Technical University of Denmark

Department of Applied Mathematics and Computer Science

Thesis Project

Exploring Process Mining in IoT Environments

Author:
Amine Abbad Andaloussi

Supervisors:
Barbara Weber
Andrea Burattin

December, 2017

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Contents

Preface

This work summarizes the thesis project entitled "Exploring Process Mining in IoT Environments", written by the author as part of his Msc. Program in Computer Science and Engineering at the Applied Mathematics and Computer Science Department of the Technical University of Denmark. This report investigates the application of process mining in Internet of Things (IoT) environments.

In IoT environments a network of inter-connected devices equipped with sensors operating autonomously is responsible for collecting and sharing data over the internet. The Business Process Management (BPM) field offers a set of disciples allowing to design, implement, execute, and analyze business processes. There is an increasing interest in Bridging the gap between the IoT world and BPM world to improve the execution of business processes. In the past decade process mining has been successfully used in several BPM contexts; however, its application in IoT environments is still not entirely understood.

Process mining can be applied as a bottom-up approach of looking at BPM-IoT relation. By transforming sensor data to log events, it is possible to obtain event logs; thus enabling all process mining capabilities. This work proposes a new approach aiming at mapping sensor data to business process activities to generate comprehensive event logs.

To achieve this goal, a new process mining scope where business processes are supported with IoT devices is introduced. Within this scope, a use case scenario illustrating the work-flow in an IoT environment is presented, and a business process simulator allowing to imitate the interactions between sensors and business processes is designed and implemented.

Besides, this work investigates also a common challenge within the process mining community that is to infer the event log scheme. While designing the sensor data mapping approach this issue was faced and solved. Consequently, a new state-of-art approach aiming at automatically inferring log scheme from event logs is proposed as part of this thesis project.

December, 2017 — Amine Abbad Andaloussi
Copenhagen, Denmark

Acknowledgements

I owe gratitude to the people who helped me to achieve this work, and without their encouragements, this project would not have seen the light. I would like to express my high gratitude to my supervisors Barbara Weber and Andrea Burattin who were always present to provide me with support and guidance throughout this project. I have been extremely lucky to have you as supervisors. Thank you for caring about my work, spending your precious time reviewing it, and providing me with constructive feedback. I must express my great acknowledgement to you for believing in me, and for giving me this opportunity to put my knowledge to evidence.

To my sweet family, thank you for your support and endless love. To my caring mother, thank you for your unconditional love and your permanent patience for the ups and the downs of my mood during this period. To my father, thank you for your greatest gift that is to believe in me and to encourage me always to be the best. To my sweet sister, thank you for your advice and continuous support which is determinant to my success.

I'm very thankful to my work colleagues with whom I have shared some great moments of pleasure playing at Table Soccer. This game helped us to foster our team building spirit and strengthen our personal and professional relations. I'm very proud to spend my days in such friendly working atmosphere with well-experienced people, and I'm truly thankful for your continuous help and suggestions whenever I needed it.

Last but not least, I would like to thank all my friends who contributed from close or from far to the realization of this work. Thank you for being always here for me. Without your encouragements, I would have never come this far.

December, 2017 Amine Abbad Andaloussi
Copenhagen, Denmark

Chapter 1

Introduction

This chapter is an introduction to this thesis report. Section 1.1 briefly emphasis on the relationship between Process Mining and IoT, and highlights the mutual benefits, and challenges raised from bridging the gap between the two fields. Section 1.2 introduces the context of thesis report. Section 1.3 defines the research scope and challenges considered by this report. Finally Section 1.4 introduces a use case scenario that will be used throughout the report as running example to illustrate the proposed approaches.

1.1 Process Mining and IoT: a Bottom-Up Approach

During the past two centuries, our world has experienced a huge industrial revolution. Starting from the Power Steam at the 18th century (Industry 1.0), the Mass Production, Assembly Line, and Electricity at the beginning of the 20th century (Industry 2.0), Automation and Computers at the end of the 20th century (Industry 3.0) and recently Cyber-Physical Systems including Internet of Things (IoT), Big Data, and Cloud Computing (Industry 4.0). Nowadays, our world becomes massively interconnected using sensors and actuators embedded in smart devices connected to the internet, which are responsible for monitoring their external environment, and enacting when it is needed. The massive amount of data generated from IoT devices created a mutual benefit relation between business process management (BPM) field and IoT domain [23].

The BPM field aims at enabling process automation, and process analysis to improve the way business processes are managed within an organisation [42, p. 3]. Exploring IoT data in process analysis allows BPM to provide a comprehensive overview of the business process execution [23]. For instance, the sensor data coming from IoT devices can help to track the progress of manual activities in a business process, thus providing fine-grained insights. Furthermore, IoT data helps to reduce the human interactions with business processes which implies less human-related errors and more time-saving. Bridging the gap between the two fields is not only advantageous to BPM but also beneficial to IoT. Indeed, BPM offers a robust process management platform to IoT networks, which allows monitoring, managing, and optimising the interactions between IoT devices from a process-oriented perspective [23].

So far the relationship between BPM and IoT is described from a top-down view (from business processes to IoT data), where the business process management system (BPMS) is seen as a central

orchestrator controlling the process execution with the support of IoT data. However, there exists another perspective to look at the relationship between BPM and IoT. In fact, instead of considering the top-down view, it is possible to use a bottom-up view (from IoT data to business processes). This perspective consists at exploring, aggregating and learning from the raw data generated by IoT devices, in order to infer high-level business process activities that can be used by process mining algorithms to support the decision making process [42].

The bottom-up approach relies mainly on the ability of process mining techniques to deal with sensor data. As process mining algorithms require event logs, a possible way to bridge the gap between process mining and sensor data is to aggregate sensor data to infer log events, thus enabling all process mining capabilities. The BPM IoT manifesto [23, p. 6] Published recently, highlights several challenges raised by this approach. Among them, the mapping of sensor data to process activities, knowing that a sensor data entry can be relevant to one process activity as it can be relevant to several process activities (one-one relation, one-many relation). Another critical challenge is to discover the corresponding process instance for such process activities, knowing that several process instances might run concurrently. The literature shows that authors have tried to tackles these challenges from different perspectives using several techniques such as interaction mining [36] and human habits and behaviour analysis [19] [7] [16] [17]. The literature review in Chapter 5 provides a comprehensive overview of these techniques.

1.2 Thesis Context

This thesis fits into the intersection of process mining with the IoT field. Precisely, it establishes a strong tie between the BPM world and the IoT world using process mining techniques as a bottom-up approach to discover the work-flow of business processes in IoT environments. Since the entry level for all process mining techniques is an event log [42], this work aims at setting up a framework that allows generating comprehensive event logs from the IoT environment data sources.

The process of generating comprehensive event logs relies extensively on merging and mapping sensor data with event logs, for this purpose several approaches are proposed. To ensure a good understandability of these approaches, a set of preliminary concepts and techniques are introduced in Chapter 2. Namely, a new process mining scope where IoT devices play a central role in the process mining game is introduced. In addition several fundamental process mining notions such as the event log components and the control-flow discovery quality dimensions are recapitulated. Furthermore, a classification of sensor data according to their mapping characteristics is introduced.

The availability of experimental data is crucial to try out the proposed approaches, although, some event logs and sensor data log are publicly available, it is still challenging to obtain real-world log files recording the interactions between IoT devices and information system in IoT environment. As alternative, synthetic logs can serve to evaluate the proposed approaches, however, as it will be presented in the literature review of Chapter 3 the existing business process simulators do not provide enough simulation mechanisms to imitate interactions between sensors and business processes, thus, designing and implementing a business process simulator able to cope with sensor data in an BPM setting is also one of the subjects investigated.

This work is suitable for IoT environments containing one or many information systems supported by a network of IoT devices. To illustrate such IoT environments, a use case scenario inspired from real-world settings is presented in Section 1.4. The aim of the mapping and merging approaches proposed for this kind of IoT environments is to analysis the correlations between log events and

between sensor data entries coming from different data sources to obtain an accurate mapping. To do so, it is necessary to group events belonging to the same process execution into cases without any domain-knowledge. Chapter 4 is allocated to investigate the way the case identifier attribute can be inferred automatically from an event log. Consequently, a new state-of-art approach consisting at using control-flow discovery quality dimensions to infer case identifiers is introduced, the approach shows promising results on both synthetic logs and real-world event logs.

Chapter 5 introduces a merging and mapping approach aiming at merging event logs, and mapping sensor data to events. The approach treats each sensor type defined in Chapter 2 independently. The work presented in this chapter is inspired by recent publications in this area. As result, a comprehensive event log is generated from the sensor logs and the event logs recording the workflow of the use case scenario. The obtained log is used to evaluate the effectiveness of the proposed approach by comparing the discovered process model with the original use model. Figure 1.1 summarizes this section and depicts the scope of this thesis project.

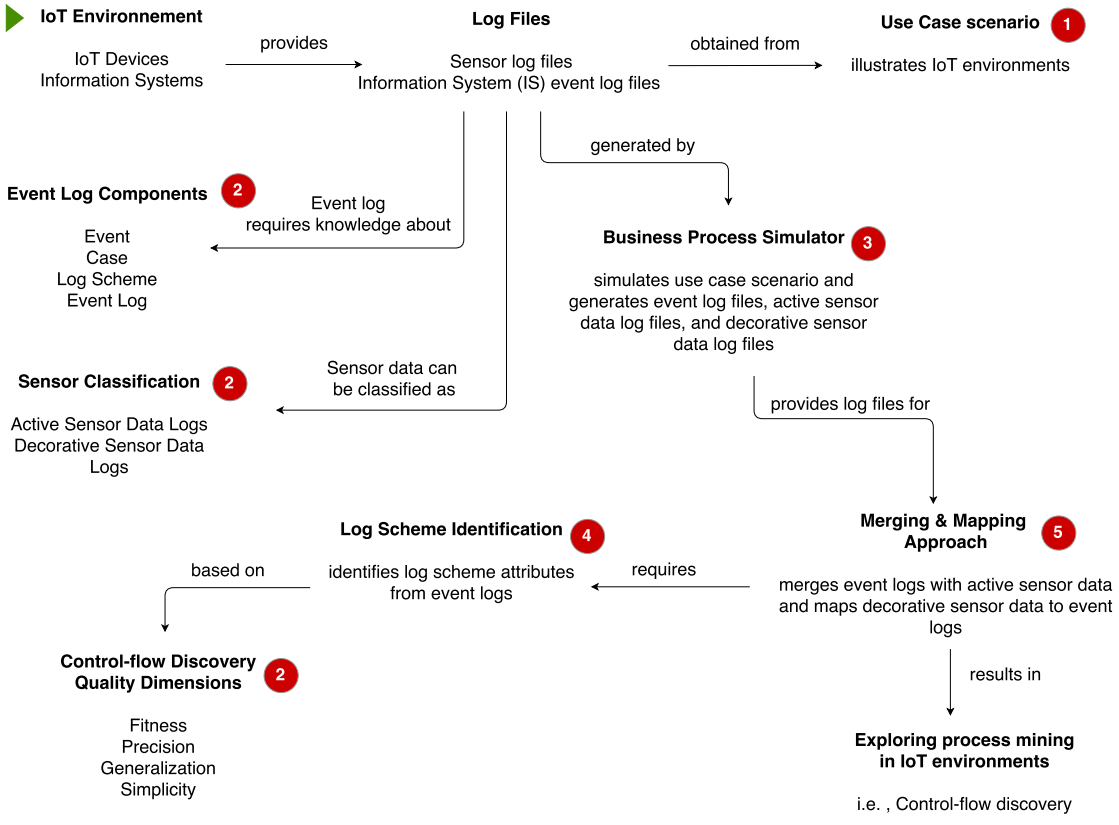


Figure 1.1: Scope of this thesis project. The green triangle refers to the starting point of the chart, and the red circles show the chapter numbers discussing the corresponding topics.

