send Fine Extension Results (all traces)

Table 6.3 shows the results with all traces which contain traces that do not conform to model perfectly. It is clearly shown that the first approach has worse performance in a situation which traces does not conform perfectly and mostly it is caused by false negative. The approach fails to add supplementary event because the extended trace will always violate conformance checking in Algorithm 1 as it requires a perfect conformance.

Scenario: Removed Send for Credit Collection

The experiment to extend the main event log with Send for Credit Collection in the 76,613 perfectly conformed traces took 137 s and it took 2,720 s for all traces. This scenario has better strict evaluation than the removed Send Fine scenario because of the following possible explanations. First, the values of the attribute ToPay in Send for Credit Collection has more variations than the values of the attribute Expenses in Send Fine. Logically, the expenses to send a letter is less varied than the number combination of people credits because the expenses has standard rates (i.e. 1 EUR stamp), hence, the guards are less strict on Send Fine candidates. As a consequence there are more Send Fine as a candidate than Send for Credit Collection that causes more false positives. There are more trace variants contain Send Fine than Send for Credit Collection as it is shown in Figure 6.1. Figure 6.1 shows that 37.56% of log contain Send for Credit Collection (red) while 57.45%

Similarly, in experiment with all traces in Table 6.5, the majority traces that contain Send for Credit Collection that conform perfectly to the model. Figure 6.1 shows at least 37.56% traces that contain Send for Credit Collection conform to the model. The third trace variant in Figure 6.1 (Create Fine, Send Fine) is an example of trace that does not conform to the model. Most of the traces that contain Send for Credit Collection conform to the model perfectly. As a consequent, this scenario has better results than the scenario removed Send Fine.
CHAPTER 6. EVALUATION

### Scenario: Removed Payment

The experiments to extend the main event log with Payment in the 76,613 perfectly conformed traces took 177 s and it took 2,792 s for all traces. Payment is an optional activity that may be executed multiple times. Although this scenario contains some traces with multiple Payment events that are not suitable for the first approach, the majority of trace variants only contain single Payment as it is shown in Figure 6.1, for this reason, the first approach does well in perfectly conformed trace scenario as shown in Table 6.6. However, in the experiment with all traces in Table 6.7, it shows worse results. The possible causes of worse performance are because there are many traces that do not conform to process model, and there are some multiple Payment events in traces that considered as a violation in the conformance checking in Algorithm 1.

<table>
<thead>
<tr>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP 27,092</td>
<td>24,784</td>
</tr>
<tr>
<td>FP 0</td>
<td>2,308</td>
</tr>
<tr>
<td>FN 957</td>
<td>957</td>
</tr>
<tr>
<td>TN 48,564</td>
<td>48,564</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>0.97</td>
</tr>
<tr>
<td><strong>F1 Score</strong></td>
<td>0.98</td>
</tr>
</tbody>
</table>

Table 6.4: Send for Credit Collection extension results(perfectly conformed traces)

<table>
<thead>
<tr>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP 57,656</td>
<td>56,164</td>
</tr>
<tr>
<td>FP 1,071</td>
<td>2,563</td>
</tr>
<tr>
<td>FN 1,357</td>
<td>1,357</td>
</tr>
<tr>
<td>TN 90,286</td>
<td>90,286</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>0.98</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>0.98</td>
</tr>
<tr>
<td><strong>F1 Score</strong></td>
<td>0.98</td>
</tr>
</tbody>
</table>

Table 6.5: Send for Credit Collection extension results(all traces)

<table>
<thead>
<tr>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP 57,581</td>
<td>55,746</td>
</tr>
<tr>
<td>FP 1,170</td>
<td>3,005</td>
</tr>
<tr>
<td>FN 1,432</td>
<td>1,432</td>
</tr>
<tr>
<td>TN 90,187</td>
<td>90,187</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>0.98</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>0.98</td>
</tr>
<tr>
<td><strong>F1 Score</strong></td>
<td>0.98</td>
</tr>
</tbody>
</table>

Table 6.6: Payment extension results (perfectly conformed traces)