1.3 Research Problem

The thesis context described in Section 1.2 highlights the research aspects considered to enable process mining in IoT environments. Therefore, it becomes clear that the main research problem is the following:

> In an IoT environment, given a set of sensor data logs generated by IoT devices and a set of event logs recorded by information systems, the primary aim is to map sensor data with log events in order to construct a comprehensive event log file providing detailed insights about the execution of the business process in question, which also implies enabling process discovery as a process mining feature.

This work investigates the research problem in a divide and conquer manner. As described in Section 1.2 the main research problem is divided into smaller sub-problems that are solved independently with the support of existing work presented in the literature review of each chapter.

1.4 Use Case

Designing a use case scenario to illustrate an IoT environment comes as a primary need to realize this work. As mentioned in Section 1.2, obtaining real-world data sets showing the interactions between IoT devices and information systems in an IoT environment is very challenging. Nevertheless, developing a strategic use case scenario to investigate this subject is also complex. As source of inspiration, the smart factory example referred in the BPM-IoT manifesto [23, p. 2] provides a great perception of how IoT can benefit from BPM.

Kiva robots represent a good example of IoT environments deployed in smart factories. They are part of the Kiva warehouse management system [50] that is a pick-pack-and-ship solution aiming at providing mobile storage shelves that can be moved by robots. The Kiva robots prevent warehouse employees from walking to pick up items, instead, a set of autonomous robots are responsible for picking up shelves of inventory and carrying them to employees. Amazon [1] is among the companies deploying Kiva robots in their large warehouses. According to Business Insider [2] thanks to Kiva robots, Amazon has cut up their operating expensive by 20%, increased their efficiency, reduced shipments cycle times, and grown up the size of their inventories considerably.

The use case scenario presented in this work is inspired by the Kiva robots deployed at Amazon warehouses. For the sake of simplicity, a prototype process of these robots is modelled using BPM notation (BPMN [3]) to emphasis only on the parts relevant to the context of this thesis. As the main interest is about exploring the interactions between sensors and business processes, the use case scenario contains a set of RFID sensors and Accelerometer sensors responsible for coordinating the work-flow between three processes. The sensor types are chosen for a strategic purpose. Indeed, the RFID sensors symbolize sensors that provide discrete data stream while Accelerometer sensors symbolize sensors that provide continuous data stream. For instance, an Accelerometer sensor is meant to deliver real-time acceleration coordinates of an object; thus it is providing a continuous data stream, while an RFID sensor is only intended to react once a matching RFID tag is detected, therefore the data stream provided in this case is discrete.

[1] See `https://www.amazonrobotics.com/`
[2] See `http://www.businessinsider.com/kiva-robots-save-money-for-amazon-2016-6?r=US&IR=T&IR=T`
[3] See `http://www.bpmn.org`

As it will be presented in Chapter 3, the use case scenario introduced in this section is simulated to generate a set of log files. In this context, it is assumed that a warehouse managed by an information system is equipped with a set of robots and smart shelves. Under the assumption that some shelves contain fragile products, an Accelerometer is plugged on each shelf to detect shakes that may damage the products on the shelf. Moreover, a collection of RFID sensors and RFID tags are assumed to be deployed all over the warehouse to ensure the communication between the different processes (i.e., sending and receiving requests, and notifications), and to track the progress of manual activities (i.e., activities done by hand). This variety of components records all their operations in log files which are generated using a simulation program developed for this purpose.

Figure 1.2 depicts the use case scenario. The *Clerk process* starts when a clerk receives an order. The clerk requests the product from an information system, which identifies the product's shelf and automatically assigns a robot to pick it up. Afterwards, a product request is sent to the robot. Once the request is received, the *Robot process* checks if the robot is already under the requested shelf, if not, the robot put-down its current shelf, and go to the appropriate shelf. Once the shelf is reached, the robot notifies the *Smart Shelf process* to start recording the Accelerometer data in order to detect shakes that may happen in case the shelf is shaken while being moved by the robot. Afterwards, the robot starts immediately moving the shelf to the clerk dock. In case a shake is detected while the shelf is being moved, the *Smart Shelf process* notifies the *Robot process* which reduces the speed of the robot to prevent the product from being damaged. Also, the *Smart Shelf process* updates a product artifact with all the products on the shelf that were shaken. Once the shelf is delivered to the clerk dock, the clerk collects the product from the shelf before the robot can dispose it. The clerk uses the product artifact to check if the product was shaken, if true an extra check is performed. Finally, the product is packaged.

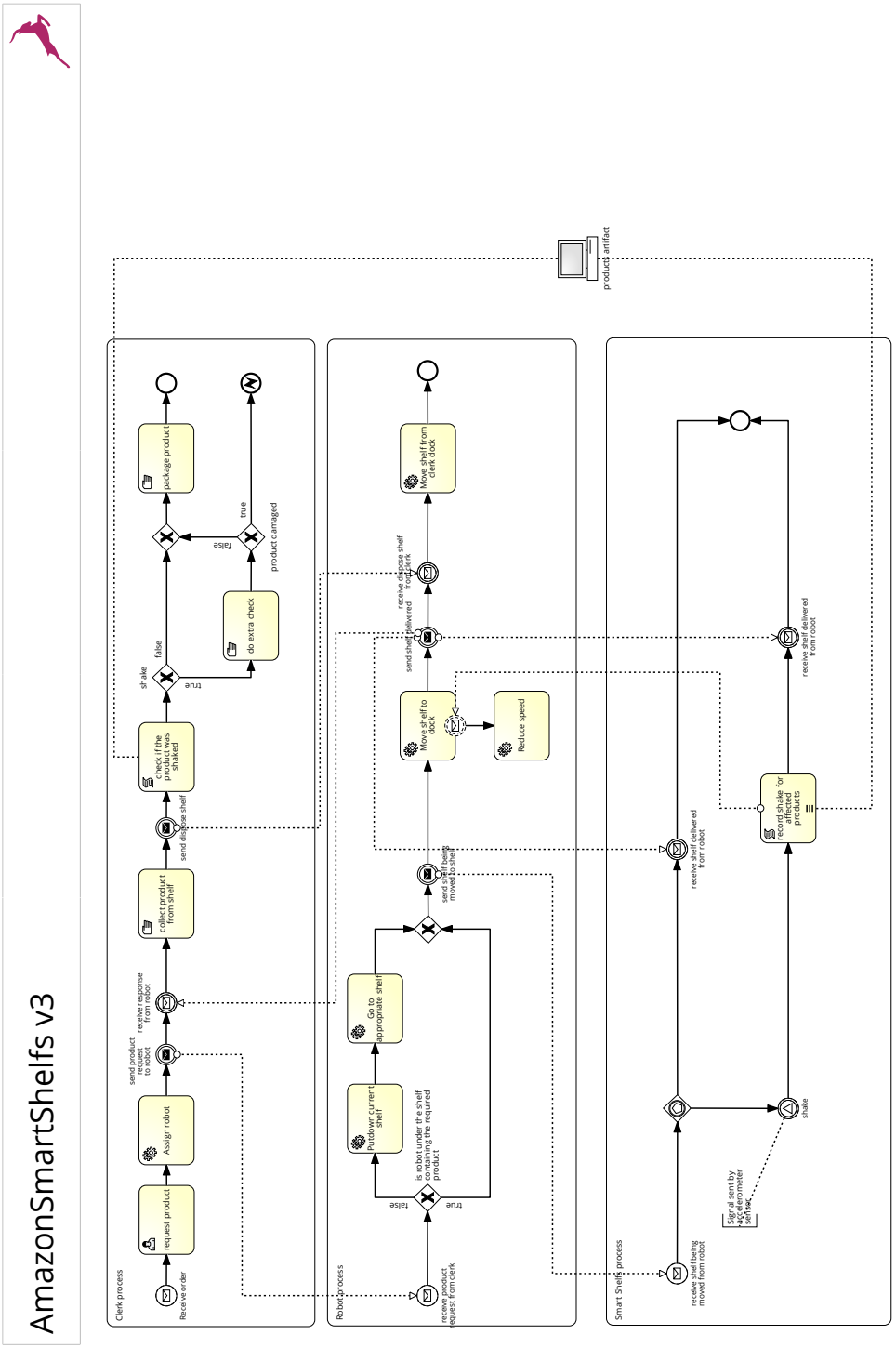
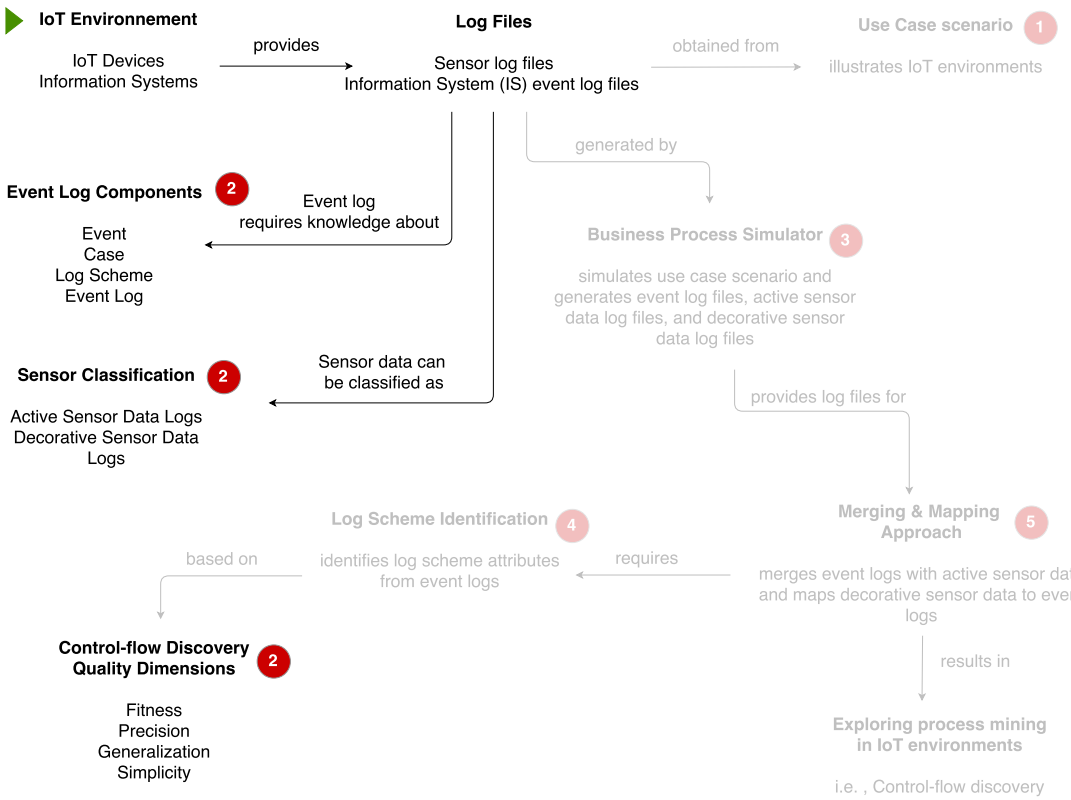


Figure 1.2: Use Case BPMN model

Chapter 2

Preliminaries



This chapter defines and formalizes the preliminary concepts used throughout this thesis report. Section 2.1 presents process mining and describes the classical process mining scope, then it highlights its limitations, and introduces an enhanced process mining scope. Section 2.2 formalizes the event log components. Section 2.3 presents the control-flow quality dimensions and illustrates how

these dimensions could be used to evaluate process models and finally Section 2.4 introduces a new classification of the sensors.

2.1 Process Mining

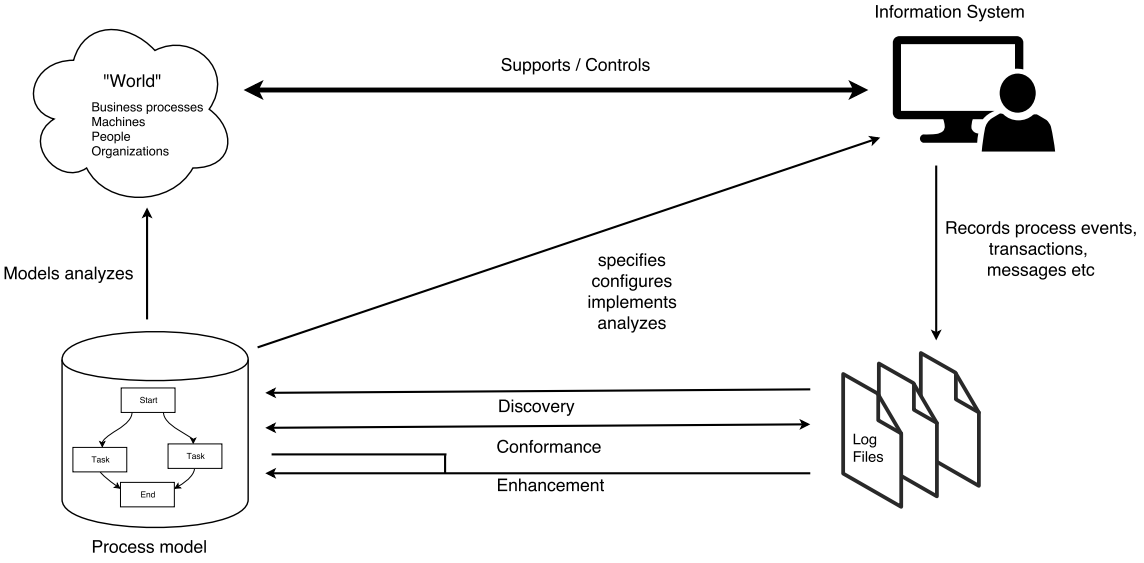
Process mining is an emerging field that bridges the gap between model-based process analysis and data-oriented analysis. It unifies data mining and machine learning concepts with process modelling and analysis approaches in order to discover, monitor, and enhance existing business processes [42, p. 8]. The event log files generated by information systems are the most important asset of any process mining algorithm. Over the last decade, event logs have become more and more available, and new process mining algorithms have emerged, which opened up opportunities for new application areas.

As shown in the Figures 2.1, the availability of event logs allows to perform the following process mining operations: (a) *Discovery* of a process model from an event log, (b) *Conformance* checking between a documented model and discovered model, and (c) *Evaluation* and extension of the existing process model. These three operations all share the same purpose that is to bring the digital world closer to the physical world. By looking at the classical process mining scope illustrated in Figure 2.1a, one can clearly notice that the information system plays a central role in the scope, since, it provides a close insight into the executed process activities. However, in real-world, this assumption is not always correct, especially if a significant portion of the business process is performed outside the information system (i.e., people doing manual work). In such circumstances, it becomes challenging to track the process control-flow and to record the executed events at a fine-grained level. An alternative solution is depicted in Figure 2.1b. Hereby, the information system is supported by IoT devices which serve as a medium to track the events that happen outside the information system scope. Therefore, it becomes possible to obtain comprehensive event log files which describe the business process execution at a fine-grained level. As mentioned in Chapter 1, one of the main challenges of this approach is the merging of information system log files, and sensor data log files, which is one of the core subjects investigated in this thesis report.

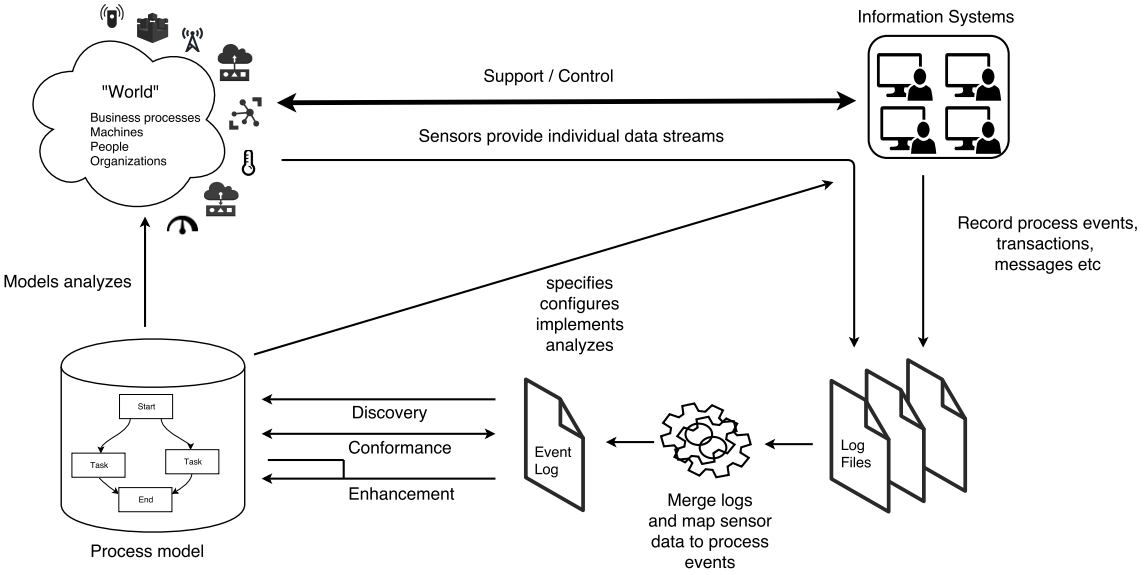
2.2 Event Log Components

This section describes some of the fundamental process mining definitions and necessary notations used to explain the approaches presented in this thesis report. As mentioned in Section 2.1 an event log is the golden asset of any process mining algorithm. In general, many process mining approaches assume the availability of a structured event log; therefore it is essential to clarify the general structure of an event log before defining its components.

An event log writes the events executed by one or many process instances. In the context of business process management (BPM), process instances are referred as cases. Each case contains a set of events preferably ordered by their time of occurrence, and each event is composed of a set of attributes. Table 2.1 presents a fragment of an event log describing an order to cash process. The first column is the *Case Id* which is a unique identifier assigned to each distinct case in the log. The Second column is the *Timestamp* which represents the moment in time when the event occurred. The timestamp is often divided into start-timestamp, and end-timestamp. The Third column is the *Event Name*. Finally, the last column is the *Ressource* which is the working entity responsible for handling the event. The presented event log does not reflect a generic structure for



(a) Classical Process Mining Scope [42]



(b) New Process Mining Scope

Figure 2.1: Comparison between two process mining scopes.

all existing event logs; however, some attributes are considered common in all event logs such as the *Case Id*, the *Timestamp*, and the *Event Name* [42].

Case Id	Timestamp	Event Name	Ressource
1	01-12-2017:08.15	Receive Order	Paul
1	01-12-2017:08.50	Prepare order	Paul
1	01-12-2017:10.25	Package order	Eric
2	01-12-2017:09.20	Receive Order	Eric
2	02-12-2017:10.15	Request product from supplier	Jacob
2	02-12-2017:11.35	Receive product	Martin
2	02-12-2017:14.54	Prepare order	Leon
3	02-12-2017:10.11	Receive order	Nadia
3	03-12-2017:13.40	Cancel order	Nadia

Table 2.1: Event log example

2.2.1 Event, Case, Event Log Scheme, and Event Log

In this section formal definitions for *event*, *case*, *event log scheme*, and *event log* are provided. These definitions are combined from existing work available in the literature [34] [51].

Definition 2.2.1 (Sequence)**.** Given a set $\mathcal{A}$, a finite sequence over $\mathcal{A}$ of length n is a mapping $s \in ([1,n] \subset \mathbb{N}) \rightarrow \mathcal{A}$, and it is represented by a string, i.e., $s = \langle s_1, s_2, \ldots, s_n \rangle$. Over a sequence s the following functions are defined:

- *selection operator* $(\cdot)$: $s(i) = s_i$, $\forall\, 1 \leq i \leq n$;
- $|s| = n$ (i.e., the length of the sequence).

Definition 2.2.2 (Event)**.** Let $\mathcal{A}$ be the set of all possible activities, and $\mathcal{C}$ be the set of all possible case ids. An *event* is a tuple $e = (a, c, t_s, t_e, d_1, \ldots, d_m)$, where:

- $a \in \mathcal{A}$ represents the activity associated to the event;
- $c \in \mathcal{C}$ represents the case Id;
- $t_s \in \mathbb{N}$ represents the start timestamp;
- $t_e \in \mathbb{N}$ represents the end timestamp;
- $d_1, \ldots, d_m$ is a list of additional event attributes, where $\forall\, 1 \leq i \leq m, d_i \in \mathcal{D}_i$, where $\mathcal{D}_i$ being the set of all possible attributes.

$\mathcal{E} = \mathcal{A} \times \mathcal{C} \times \mathbb{N} \times \mathbb{N} \times \mathcal{D}_1 \times \cdots \times \mathcal{D}_m$ is called the event universe. In an event e, the following projection functions are defined: $\pi_a(e) = a$, $\pi_c(e) = c$, $\pi_{t_s}(e) = t_s$, $\pi_{t_e}(e) = t_e$ and $\pi_{d_i}(e) = d_i, \forall\, 1 \leq i \leq m$. If e does not contain the attribute value d_i for some $i \in [1,m] \subset \mathbb{N}$, $\pi_{d_i}(e) = \perp$.

Definition 2.2.3 (Trace, Case)**.** A *trace* is defined as a finite sequence of events $\sigma_c = \langle e_1, e_2, \ldots, e_{|\sigma_c|} \rangle \in \mathcal{E}^*$ such that $\forall\, 1 \leq i \leq |\sigma|, \pi_c(e_i) = c \wedge \forall\, 1 \leq j < |\sigma_c|$, $\pi_t(\sigma_c(e_j)) \leq \pi_t(\sigma_c(e_{j+1}))$. In the context of this thesis report, each *case* is a grouping of events belonging to the same process execution and having same case id. Thus, each *case* is a distinct *trace*.

Definition 2.2.4 (Event Log Scheme)**.** A *Event Log Scheme* $S(\mathcal{E})$ is a finite set {"*caseid*","*activity*", "*starttime*","*endtime*","*attribute*$_1$" ..."*attribute*$_n$"} such that $|S(\mathcal{E})|= 4 + n$. In an event log scheme the projection function $\pi_s(e)$ is defined with $s \in S(\mathcal{E})$ and $e \in \mathcal{E}$ such that $\pi_{"activity"}(e) = \pi_a(e)$, $\pi_{"caseid"}(e) = \pi_c(e)$, $\pi_{"starttime"}(e) = \pi_{t_s}(e)$, $\pi_{"endtime"}(e) = \pi_{t_e}(e)$ and $\pi_{"attribute_i"}(e) = \pi_{d_i}(e), \forall\, 1 \leq i \leq n$ The event log scheme is used to define the header of an *Event Log*.

Definition 2.2.5 (Event Log)**.** An *Event Log* L is defined as a tuple $L = (S(\mathcal{E}), T)$ such that $S(\mathcal{E})$ is an event log scheme and T is a set of traces.

Definition 2.2.6 (Sensor Log, Sensor Data Entry)**.** Let E be the set of all possible sensor data entry identifiers, S be the set of all possible sensor data entry sources, and V be the set of all possible sensor data entry values. A sensor log $\mathcal{O}$ is an ordered sequence of sensor data entries, where each sensor data entry is a tuple $o = (s, t, v)$, with $s \in S$ is the sensor data entry source, $t \in \mathbb{N}$ is the sensor data entry timestamp, and $v \in V$ is the sensor data entry value.