<table>
<thead>
<tr>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP 53,649</td>
<td>19,678</td>
</tr>
<tr>
<td>FP 8,755</td>
<td>42,726</td>
</tr>
<tr>
<td>FN 16,066</td>
<td>16,066</td>
</tr>
<tr>
<td>TN 71,900</td>
<td>71,900</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>0.86</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>0.77</td>
</tr>
<tr>
<td><strong>F1 Score</strong></td>
<td>0.81</td>
</tr>
</tbody>
</table>

Table 6.7: Payment extension results (all traces)

### 6.1.4 Evaluation: Approach 2

The second approach does not require a scenario with a perfect conformance. Therefore, this evaluation does not distinguish between perfect and imperfect traces. The experiment is done with Algorithm 9 for an extension with a single type of event and Algorithm 8 for an extension with multiple types of event. The second approach is computational expensive, hence the experiment is done only in 1,000 traces that arbitrarily selected out of the 150,370 traces. For comparison, the experiment is also done with the first approach.

**Scenario: Removed Send Fine**

This experiment took 3 s for the first approach and 1,112 s for the second approach. Table 6.8 and 6.9 show the results of experiment. Table 6.8 shows the second approach generates a lot of False Positive and a small number of True Negative. This is caused by the third heuristic where
an event is added if the extended trace has lower or equal cost than before. Although it is not perfect, the second approach still give a reasonable results.

<table>
<thead>
<tr>
<th></th>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>646</td>
<td>581</td>
</tr>
<tr>
<td>FP</td>
<td>347</td>
<td>412</td>
</tr>
<tr>
<td>FN</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>TN</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Precision</td>
<td>0.65</td>
<td>0.59</td>
</tr>
<tr>
<td>Recall</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>F1 Score</td>
<td>0.79</td>
<td>0.74</td>
</tr>
</tbody>
</table>

Table 6.8: Send Fine Extension Results (Approach 2)

<table>
<thead>
<tr>
<th></th>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>403</td>
<td>257</td>
</tr>
<tr>
<td>FP</td>
<td>0</td>
<td>146</td>
</tr>
<tr>
<td>FN</td>
<td>249</td>
<td>249</td>
</tr>
<tr>
<td>TN</td>
<td>348</td>
<td>348</td>
</tr>
<tr>
<td>Precision</td>
<td>1</td>
<td>0.64</td>
</tr>
<tr>
<td>Recall</td>
<td>0.62</td>
<td>0.51</td>
</tr>
<tr>
<td>F1 Score</td>
<td>0.76</td>
<td>0.57</td>
</tr>
</tbody>
</table>

Table 6.9: Send Fine Extension Results (Approach 1)

Scenario: Removed Send for Credit Collection

This experiment took 2 s for the first approach and 492 s for the second approach. Table 6.10 and 6.11 show the results of the experiment. Table 6.11 shows that the first approach is better than the second approach in an optional activity scenario. As already explained in Section 6.1.3, most of the traces that contain Send for Credit Collection conform to process model; hence, the first approach gives good results for this scenario.

<table>
<thead>
<tr>
<th></th>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>303</td>
<td>301</td>
</tr>
<tr>
<td>FP</td>
<td>462</td>
<td>465</td>
</tr>
<tr>
<td>FN</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>TN</td>
<td>221</td>
<td>221</td>
</tr>
<tr>
<td>Precision</td>
<td>0.40</td>
<td>0.39</td>
</tr>
<tr>
<td>Recall</td>
<td>0.96</td>
<td>0.96</td>
</tr>
<tr>
<td>F1 Score</td>
<td>0.56</td>
<td>0.56</td>
</tr>
</tbody>
</table>

Table 6.10: Send for Credit Collection Extension Results (Approach 2)

<table>
<thead>
<tr>
<th></th>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>312</td>
<td>312</td>
</tr>
<tr>
<td>FP</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>FN</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>TN</td>
<td>678</td>
<td>678</td>
</tr>
<tr>
<td>Precision</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>Recall</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>F1 Score</td>
<td>0.98</td>
<td>0.98</td>
</tr>
</tbody>
</table>

Table 6.11: Send for Credit Collection Extension Results (Approach 1)

Scenario: Removed Payment Single

This experiment took 3 s for the first approach and 1,449 s for the second approach. Table 6.10 and 6.11 show the results of the experiment. Table 6.11 shows that the first approach is better than the second approach in an optional activity scenario like Payment.
CHAPTER 6. EVALUATION

<table>
<thead>
<tr>
<th></th>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>296</td>
<td>29</td>
</tr>
<tr>
<td>FP</td>
<td>512</td>
<td>779</td>
</tr>
<tr>
<td>FN</td>
<td>173</td>
<td>173</td>
</tr>
<tr>
<td>TN</td>
<td>19</td>
<td>19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Precision</th>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.37</td>
<td>0.04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recall</th>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.32</td>
<td>0.05</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F1 Score</th>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.46</td>
<td>0.06</td>
</tr>
</tbody>
</table>

Table 6.12: Single Payment Extension Results (Approach 2)

<table>
<thead>
<tr>
<th></th>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>414</td>
<td>135</td>
</tr>
<tr>
<td>FP</td>
<td>53</td>
<td>332</td>
</tr>
<tr>
<td>FN</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>TN</td>
<td>478</td>
<td>478</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Precision</th>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.89</td>
<td>0.29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recall</th>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.89</td>
<td>0.61</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F1 Score</th>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.88</td>
<td>0.44</td>
</tr>
</tbody>
</table>

Table 6.13: Single Payment Extension Results (Approach 1)

In the second approach, adding an optional activity to a trace may result the extended trace to have the same cost as the trace before. This may happen if there is no guard in the transition that correspond to an optional activity or there is no strict guard in the model as illustrated in Figure 6.3. The transition B in the model is a transition that correspond to an optional activity. The first trace [A,B] has cost 0, after an extension with B, it still has the same cost as before.

![Figure 6.3: extended trace has the same cost as before](image)

Section 6.1.4 shows that the second approach can extend an optional activity accurately because the transition Send for Credit Collection that correspond to an optional activity has a guard. In a case like this, extending a trace with a right optional activity will always lower the extended trace cost.
Scenario: Removed Payment Multiple

This experiment took 107 s for the second approach and the first approach unable to extend a trace with multiple types of event. Table 6.14 shows the results of the second approach. Figure 6.4 shows the extended trace with the second approach.

<table>
<thead>
<tr>
<th></th>
<th>Relaxed</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td>FP</td>
<td>349</td>
<td>408</td>
</tr>
<tr>
<td>FN</td>
<td>409</td>
<td>409</td>
</tr>
<tr>
<td>TN</td>
<td>182</td>
<td>182</td>
</tr>
<tr>
<td>Precision</td>
<td>0.15</td>
<td>0</td>
</tr>
<tr>
<td>Recall</td>
<td>0.13</td>
<td>0</td>
</tr>
<tr>
<td>F1 Score</td>
<td>0.14</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6.14: Multiple Payment Extension Results

Figure 6.4 shows that the event Payment is added multiple times in some traces. This is caused by three reasons. First, adding an optional activity may give the same cost as before the addition as already mentioned in Section 6.1.4, accordingly, the Payment is always added as long as the cost after an extension is equal or lower than the cost before an extension. Second, the guards in the transition Send for Credit Collection, Inv1, Inv2, and Inv3 are examples of a combinatorial problem. The alignment only measures correct or incorrect write operations which is a boolean, consequently the second approach cannot find the right combination in order to comply the guards. For example, if we have a guard (TotalPayment = 3) with the variable TotalPayment as a total of a variable Payment and events [(Payment{Payment=1}), (Payment{Payment=2}), (Payment{Payment=3})]. The possible solution for this example is a combination of events with a TotalPayment of 3 such as (Payment{Payment=3}) or [(Payment{Payment=1}), (Payment{Payment=2})], this possible combinations cannot be solved with the second approach. Third, because of the first and second reason, the algorithm in the second approach does not know when to stop adding an optional event unless the extended cost is zero (perfect conformance). For example if we have the guard (TotalPayment = 3), a combination of events that gives TotalPayment = 4, and it is assumed that the current alignment cost is 5. The guard is false because the Payment is 4. Therefore, adding another optional event will not change the alignment cost, because the guard will always false. As a consequence all the remain events will be added because the extended trace will have the same cost as before. Even if the guard is true, for example
(TotalPayment > 3) with the current alignment cost is 4, the events will be keep added because the guard will always remain true that makes the cost remains the same.