2.3 Control-flow Discovery Quality Dimensions

The availability of an event log allows generating different process models depending on the discovery algorithm used. The generated models can be evaluated based on the following four quality dimensions: *Fitness*, *Precision*, *Generalization*, and *Simplicity* [43].

A model with high fitness allows all the traces in an event log to be replayed. A model is simple if it describes the behaviours inferred from the traces in the simplest way. Fitness and simplicity are not enough to evaluate a process model [42, p. 151]. As example Figure 2.2 depicts a *Flower Model* as BPMN diagram generated from the event log shown in Table 2.1. In this model, all possible sequence of events can be replayed.

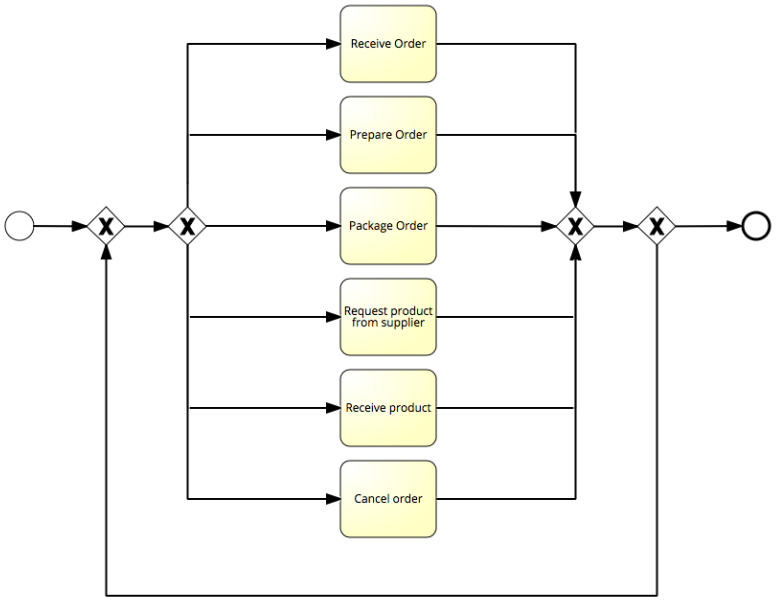


Figure 2.2: Flower Model generated from the example event log shown in Table 2.1.

Imprecise models (i.e., flower models) are referred as *Underfitting* models [42]. To avoid such models, it is necessary to consider precision. Indeed Precision restricts the allowed behaviours in a model. Nevertheless, a model should not be too restricted to the behaviours of the event log to allow also for the unseen behaviours (i.e., slightly different behaviours not yet recorded in the event log); otherwise, the model is *Overfitting* [42]. To balance between underfitting and overfitting, generalization is considered in parallel with precision. Indeed generalization allows the model to be flexible with the unseen behaviours.

The control-flow discovery quality dimensions described in this chapter are the main building block of a state-of-art approach aiming at automatically inferring case ids from event logs. The following sub-sections illustrate different techniques allowing to measure the four quality dimensions. As it will be presented in Section 4.4.4 (cf. Chapter 4), these techniques have been chosen because they have demonstrated high accuracy in inferring case ids. Sections 2.3.1, 2.3.2, 2.3.3, and 2.3.4 explain *Fitness*, *Precision*, *Generalization*, and *Simplicity* respectively.

2.3.1 Fitness

Fitness represents the ability of a process model to reproduce the control-flow of the traces recorded in the event log. Measuring fitness can be performed using several approaches. Rozinat et al. in [38] measure fitness by replying the log events onto a Petri-net [30] to detect possible mismatches. *"Alignment-based Replay Fitness"* is an alternative approach proposed by van der Aalst in [1] to quantify fitness. It uses the alignment technique [42] to align each trace of the event log with the closest trace that the model can generate. This way it becomes possible to quantify the extent to which the traces observed in the log can be reproduced in the given process model.

Finding the optimal alignment implies assigning a cost to any misalignment found while replaying

a log trace onto the process model, such that when a trace is about to diverge from the process model, a move to the next event in the trace or in the process model (asynchronous move) is performed and the total alignment cost is increased. It is possible to assign different costs to each process activity or each move type (move on model or move on trace). In general, it is assumed that deviations from the replayed log traces are more expensive than deviations from the process model [9].

The total fitness score is defined as the sum of each trace alignment cost multiplied by its frequency and divided by the log-model alignment cost without synchronous moves [9]. Let n be the number of distinct traces in a log, c_i the cost of trace i, and f_i the frequency of trace i, then $\sum_{i=0}^{n} c_i \times f_i$ is the sum of each trace alignment cost multiplied by its corresponding frequency. Also let e_i the number of events in trace i, and T the cost of an asynchronous move on the trace, then $\sum_{i=0}^{n} f_i \times e_i \times T$ is the total move on log cost. Moreover let l be the total number of traces in the log, M the cost of an asynchronous move on the model, and s the number of events of the shortest trace in the model, then $l \times s \times M$ is the cost of executing the process model without synchronous moves. The total fitness score is computed as shown in Equation 1. The optimal alignment approach and total fitness calculation are illustrated in Example 2.3.1.

$$fitness = 1 - \frac{\sum_{i=0}^{n} c_i \times f_i}{(\sum_{i=0}^{n} f_i \times e_i \times T) + l \times s \times M} \tag{1}$$

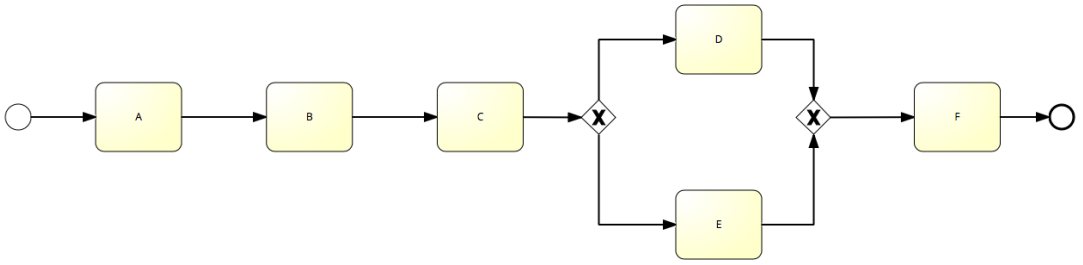


Figure 2.3: Example of BPMN model.

Trace	Frequency
A B C D F	40
A B C E F	40
A E B C F	20
A B D E F	10
A B B C E F	10

Table 2.2: Log traces and their corresponding frequency

Example 2.3.1. Figure 2.3 shows a possible BPMN model generated from the event log shown in Table 2.2. Note that by using different control-flow discovery algorithms [42], different process models can be obtained, therefore, the BPMN models used in this example and the following examples are not necessarily the only models.

It is clear that the model allows only for two possible traces that are $A\ B\ C\ D\ F$ and $A\ B\ C\ E\ F$. An optimal alignment for each trace of Table 2.2 is shown in alignment Tables 2.3, 2.4, 2.5, 2.6, and 2.7 respectively.

From the alignments shown in Tables 2.3 and 2.4, it is notable that traces $A\ B\ C\ D\ F$ and $A\ B\ C\ E\ F$ are perfectly aligned with the process model depicted in Figure 2.3, hence the alignment cost is 0. However, the traces aligned in Tables 2.5, 2.6, and 2.7 cannot be fully replayed on the process model (Figure 2.3). Therefore, asynchronous move operations (labelled with >> symbol) are introduced to obtain an optimal alignment; consequently, an alignment cost is assigned to each asynchronous move operation depending on its type. Assuming that an asynchronous move on the model cost 2, and an asynchronous move on the trace cost 5, the total alignment cost for trace $A\ B\ C\ D\ F$ is 0; the total alignment cost for trace $A\ B\ C\ E\ F$ is 0; the total alignment cost for trace $A\ E\ B\ C\ F$ is 7; the total alignment cost for trace $A\ B\ D\ E\ F$ is 7, and the total alignment cost for trace $A\ B\ B\ C\ E\ F$ is 2.

To calculate the total fitness score, one needs to calculate the sum of each trace alignment cost multiplied by its corresponding frequency, divided by the log-model alignment cost without synchronous moves. In this example, l be the total number of traces in the log is 120, M the cost of an asynchronous move on the model is 2, and s the number of events of the shortest trace in the model is 5. Using Equation 1 the total fitness score calculations are shown in Equation 2.

$$fitness = 1 - \left[\frac{(0 \times 40) + (0 \times 40) + (7 \times 20 + (2 \times 10))}{\begin{array}{c}(40 \times 5 \times 5) + (40 \times 5 \times 5) + (20 \times 5 \times 5) + \\ (20 \times 5 \times 5) + (10 \times 6 \times 5) + 120 \times 5 \times 2\end{array}} \right] = 0.93 \tag{2}$$

Finally, the total fitness score for this example is 0.93.

Model	A	B	C	D	F
Trace	A	B	C	D	F

Table 2.3: Alignment between process model Figure 2.3 and trace $A\ B\ C\ D\ F$

Model	A	B	C	E	F
Trace	A	B	C	E	F

Table 2.4: Alignment between process model Figure 2.3 and trace $A\ B\ C\ E\ F$

Model	A	>>	B	C	D || E	F
Trace	A	E	B	C	>>	F

Table 2.5: Alignment between process model Figure 2.3 and trace $A\ E\ B\ C\ F$

Model	A	B	C	D	>>	F
Trace	A	B	>>	D	E	F

Table 2.6: Alignment between process model Figure 2.3 and trace *A B D E F*

Model	A	B	>>	C	E	F
Trace	A	B	B	C	E	F

Table 2.7: Alignment between process model Figure 2.3 and trace *A B B C E F*

2.3.2 Precision

The precision quality dimension is used to quantify the extra behaviours allowed by the generated process model that are not recorded in the log. In case the process model contains loops, this will generate infinite behaviours. Therefore, counting the number of all possible traces is impossible. As solution one can estimate only the allowed behaviors which can be done using several approaches such as the *Escaping edges* technique [9]. Using this technique, a decision choice is defined as a decision point in the process model (i.e., gateway) where different control-flows are possible, and an escaping edge is defined as a decision choice in the process model that was never seen in the log.

The total precision score can be obtained from a partial state space that shows the possible transitions within the process model. Figure 2.5 depicts an example of partial state space generated from the process model shown in Figure 2.4 and the event log presented in Table 2.8. Each state represents a visited event, each transition represents a possible control-flow from that event, and each label attached to a specific state represents the number of traces in the log having similar control-flow. For instance, in the partial state space (Figure 2.5), there are 60 traces with the control-flow *A B C D E*. Note that the escaping edges in this example are represented with crossed transitions.

Given a partial state space, the total precision score is the sum of each visited state multiplied by the difference between the total number of outgoing edges from the state in the partial state space and the number of outgoing edges from the state recorded in the log. In other words, let n represent the number of states in a partial state space, v_i represents a visited state i, o_i represents the total number of outgoing edges from state i in the partial state space, and u_i represents the number of outgoing edges from state i recorded in the log, then the total precision can be computed as shown in Equation 3.

$$precision = 1 - \frac{\sum_{i=0}^{n} v_i \times (o_i - u_i)}{\sum_{i=0}^{n} v_i \times o_i} \tag{3}$$

Example 2.3.2 illustrates the escaping edge approach.