6.2 Case 2: WW-uitkering UWV

This case study uses a data from a real-life case about the process to claim unemployment benefits in UWV. Section 6.2.1 describes the data that used for experiment. Section 6.2.2 discusses the evaluation method used in this case study. Section 6.2.3 and 6.2.4 discuss the evaluation result for the first and the second approach.

6.2.1 Experiment Data

The data for the second case study consists of 2 anonymized logs that are acquired from an information system with overlapping cases. The first log contains 1,000 traces which with 2,813 events of claim unemployment benefits process, we call it as the main event log and the second log contains a trace of 22,087 unmapped events which are the supplementary events. The main event log contains 3 variants as shown in Figure 6.5 and the supplementary events contain multiple events as shown in Figure 6.6. There exists a customer ID in the log, however every customer ID may involve in multiple cases. Therefore, the separate logs cannot be trivially matched with the customer ID. The customer ID helps to reduce computational times in the second approach.

![Figure 6.5: Main Event Log](image1)

![Figure 6.6: Supplementary Events](image2)

Figure 6.7 shows the process model used in this scenario. The process starts when a customer makes a claim for unemployment benefits. After receiving some forms and letters, the customer receives a decision whether the claim is accepted or rejected. The process ends if the claim is rejected or the claim is withdrawn. If the claim is accepted, a payment and some letters or forms may be sent regularly. While receiving benefits, the benefits can end in three ways: the customer...
has received maximum benefits, the customer is working, or other reasons. Finally, after the benefits end, the customer received the last payment and the process ends. The guards are listed in Table 6.15.

Figure 6.7: DPN-net of Unemployment Benefits Claim Process
<table>
<thead>
<tr>
<th>Transition(s)</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Income Form IN</td>
<td>((IKF_Kanaal_IN' == IKF_Kanaal_UIT) &amp;&amp; (IKF_PeriodeID_IN' == IKF_PeriodeID_OUT))</td>
</tr>
<tr>
<td>Income Form OUT</td>
<td>(IKF_PeriodeID_OUT' &gt; IKF_PeriodeID_IN)</td>
</tr>
<tr>
<td>Letter Benefits End Max</td>
<td>(WW_RedenEindeCat == &quot;Max Duration&quot;)</td>
</tr>
<tr>
<td>Letter Benefits End Other</td>
<td>(WW_RedenEindeCat == &quot;Other Reason&quot;)</td>
</tr>
<tr>
<td>Letter Benefits End Work</td>
<td>(WW_RedenEindeCat == &quot;Work&quot;)</td>
</tr>
<tr>
<td>Letter Claim Accepted</td>
<td>(WW_UitkomstBeslissing == &quot;Toekenning&quot;)</td>
</tr>
<tr>
<td>Letter Claim Rejected</td>
<td>(WW_UitkomstBeslissing == &quot;Afwijzing&quot;)</td>
</tr>
<tr>
<td>Letter Withdrawal Claim</td>
<td>(WW_UitkomstBeslissing == &quot;Intrekken&quot;)</td>
</tr>
</tbody>
</table>

Table 6.15: Guards of the unemployment benefits claim process DPN-net

6.2.2 Evaluation Method

As the original complete event log that is currently used in experiment does not exist, the evaluation for the second scenario cannot be done in the same way as the first case study. The evaluation for the second case study is done by few methods and through consulting a domain expert from UWV. The following analysis are used as evaluation:

- Conformance checking: the alignment before and after the extension are compared.
- Log comparison: the statistics of the extended log and another event log from the same process are compared. The statistics used are the average number of events per trace, the minimum and maximum number of events in a trace, and the average fitness.
- Individual case analysis: this method evaluates the approach with a few selected extended traces that involved an optional activity. The selected traces are a common case that usually happen in the process. The extended traces are consulted with a domain expert. This method evaluates the reliability of the approach to extend an event log in a general situation according to a domain expert.

6.2.3 Evaluation: Approach 1

The extended log from the first approach is exactly the same as the main event log where no extension is happened. The first approach cannot extend an event log that does not conform to process model perfectly and it cannot extend a trace with multiple types of event. The first approach is limited to one single type of event.

6.2.4 Evaluation: Approach 2

First, the events in the supplementary events that are not available as a transition in the model are filtered. The filtered supplementary events consist of 18,060 events. The experiment took 362 s for extending 1,000 traces with the second approach. The main event log is successfully extended from total of 2,813 to 9,312 events. 11,561 out of 18,060 supplementary events are not added.

Conformance Analysis

By using the conformance checking in [1] with a default configuration, the average fitness before the extension is 45.6% and the average fitness after the extension is 64.5%. Figure 6.8 shows the
average fitness of trace variants of the original main log. The first trace variant on the top of Figure 6.8 (812 traces) has the average fitness of 44.44%. The size of the trace variant with the average fitness of 44.44% is reduced from 812 to 34 in the extended log. The highest and lowest average fitness for the original main log are 55.56% and 20.00%. After the extension, the highest and the lowest average fitness are 100% and 22.22%. It shows the second approach is able to combine the main event log with supplementary event to some extent. The uncorrelated supplementary events are mostly the following events: Income IN, Income OUT, Change Form IN, Change Form OUT. Those events have a similar pattern, they are in an optional composite activities or optional sub process. The pattern is shown in Figure 6.9. The second approach cannot correlate supplementary event in this pattern because an event is added one by one. As for this reason, adding an event that belong to a sub process will increase cost because all events beside the one that is added in the sub process require to be executed in the model. This gives additional moves in log and increases cost.

**Comparison with Another Complete Event Log**

A complete event log from the same process is used for a comparison. Although it has the same process, there are some events in the log that are not included in the process model. The event log contains 1,000 traces with 28,182 events in total. On average each trace contains 28 events. The minimum and maximum events in a trace in the event log are 3 and 182. The average fitness according to the conformance checking in [1] with a default configuration is 79.4%. The extended log from the second approach contains 1,000 traces with 9,312 events in total. The average number of events per trace is 9 and the minimum and maximum events in a trace in the extended event log are 2 and 55. The average fitness of the extended log is 64.5%.

The statistics between the complete event log and the extended event log show differences. The differences are caused by the following reasons. First, not all events in the complete event log are available in the process model; hence, the number of minimum and maximum events in a trace are difference. Second, as confirmed by the domain expert, the supplementary events may miss some events in a trace. Third, the approach is unable to correlate some events in an optional sub process. Although this cannot extend an event log with the similar statistics as the complete event log, it still shows a reasonable results.
CHAPTER 6. EVALUATION

Figure 6.9: Optional Subprocess

Case Analysis

This evaluation investigates the reliability of the extended result according to a domain expert through some selected individual cases. The following individual cases are analysed and verified by domain experts:

1 Sequential Case
Customer with the same Customer ID may have multiple claim processes as shown in Figure 6.10. A process is represented as a trace in the figure. The first process happens on 04.03.14 and ends on 27.05.14 and the second process happens on 25.05.14 and ends on 04.09.14. This is a case where customer is rejected at first, but successfully received benefits in the next claim. This case proves the second approach successfully add supplementary events with a correct time order.

Figure 6.10: Extended Log Sequential Case

2 Parallel Case
Customer may have few parallel processes at the same time as shown in Figure 6.11. As mentioned in Section 6.1.4, the second approach has a difficulty to find a stop condition for adding an optional activity. The second approach adds all Payment that happens between 21.12.13 and 26.03.14 to the first process. The second process only has the leftover Payment that happened after 26.03.14. If the second process is extended first, the first process will not be extended with any Payment because all Payment happen between 21.12.13 and 28.07.14 will be extended to the second process. This case shows the second approach successfully added
optional activity *Payment* with its limitation.