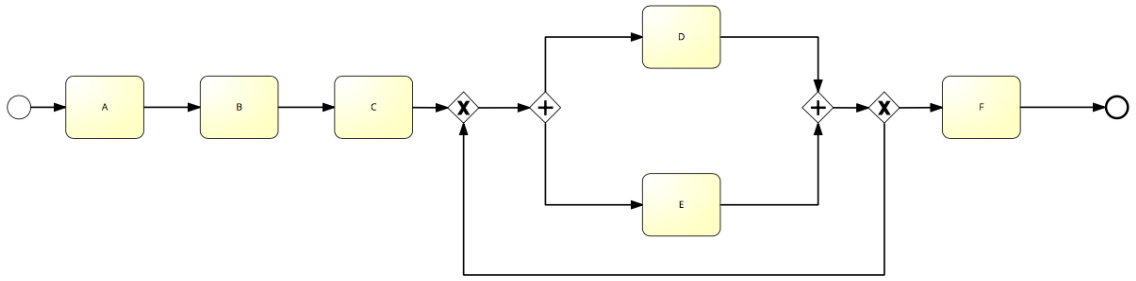


Figure 2.4: Example of a possible BPMN model.

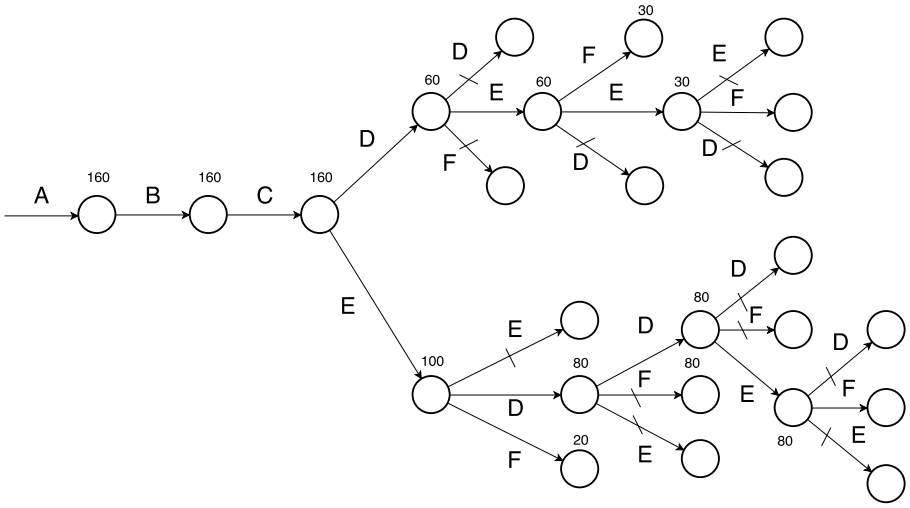


Figure 2.5: A transition system showing the partial state space of the possible transitions within the process model depicted in Figure 2.4.

Trace	Frequency
A B C D E F	30
A B C D E E F	30
A B C E F	20
A B C E D D E F	80

Table 2.8: Log traces and their corresponding frequency

Example 2.3.2. Given the process model depicted in Figure 2.4, and the corresponding event log is shown in Table 2.8, it is possible to calculate the precision score for the process model using Equation 3. Firstly, one needs to generate a transition system representing the partial space of the possible transitions within the process model. Using both the process model (Figure 2.4) and the corresponding event log (Table 2.8), the corresponding transition system is depicted in Figure 2.5. Note that the escaping edges are crossed transitions. Using Equation 3 the total precision is

calculated as shown in Equation 4. Indeed the calculation shows that the obtained process model has low precision.

$$precision = 1 - \left[\frac{\begin{array}{r} 160(1-1) + 160(1-1) + 160(2-2) + 60(3-2) + 60(3-2) + \\ 30(3-2) + 100(3-2) + 80(3-2) + 80(3-2) + 80(3-2) \end{array}}{\begin{array}{c} 160(1) + 160(1) + 160(2) + 60(3) + 60(3) \\ + 30(3) + 100(3) + 80(3) + 80(3) + 80(3) \end{array}} \right] = 0.77 \quad (4)$$

2.3.3 Generalization

The generalization dimension is used to quantify the extent to which the generated process model can replay log traces that are not yet recorded in the event log [42]. There exist several approaches to estimate generalization such as *The frequency of use* approach, which is based on the assumption that a generic model is a model whose parts are all used with the same frequency [9]. In this context, given a generated business process model (in BPMN for example), one needs to consider the frequency of each node (i.e., activities and gateways). Buijs in [9] used a process tree as a reference to determine the generalization score. However, for sake of simplicity this example uses a BPMN model instead, thus, the same technique is adapted, but this time using a BPMN model as reference.

The frequency of use can be calculated as follow: Let n represents the total number of nodes in the model, i represents a node, v_i represents the number of times node i was visited when the log traces are replayed onto the process model, then the total generalization score can be calculated as shown in Equation 5. Example 2.3.3 illustrates the frequency of use approach.

$$generalization = 1 - \frac{\sum_{i=0}^{n} \sqrt{v_i}^{-1}}{n} \quad (5)$$

Example 2.3.3. Given the process model depicted in Figure 2.6, and the corresponding event log shown in Table 2.9, it is possible to calculate the generalization score for the process model. Firstly, one need to obtain the frequency of process model nodes, which can be done by counting the number of times each node in the process model was visited in the event log. Table 2.10 presents the frequency of the nodes obtained from the process model depicted in Figure 2.6, and the log traces shown in Table 2.9. Note that the table shows both activities and gateways of the BPMN model. Finally, the generalization score can be calculated using Equation 5 as shown in Equation 6

$$generalization = 1 - \left[\frac{\begin{array}{c} \sqrt{110}^{-1} + \sqrt{110}^{-1} + \sqrt{110}^{-1} + \sqrt{90}^{-1} + \sqrt{20}^{-1} + \sqrt{50}^{-1} \\ \sqrt{40}^{-1} + \sqrt{110}^{-1} + \sqrt{110}^{-1} + \sqrt{90}^{-1} + \sqrt{90}^{-1} + \sqrt{110}^{-1} \end{array}}{12} \right] = 0.88 \quad (6)$$

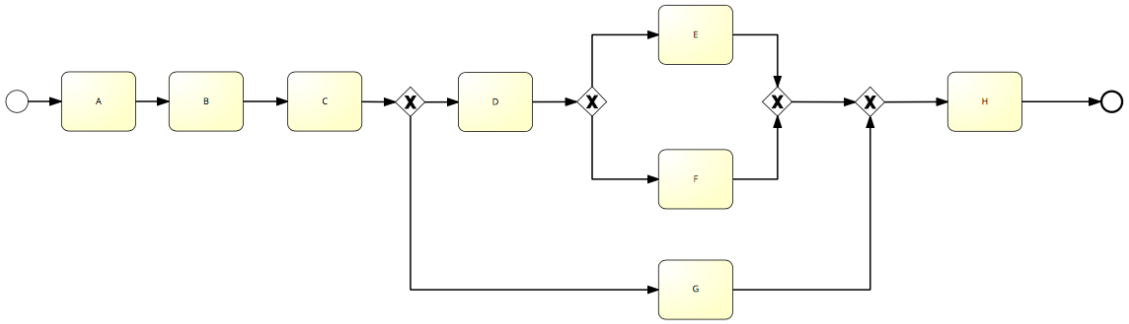


Figure 2.6: Example of possible BPMN model.

Trace	Frequency
A B C D E H	50
A B C D F H	40
A B C G H	20

Table 2.9: Log traces and their corresponding frequency

Node	Frequency
A	110
B	110
C	110
D	90
G	20
E	50
F	40
H	110
XORsplit(D,G)	110
XORsplit(E,F)	90
XORjoin(E,F)	90
XORjoin(XORjoin(E,F),G)	110

Table 2.10: Frequency of the nodes obtained from the process model depicted in Figure 2.6, and the log traces shown Table 2.9. Note that XORsplit, and XORjoin refer to BPMN gateways considered also as visited nodes.

2.3.4 Simplicity

The simplicity dimension is used to quantify the extent to which a generated process model is simple. It is defined according to two main principles: (a) The the Occam's Razor principle which states "One should not increase, beyond what is necessary, the number of entities required to explain anything." (b) The understandability of the process model by the user [9]. Figl [21] analyzed the

origin of the cognitive effort in comprehending process models in a systematic literature review based on the results of several empirical studies conducted to identify the factors affecting the understandability of process models. Furthermore, Reijers and Mendling in [37] pointed-out several factors affecting the process model understanding. The factors can be classified into cognitive factors such as abstraction level, model size, and hidden dependencies, and personal factors such as user expertise, and personal reasoning capabilities. The simplicity dimension of a generated process model is affected by the cited cognitive factors.

The literature presents several heuristics that could be used to estimate simplicity such as the *Simplicity by activity occurrence* [9]. This heuristic assumes that the less duplicate activities there are in a generated process model, the more simple it is.

Let d be the number of duplicated activities in a process model, m the number of missing activities (activities present in the log file but not on the generated process model), n the number of nodes in the process model, and a the number of distinct activities in the process model, then the simplicity score can be expressed as shown in Equation 7. This approach is illustrated in Example 2.3.4.

$$simplicity = 1 - \frac{d + m}{n + a} \tag{7}$$

Example 2.3.4. By comparing the generated process model depicted in Figure 2.7, and its corresponding event log presented in Table 2.11, the variables of the simplicity Equation 7 can be inferred. Clearly, the number of nodes n is 14, the number of distinct activities a is 6, the number of duplicated activities d is 2, and the number of missing activities m is 1. Hence, simplicity can be calculated using the formula on Equation 7 as shown in Equation 8.

$$simplicity = 1 - \frac{2 + 1}{14 + 6} = 0.85 \tag{8}$$

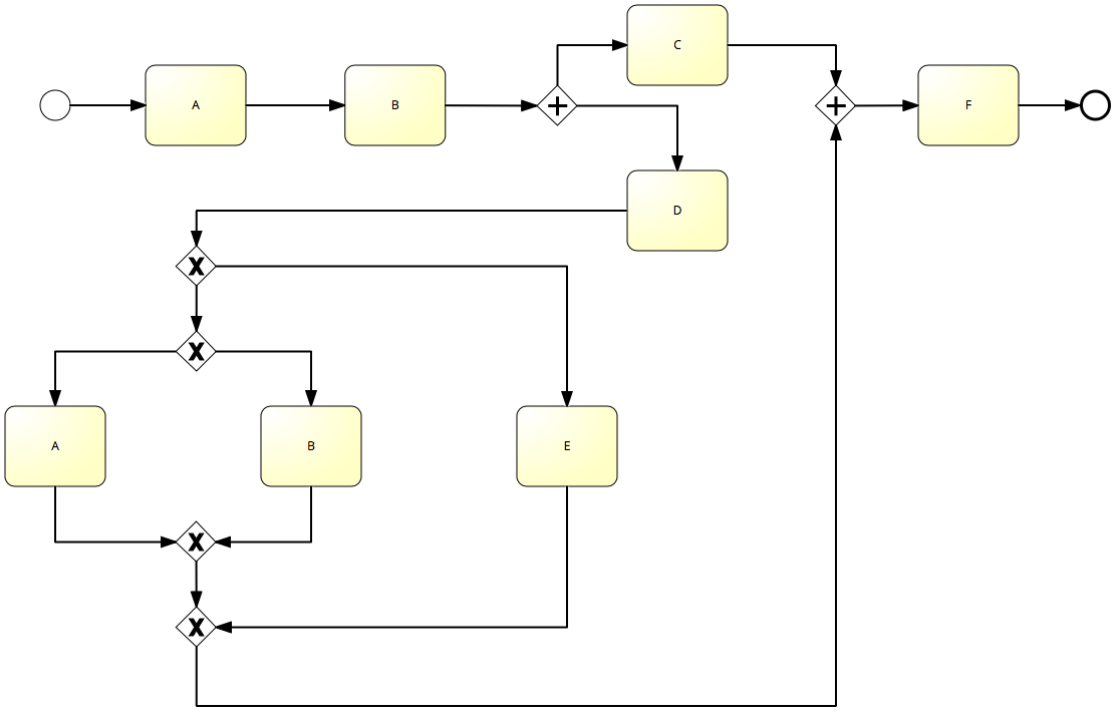


Figure 2.7: Example of BPMN model obtained from an event log.

Trace	Frequency
A B C D A F	30
A B C D B F	50
A B D K E C F	70

Table 2.11: Log traces and their corresponding frequency

2.4 Sensors Classification

The massive amount of data coming from IoT devices opened up the need for new approaches to mine models representing sensors' environment dynamics. As will be presented in the related work Section 5.2 (cf. Chapter 5) the number of publications exploring process mining capabilities in this area increased notably during the past years. Since then, several process discovery approaches [17], and conformance checking approaches [39] have emerged in this context. To cope with sensor data using process mining, a fine-grained analysis of sensor data properties is required.

By looking at the different environments where IoT devices are deployed, a new classification of sensors that allow treating each sensor type independently of each other is introduced. In fact, classifying sensors is not a new approach. The APUBS system [5] also tried to categorize sensors to mine user-behaviors in smart environments (i.e., smart house). The provided classification

distinguishes between object sensors (sensors plugged into physical object to track the human interactions), environment sensors (sensors deployed over the external environment such as temperature sensors), and position sensors (sensors used to provide information about the user position). This thesis report generalizes the APUBS approach by considering only two types of sensors entitled *Active Sensors* and *Decorative Sensors*. This generalization comes out in the context of merging sensor data log files with information systems (IS) log files, such that active sensor data are considered as atomic events that are directly merged with IS events, whereas, decorative sensor data are seen as sources of complementary knowledge that serve to decorate atomic events. More insight about merging and mapping sensor data is provided in Chapter 5.

The active sensor type includes all the sensors used to report the interactions between users and objects. In the presented use case scenario (cf. Chapter 1, Section 1.4), RFID sensors are classified as active sensors because they are deployed to track clerks progress on manual activities such as *check if product was shacked* activity, and *package product* activity. Furthermore, they are also used to exchange data and trigger different sub-processes. The decorative sensor type includes all sensors responsible for updating the process model with the state of the external environment. Decorative sensors are used in the use case scenario to enrich the process model with a continuous stream of information coming from the external environment. For instance, accelerometer sensors are classified as decorative sensors because they are meant to report the acceleration coordinates of shelves in real-time, which allow inferring possible shakes. Table 2.12 compares the features of active sensors and decorative sensors.

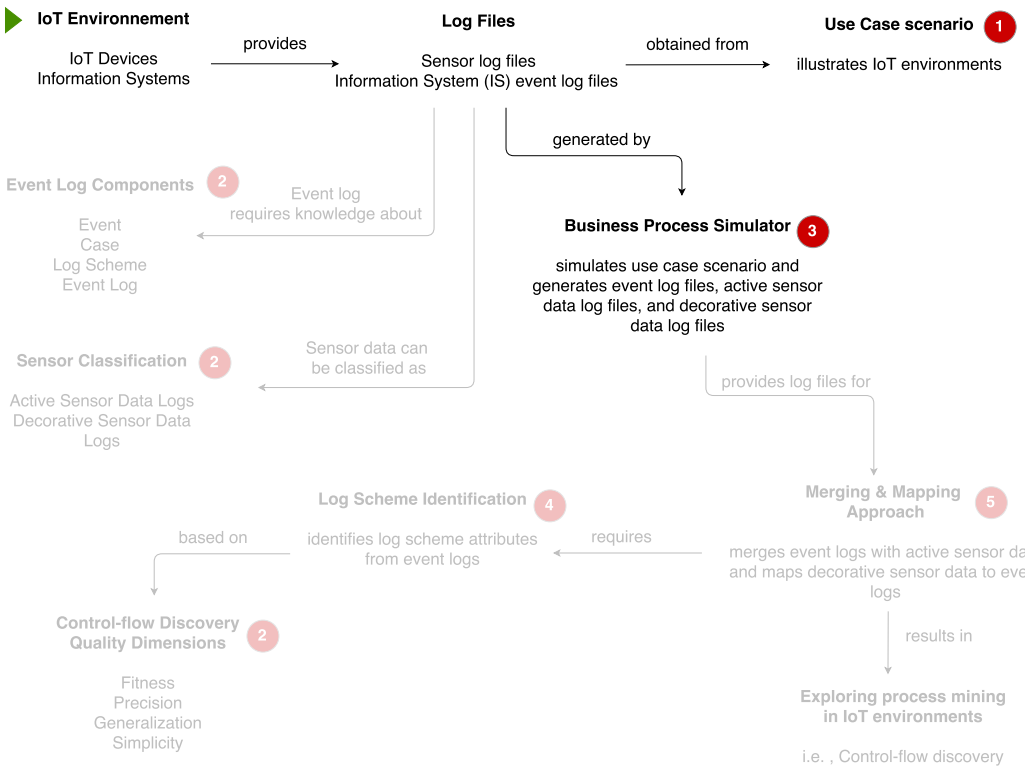
Features	Active S.	Decorative S.
Discrete data stream	✓	✓
Continuous data stream		✓
Message passing between processes	✓	
Track progress of manual activities	✓	
Track state changes in external environment		✓

Table 2.12: Comparison between Active Sensors and Decorative Sensors

The classification provided in this report applies also to the sensor types proposed by the APBUS system. The object sensors and position sensors in APBUS can be categorized as active sensors, and the environment sensors can be classified as decorative sensors. Therefore, the new classification aligns with the APUBS classifications except that it abstracts from the ambient systems (i.e., smart houses) application area and generalizes over all the other potential applications of IoT devices where mapping sensor data to process models is important.

Chapter 3

Process Simulation



This chapter describes the design and the implementation characteristics of the use case simulator. Section 3.1 provides an overview of the simulation models. Section 3.2 presents the background and the related work. Section 3.3 highlights the different concepts and mechanisms considered when designing the simulator such as queuing and randomness. Finally, Section 3.4 describes the key features of the implementation.

3.1 Overview

The starting point of any process mining algorithm is an event log organized in a process-oriented perspective [42, p. 95]. Such event logs are usually not available at first glance; thus, some pre-processing of the data sources is required. Depending on the business process complexity, the data necessary for process mining can be distributed across different data sources such as databases, files, message logs, and sensors logs [17]. Due to the increasing demand for data sets to explore process mining capabilities, several initiatives have been made by different communities such as the BPM community, to share event logs and make them available publicly [1]. However, the proposed event logs are limited to some specific areas and do not reflect a wide variety of settings. Moreover, requesting real-world data fitting some particular requirements and settings from companies can be challenging, especially if the data required is considered as a competitive asset by the company [10].

In the context of IoT environments, although, sensors provide a massive amount of data about their external environment, it is still challenging to find real-world log files that combine data coming from both sensors and information systems. Moreover, the existing simulators and log generators such as PLG [11] and PLG2 [10] do not provide enough simulation mechanisms to mimic the interaction between sensors and business processes. Thus, it is necessary to design and implement a tailored simulator where sensors and business processes continuously interact to achieve a set of tasks.

3.2 Background and Related Work

Over the past fifty years, process simulation has been used in several computer science areas. A known example is SIMULA language that was developed in 1965 to allow discrete event simulation [25]. Since that time several simulation languages and tools have been developed. In the business process management (BPM) field, many work-flow management systems such as IBM WebSphere [2], and Adaptive Case management systems such as Exformatics DCR graphs [3] provide simulation toolkits. However, none of these simulators is developed for the purpose to evaluate process mining algorithms.

Van der Aalst et al. in [44] describe a set pitfalls of the current simulation techniques such as the improper modeling of resources. The paper also provides a set guidelines to follow when designing a process simulator especially from the resource perspective where a new manner of characterizing resources availability is presented. Using CPN tools [24], the authors were able to demonstrate the usefulness of the suggested resource characteristics in designing business process simulators.

Van Hee and Liu in [46] proposed a technique to generate random test data sets using Petri-nets random classes. The top-down approach proposed uses stepwise construction rules such as refinement rules to generate process models belonging a to a specific class randomly. Burattin and Sperduti in [11] presented a process logs generator (PLG) tool that allows to generate event log files from a business process model designed inside the tool or imported as a BPMN file. This work overcomes the complexity of Petri-nets by using dependency graphs instead. The tool implements a set of process patterns such sequence, concurrency, mutual exclusion, and repetition, and allows to associate rules and probabilities to each pattern. The second version of PLG (PLG2) [10]

[1] Public events logs are available on `https://data.4tu.nl/repository/collection:event_logs`.
[2] See `https://www-01.ibm.com/software/dk/websphere/`
[3] See `http://wiki.dcrgraphs.net/`

developed by Burattin takes the tool a step forward by enabling the generation and simulation of multi-perspective process models in both offline and online settings. However, no existing simulator allows mimicking the interactions between sensors and process instances in a BPM context.

3.3 Simulation Model

A business process simulator (BPS) is a computer model meant to imitate the behaviour of real-world business processes [26, p. 223]; thus, allowing to evaluate, predict, analyze, and optimize the performance of a process design in a controlled environment. It comes as an alternative to developing complex mathematical models to evaluate the decision-making problems involved in a business process.

Simulation models can be seen from the following three different perspectives: static or dynamic, discrete or continuous, and deterministic or stochastic [26, p. 224]. A static model ignores the time aspect; thus the process execution is independent of time, while a dynamic model considers the time aspect. A deterministic model is a model where the simulation output can be correctly predicted if the input is known, unlike a stochastic model where the simulation output is unknown apriori. A discrete model allows coping with a discrete sequence of events where each event occurs at a specific time instance and results in a state change within the system, while a continuous model is meant to cope with continuous state changes. Simulating sensors interactions with business processes requires a dynamic, stochastic, and continuous model, whereas, existing BPSs such as PLG [11] deploy a dynamic, stochastic, and discrete model [26, p. 224].

3.3.1 Queuing

One of the main functionalities of a BPS is to imitate the execution of several process instances sequentially or in parallel, which implies the interaction of several entities according to a set of rules and standards to achieve a specific task. In this context, it is important to distinguish between transient entities and resident entities [26, p. 225]. In a business process context, a transient entity is a temporary job that flows from the initial state to the end state of a process such as a case, while a resident entity is a long-lasting entity often responsible for handling transient entities such as workstations, or resources [22, p. 232].

Exploiting the resources responsible for executing process instances is one of the important roles of a BPS. Unless the number of resources is not restricted, a BPS should pool these resources accordingly between the running process instances, which results in the necessity of queuing some of the process instances until the required resources are available. Therefore, queuing is a BPS component. Depending on the simulated use case, queuing should be controlled with several approaches such as first-in-first-out (FIFO), last-in-first-out (LIFO), or priority queues [26, p. 275], and designed in such a way that it can cope with frequent concurrency drawbacks such as deadlocks and starvations.

Resource pooling and queuing strategies influence directly the total waiting time of a process instance, which might be subject to simulation of as well (i.e., simulating a service system) [26]. Another essential property to consider in parallel with queuing is balking that is the behaviour of allowing a process instance to leave the queue at any time (i.e., instance termination). This property allows introducing some infrequent behaviours to the simulated process referred as noise

in the process mining domain [42, p. 148]. Noise is a frequent phenomenon that is encountered in real-world event logs, thus, it is important to consider it when designing a BPS.