![Figure 6.11: Extended Log Parallel Case](image)

### 3 Special Case

Normally, if a claim is accepted, the *Decision End Benefits* is always executed before the process ends. However, in one case, the extended log of one customer has two accepted claim traces where one of the trace does not end. At first, it seems *Decision End Benefits* is failed to be added, however after checking the supplementary events, there is only one *Decision Benefits End* available for that customer. Then, after consulting with a domain expert, it is found that this is special case where it caused by administration problem. The customer is a civil servant where as a civil servant, he needs to claim benefits in specific locations. At first, he came to an arbitrary location to claim benefits, then he made another claim in suggested location that made him has two traces where one of it is a faulty trace. This case shows the extended log may give an insight to a domain expert.

### 6.3 Summary

In this chapter the approaches are evaluated with two study cases. In the first case study, some selected events have been removed for evaluation. In the second case study, the data already contains the main event log and supplementary events. Both approaches able to extend an event log with supplementary events that correlate to an optional activity. In evaluation, some limitations are discovered. Both approaches some a better result if a transition that correspond to an optional activity contains a guard. The first approach works only for a single type of event and in a scenario with a perfect conformance, and hence the first approach cannot extend an event log in the second case study. The second approach is computationally expensive in comparison with the first approach. The second approach cannot extend a supplementary events that belong to an optional sub process. Moreover it also cannot solve combinatorial problem.
Chapter 7

Conclusions

In this chapter the research questions in Chapter 1 are answered and discussed in Section 7.1. Section 7.2 describes the reflection from the author of this thesis. Finally, in Section 7.3 limitations and future works are discussed.

7.1 Conclusions

This thesis proposes two approaches to combine an event log with supplementary events that capable of correlating an optional activity. The approaches are available as plug-ins in ProM. This thesis answers the following research sub questions:

1 Why is the approach in [4] unable to correlate supplementary events that correspond to an optional activity?
   The current approach in [4] cannot correlate any optional activity because in the alignment, a trace with an optional activity has a bigger cost than a trace without an optional activity, hence the approach ignored optional activities (Section 3.1).

2 How do we correlate an optional activity?
   Chapter 4 proposes heuristics and algorithm that capable to correlate an optional activity. An optional activity is correlated by using the greedy algorithm and verified by conformance checking.

7.2 Reflection

This thesis uses the regulative cycle [9]. I choose this method, because it is a classic problem solving method and it is used in practice. Moreover, the regulative cycle is similar to the software development method (e.g., evolutionary, agile) and the five steps of it are suitable to be used as chapters in writing (Chapter 1, 3, 4, 5, and 6). This thesis is written in \LaTeX. It makes writing mathematical notation, algorithm, and referencing easier. However, it does not have any grammar checker, even after I did some proofreading, this thesis probably still contains a lot of grammar mistakes. In addition, editing a table is difficult in \LaTeX.

The main obstacle in this thesis is the implementation. Firstly, the ProM framework is new for me, so it took a while to get use of it. Moreover the documentation how to develop a plug-in in ProM is already out of date. Secondly, not all sources are documented, sometimes it needs time to understand a source. Lastly, the sources I uses are still in development, so a change is always
there and some required me to update my source code, reran evaluation, and revised the result of my evaluation.

Instead of proposing two approaches, I think it is better to focus just to one approach. Therefore, more deep understanding can be found. For example, an optional activity can be accurately added, if a transition that related to optional activity contain a guard. Maybe there are more conditions that make adding an event more precise.

Overall, I am happy with this project. The project gives me some new knowledge, the problem is also challenging, and my supervisors are really supportive.

7.3 Limitations and Future Works

Both approaches cannot extend an event log with supplementary events perfectly. The first approach is limited for a single type of event and it only works for a scenario with a perfect conformance. In reality, it is difficult to a process that comply to a process model perfectly, and hence the usage of the first approach is limited in a real world scenario. The second approach limitations are as follow. Firstly, the second approach is computationally expensive, consequently it has a limited scalability; hence, the second approach requires optimisation. Secondly, the second approach cannot extend an event that belong to an optional sub process and combinatorial problem as discussed in Section 6.2. Finally, it has a difficulty to find a stop condition for adding an optional activity.
Bibliography


Combining Event Logs with Supplementary Events 55