Another important factor that affects the queuing time is resource availabilities which can be defined with a set of parameters usually provided as input to the BPS. The same parameters are used as process performance measures [18] to quantify cost per execution, resource utilization, and waste in a business process. Among these parameters, the followings are considered:

- **Arrival rate** λ is the average number of process instances (i.e., cases) arriving per time unit [44], which can be modelled using the Poisson process. Let T be the time between the arrival of two consecutive cases, $X(t)$ a random variable representing the number of cases that arrive in a time interval t, and λ the probability of a case arriving within time interval t. Then one can estimate $X(t)$ using a Poisson distribution with mean value λt. The probability that n cases arrive at t is defined in Equation 1.

$$P(X(t) = n) = \frac{(\lambda t)^n e^{-\lambda t}}{n!} \tag{1}$$

  Under the assumption that the cases arrival time is independent, exponential, and identically distributed, the mean arrival rate for a Poisson process defined with $X(t)$ such that $t >= 0$ is λ and the average time between two subsequent cases is $\frac{1}{\lambda}$.

- **Service rate** μ is the average number of cases to be processed per time unit and the mean processing time is $\frac{1}{\mu}$ such that $\mu > 0$

- **Utilization rate** ρ is obtained by dividing the arrival rate over the service rate ($\rho = \frac{\lambda}{\mu}$), and refers to the probability that a resource is busy.

  Other parameters such as chunk size, horizon, and availability rate [44] can be used to better leverage the resources utilization, however, those parameters were abstracted from this design, and the focus was mainly on the parameters described previously.

3.3.2 Randomness

Randomness is an important feature of any BPS. A typical manner of imitating randomness in real-world business processes is to gather a data sample representing the overall population and use it as a reference to generate similar data. However, in the absence of sample data, process documentation and experts' knowledge might allow to obtain some approximations that could be used to adjust randomness in a BPS. For instance, by knowing the minimum and the maximum activity service time, one can deploy a uniform distribution model [26, p. 340]. A more efficient alternative to better leverage an activity service time in a simulation model in the presence of more data is to use the Beta distribution [26, p. 341]. For example, given the knowledge from the business process documentation that an activity requires a minimum of 10 minutes, a maximum of 30 minutes, and an average of 22 minutes to be completed, and considering the clerks estimation, who affirms that the activity is more likely to take 15 minutes, the Beta distribution can be used with following parameters: minimum $a = 10$, maximum $b = 30$, average $\bar{x} = 22$, and mode $c = 15$. Beta distribution is calculated according to two parameters α and β. By determining these parameters it is possible to calculate the mean μ and the mode c as shown in Equations 1 and 2.

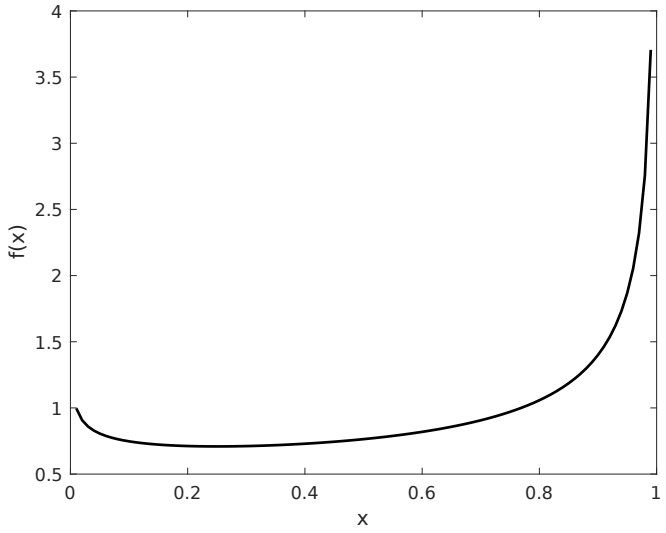


Figure 3.1: Beta Distribution with $\alpha = 0.85$ and $\beta = 0.57$

$$\mu = a + \frac{\alpha(b-a)}{\alpha+\beta} \tag{1}$$

$$c = a + \frac{(\alpha-1)(b-a)}{\alpha+\beta-2} \tag{2}$$

Given the true mean $\bar{x}$, and the estimated mode c, it is possible to calculate α and β as shown in Equations 3 and. 4

$$\alpha = \frac{(\bar{x}-a)(2c-a-b)}{(c-\bar{x})(b-a)} \tag{3}$$

$$\beta = \frac{\alpha(b-\bar{x})}{\bar{x}-a} \tag{4}$$

The Beta distribution allows to explore various properties provided by the distribution model. For example, the uniform distribution can be obtained by setting $\alpha = 1$ and $\beta = 1$. Moreover, a random variable x obtained from the Beta distribution function can be scaled from the interval $[0, 1]$ to any interval $[A, B]$ using the transformation shown in Equation 5.

$$A + (B-A)x \tag{5}$$

Now let's consider the example where the minimum $a = 10$, the maximum $b = 30$, the average $\bar{x} = 22$, and the mode $c = 15$. Using Equations 3 and 4, the obtained values are $\alpha = 0.85$ and $\beta = 0.57$ respectively. The Beta distribution shape is depicted in Figure 3.1 where x is the probability ratio and $f(x)$ is the probability density.

The Beta distribution can be seen as the probability distribution of probabilities. Note that the x values obtained from Figure 3.1 should be scaled to the range $[10, 30]$ using the transformation in 5.

Thus, in this example it is excepted that most of the randomly generated values will be within the range of $[24, 30]$

Randomness is an indispensable feature in any BPS, the probability distribution models such as the uniform distribution, and the beta distribution can be used to set activity service times randomly, deferred choices decisions, and priorities.

3.4 Implementation

The implementation is developed in Java. It comprises the components depicted in Figure 3.2. The *Controller* implements the *Use Case Scenario* described in Section 1.4 (cf. Chapter 1), and the *Simulation Model* presented in Section 3.3. The *Controller* takes as input a set of configuration parameters, and populate the *Data Model* holding the system entities (i.e., robots, shelves, and products). The work-flow is handled by the *Thread Pool* component and the interactions between the system entities are recorded into log files using the *Logger* component. Lastly, the simulation run time is reduced to the order of milliseconds using the *Artificial Clock* component responsible for reducing the system clock granularity.

The set of assumptions, and configuration parameters considered while developing the BPS are presented in Sections 3.4.1, and 3.4.2 respectively. The approach to generating artificial log files is described in Section 3.4.3. Furthermore, some technical details related to multi-threading, threads pooling, and the artificial clock thread used to leverage the simulated control-flow are highlighted in Section 3.4.4.

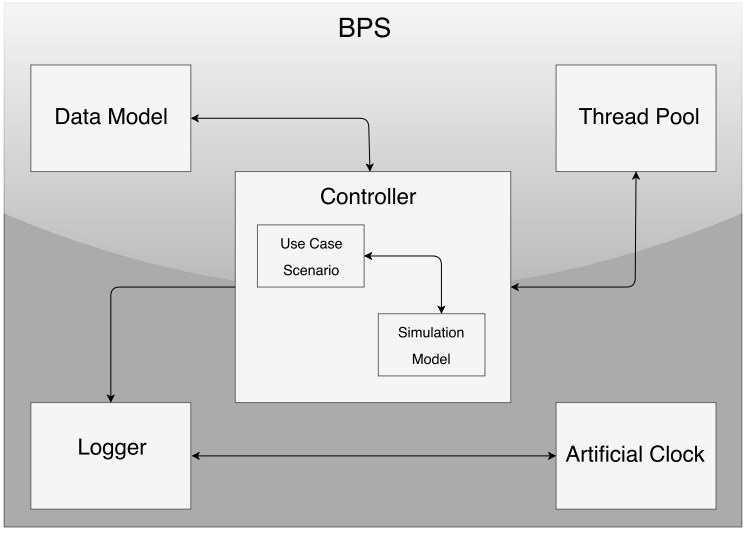


Figure 3.2: BPS Architecture

The BPS implementation is available at the Github Repository `https://github.com/DTU-SE/AmaSmart`.

3.4.1 General Assumptions

The assumptions considered while developing the BPS application are linked to the use case scenario defined in Section 1.4 (cf. Chapter 1) and the simulation model explained in Section 3.3.

To recapitulate, in Section 1.4 a use case scenario illustrating an IoT environment was presented. The scenario is inspired by the Kiva robots deployed at Amazon warehouses. The role of the Kiva robots is to transport mobile shelves containing inventory products to clerks. The use scenario has three processes that are *Clerk Process*, *Robot Process*, and *Smart Shelf Process*. As depicted in Figure 1.2 (cf. Chapter 1), the *Clerk Process* is responsible for requesting products from an information system, and assigning a robot to pick-up the shelf containing each product. Once the robot delivers the shelf, the product is collected from the shelf, checked in case it was shacked while being moved, and packaged. The *Robot Process* is responsible for bringing the shelf to the clerk dock, and disposing of the clerk dock it once the product is collected from the shelf. The *Smart Shelf Process* is responsible for interpreting the Accelerometer data provided by an Accelerometer plugged into the shelf to infer shakes. A more detailed description of the use case scenario is provided in Section 1.4 (cf. Chapter 1).

For the use case scenario the following assumptions are made:

- A stock contains a limited number of product types, and each product type has a unique identifier.
- The product inventory is abstracted; thus, it is assumed to have an unlimited amount of products of the same kind in the stock.
- The products are distributed over a set of shelves, each shelf contains a fixed number of products, and each product is available on only one shelf.
- All shelves have the same products capacity that should not be exceeded.
- At any point in time, a shelf can be in one of the following states: idle or busy. A busy shelf is a shelf being moved by a robot.
- If a shelf is busy, no other robot can access it until the shelf gets back to the idle state.
- Each shelf is equipped with an accelerometer that continuously reports some accelerometer data from which shakes can be inferred.
- In case the accelerometer detects a shake, the robot decreases its current speed by half, which increases the service time of activity *move shelf to dock*.
- In order to track the progress of manual activities (i.e., package product) and message passing between subprocesses (i.e., receive product request from clerk) RFID sensors are used.
- Each RFID sensor has a chip identifier that reads the RFID code of the scanned element (i.e., product, robot, or shelf), and the RFID code is assumed to be the element identifier.

Concerning the BPS the following assumptions are made:

- The simulator mimics the workflow of a distributed system that combines information systems and sensors which interact together to process a case order.
- Each distributed system entity (i.e., clerk, robot, shelf) has a distinct process instance identifier.

- The events happening within the distributed system are ordered according to a distributed algorithm responsible for synchronizing the system's logical clocks.
- Products are generated and distributed across the shelves randomly.
- Shelves are assigned to robots randomly.
- Many process instances (cases) are allowed to run concurrently.
- Cases are processed in a FIFO order.
- Each business process activity has a minimum service time, a maximum service time, an average, and a mode (most likely service time). It is assumed that this information is provided by domain experts.
- All sensors are assumed to respond instantly.

3.4.2 Configuration Parameters

To ensure a high flexibility to the BPS, a set of configuration parameters provided as input to the simulator is defined. Although the control-flow is restricted to the use case scenario, it is possible to play-out the simulation with different settings. This feature is important to evaluate the algorithms presented in Chapters 4, and 5. The configuration parameters are divided into the following three categories:

3.4.2.1 Use Case Parameters

The use case parameters are used to determine the number of products, shelves, and robots to be considered for each simulation run. The following parameters are considered:

- **Number of cases** is the number of process instances generated per simulation run.
- **Number of shelves** is the number of shelves to be considered per simulation run.
- **Products per shelf** is the maximum product capacity per shelf.
- **Number of robots** is the number of robots to be considered per simulation run.
- **Shake probability** is the probability that a shelf will be shaken while being moved by a robot.

3.4.2.2 Randomness and Queuing parameters

The randomness parameters are used to leverage randomness in the BPS. In this implementation the following randomness parameters are considered:

- **Service Time parameters** are used to generate random activity service times. For each activity, one should provide the minimum duration, the maximum duration, the average duration, and the mode (most likely duration). Using these data a random estimation of the service time is generated using a beta distribution function as described in Section 3.3.2.
- **Arrival rate** is the average number of cases arriving per time unit.
- **Service rate** is the average number of cases to be processed per time unit.

- **Balking rate** is the average number of cases that suddenly terminate before being completed.

3.4.3 Log Files Generation

For the sake of evaluating the algorithms presented in Chapters 4, and 5, it is necessary to generate a set of log files recording the interactions between the different entities of the BPS. Furthermore, the generated files should comply with a standard event log format. A well-known event log standard is the XES[4] format designed by the IEEE Task Force on Process mining. As described in Section 2.2.1 (cf. Chapter 2), an event log is a collection of cases, where each case comprises a sequence of events, and each event is composed of a set attributes.

IS event log files record the events executed by the 3 sub-processes depicted in the BPMN model in Figure 1.2 (cf. Chapter 1). Examples of the generated IS event log files are shown in Figure 3.3.

The generation of IS log files is done during the simulation run. At the beginning of the run the log files are created and populated with the corresponding event log scheme (header). Then as process instances run, the executed events are logged to their corresponding event log file. For sake of simplicity, all IS event log files have similar event log scheme. As shown in Figure 3.3, they include *Case Id* referring to the process instance identifier, a unique *Event Id*, a *Start Timestamp*, an *End Timestamp*, an *Event Name*, an *Event Type*, a *Subject Group*, a *Subject Id*, an *Object Group*, and an *Object Id*. The four last attributes correspond to the resources responsible for the event execution. The subject and object attributes can be seen as a metaphor of *"who did what ?"*.

The events recorded by the BPS in the *Clerk Process Log* (Figure 3.3a) are: *Request Product*, *Assign Robot*, *Check if product was shacked*, and *Product Delivered*. The events recorded by the BPS in the *Robot Process Log* (Figure 3.3b) are: *Putdown Current Shelf*, *Go to appropriate shelf*, *Move shelf to dock*, *Reduce Speed*, and *Move shelf from dock*.

[4]See `http://www.xes-standard.org`.

Case Id	Start Timestamp	End Timesstamp	Event Name	Event Type	Subjec Group	Subject Id	Object Group	Object Id	Event Id
0CP	2017-12-25 01:09:50	2017-12-25 01:24:50	Request product	process activity	clerk	1c	product	0p	1313
0CP	2017-12-25 01:24:50	2017-12-25 01:24:50	Assgin robot	process activity	robot	3r	shelf	4s	1321
0CP	2017-12-25 02:00:50	2017-12-25 02:00:50	check if the product was shaked	process activity	clerk	1c	product	0p	1357
0CP	2017-12-25 02:14:50	2017-12-25 02:14:50	Product Delivered	process activity	clerk	1c	product	0p	1370
16CP	2017-12-25 01:29:50	2017-12-25 01:37:50	Request product	process activity	clerk	1c	product	16p	1326
16CP	2017-12-25 01:38:50	2017-12-25 01:38:50	Assgin robot	process activity	robot	9r	shelf	2s	1334
16CP	2017-12-25 05:28:50	2017-12-25 05:28:50	check if the product was shaked	process activity	clerk	1c	product	16p	1530
16CP	2017-12-25 05:36:50	2017-12-25 05:36:50	Product Delivered	process activity	clerk	1c	product	16p	1538
19CP	2017-12-25 01:25:50	2017-12-25 01:31:50	Request product	process activity	clerk	1c	product	19p	1322
19CP	2017-12-25 01:32:50	2017-12-25 01:32:50	Assgin robot	process activity	robot	7r	shelf	2s	1329
19CP	2017-12-25 04:02:50	2017-12-25 04:02:50	check if the product was shaked	process activity	clerk	1c	product	19p	1480
19CP	2017-12-25 04:25:50	2017-12-25 04:25:50	Product Delivered	process activity	clerk	1c	product	19p	1492

(a) Fraction of Clerk Process Log

Case Id	Start Timestamp	End Timesstamp	Event Name	Event Type	Subjec Group	Subject Id	Object Group	Object Id	Event Id
0RP	2017-12-25 01:25:50	2017-12-25 01:29:50	putdown current shelf	process activity	robot	3r	shelf	11s	1323
0RP	2017-12-25 01:29:50	2017-12-25 01:33:50	go to approriate shelf	process activity	robot	3r	shelf	4s	1325
0RP	2017-12-25 01:35:50	2017-12-25 01:42:50	move shelf to dock	process activity	robot	3r	shelf	4s	1331
0RP	2017-12-25 01:49:50	2017-12-25 01:53:50	move shelf from dock	process activity	robot	3r	shelf	4s	1352
16RP	2017-12-25 03:35:50	2017-12-25 03:40:50	putdown current shelf	process activity	robot	9r	shelf	45s	1459
16RP	2017-12-25 03:40:50	2017-12-25 03:54:50	go to approriate shelf	process activity	robot	9r	shelf	2s	1467
16RP	2017-12-25 03:55:50	2017-12-25 04:08:50	move shelf to dock	process activity	robot	9r	shelf	2s	1478
16RP	2017-12-25 04:14:50	2017-12-25 04:25:50	move shelf from dock	process activity	robot	9r	shelf	2s	1491
19RP	2017-12-25 02:15:50	2017-12-25 02:21:50	putdown current shelf	process activity	robot	7r	shelf	56s	1372
19RP	2017-12-25 02:21:50	2017-12-25 02:35:50	go to approriate shelf	process activity	robot	7r	shelf	2s	1383
19RP	2017-12-25 02:50:50	2017-12-25 03:07:50	reduce speed	process activity	robot	7r	shelf	2s	1410
19RP	2017-12-25 02:37:50	2017-12-25 03:07:50	move shelf to dock	process activity	robot	7r	shelf	2s	1397

(b) Fraction of Robot Process Log

Figure 3.3: Examples of IS event log files.

Sensor log files are used to record the progress of the manual activities and the message passing between the 3 sub-process shown in the BPMN model Figure 1.2 (Chapter 1). Examples of generated sensor log files are depicted in Figure 3.4. From an implementation perspective, sensor data is recorded the same way as normal events, however, the log scheme used is different depending on the sensor type (active or decorative). As shown in Figures 3.4a, and 3.4b, in the RFID sensor logs, the log scheme comprises, *Chip Name*, *Timestamp*, a scanned *Item Id*, and *Entry Id*. Concerning the sensor logs referring to the accelerometer data depicted in Figure 3.4c, the log scheme contains *Shelf Id*, *Entry Id*, and a *Timestamp* referring to the moment in time when a shake was recorded.

The sensor data recorded by the BPS in the *Clerk RFID log* (Figure 3.4a) are *Collector* referring to the manual process activity *Collect product from shelf*; *Extra check* referring to the manual process activity *Do extra check*; and *PackageDone* referring to the manual process activity *Package Product*. The sensor data recorded by the BPS in the *Robot RFID log* (Figure 3.4b) are *ShelfMoved* referring to the message passing *Send shelf being moved to shelf - Receive shelf being moved from robot* between *Clerk process* and *Robot process*; *ShelfDelivered* referring to the message passing *Send shelf delivered - Receive shelf delivered/response from robot* between *Robot process* and *Clerk process*, and *Robot process* and *Smart Shelf process*; *DisposeShelf* referring to the message passing *Send dispose shelf - Receive dispose shelf from clerk* between *Clerk process* and *Robot process*.

Concerning the data recorded by the BPS in the *Accelerometer log* (Figure 3.4c), the accelorometer coordinates' values are abstracted, and only the timestamp when a shelf was shaken is recorded.

Chip	Timestamp	Item Id	Entry Id
collector	2017-12-25 01:39:50	60p	1333
collector	2017-12-25 01:48:50	0p	1347
packageDone	2017-12-25 01:59:50	60p	1345
collector	2017-12-25 02:11:50	1p	1367
packageDone	2017-12-25 02:13:50	0p	1358
collector	2017-12-25 02:18:50	50p	1376
extraCheck	2017-12-25 02:27:50	1p	1378
packageDone	2017-12-25 02:32:50	1p	1388
collector	2017-12-25 02:37:50	37p	1392
extraCheck	2017-12-25 02:43:50	50p	1395
packageDone	2017-12-25 02:54:50	50p	1407
collector	2017-12-25 03:00:50	69p	1421

(a) Fraction of Clerk RFID log

Chip	Timestamp	Item Id	Entry Id
shelfMoved	2017-12-25 01:31:50	4r	1327
shelfMoved	2017-12-25 01:34:50	3r	1330
shelfDelivered	2017-12-25 01:36:50	4r	1332
shelfMoved	2017-12-25 01:39:50	5r	1335
disposeShelf	2017-12-25 01:41:50	4r	1338
shelfDelivered	2017-12-25 01:44:50	3r	1342
shelfMoved	2017-12-25 01:48:50	8r	1348
disposeShelf	2017-12-25 01:49:50	3r	1350
shelfDelivered	2017-12-25 02:09:50	5r	1364
disposeShelf	2017-12-25 02:13:50	5r	1369
shelfDelivered	2017-12-25 02:14:50	8r	1371
disposeShelf	2017-12-25 02:19:50	8r	1379

(b) Fraction of Robot RFID log

Shelf Id	Entry Id	Timestamp
4s	1344	2017-12-25 01:45:50
2s	1349	2017-12-25 01:48:50
1s	1355	2017-12-25 01:56:50
2s	1404	2017-12-25 02:40:50
0s	1405	2017-12-25 02:41:50
3s	1435	2017-12-25 03:10:50
1s	1443	2017-12-25 03:14:50
0s	1455	2017-12-25 03:24:50
1s	1471	2017-12-25 03:43:50
2s	1511	2017-12-25 05:03:50
3s	1521	2017-12-25 05:13:50
4s	1542	2017-12-25 05:38:50

(c) Fraction of Accelerometer log

Figure 3.4: Examples of sensor log files.

3.4.4 Multi-Threading

The BPS uses multi-threading to mimic the workflow of the use case scenario, hence, the process instances are handled using a group of threads that interact together and synchronize in order to perform a set of tasks. As described in Section 3.3.1, It is important to distinguished between transient entities and resident entities. In this context, all the resident entities such as the clerk, the robot, and the shelf are implemented as threads.

The execution of a process instance requires the interaction of different threads. Therefore, a wait/notify mechanism is implemented, which comprises a set of objects used to enable message passing and synchronization between threads. Each object is replicated in a Hashmap structure and

can only be accessed using a unique key, that is granted to the corresponding process instance. This approach allows avoiding synchronization conflicts between different process instances. Another important requirement to consider while dealing with multi-threading is to cope with deadlocks and starvation. These two common issues are usually caused by the lack of priority handling mechanisms. For this purpose, a thread pooling mechanism using a linked blocking queue[5] is implemented to order thread requests in a FIFO order.

Threads are also used to simulate the continuous data stream provided by sensors. For instance, the accelerometer sensors are implemented as independent threads that report a continuous stream allowing to notify shakes when a shelf is moved.

The system's artificial clock is also implemented as a thread that is initiated at the beginning of the simulation run with a pre-defined start time value. In order to optimize the simulation runtime, the activity service times are reduced to the order of milliseconds. However, to obtain realistic log files, it is mandatory to report timestamps in the normal order (minutes and hours). In this context, the artificial clock thread is used as a logging time reference with reduced clock granularity.

3.5 Conclusion

This chapter investigates the design and implementation characteristics of the BPS which uses a dynamic, stochastic, and continuous simulation model. Although the simulator is meant only to simulate a specific use case scenario, it provides several simulation characteristics such as queuing and randomness allowing for several settings.

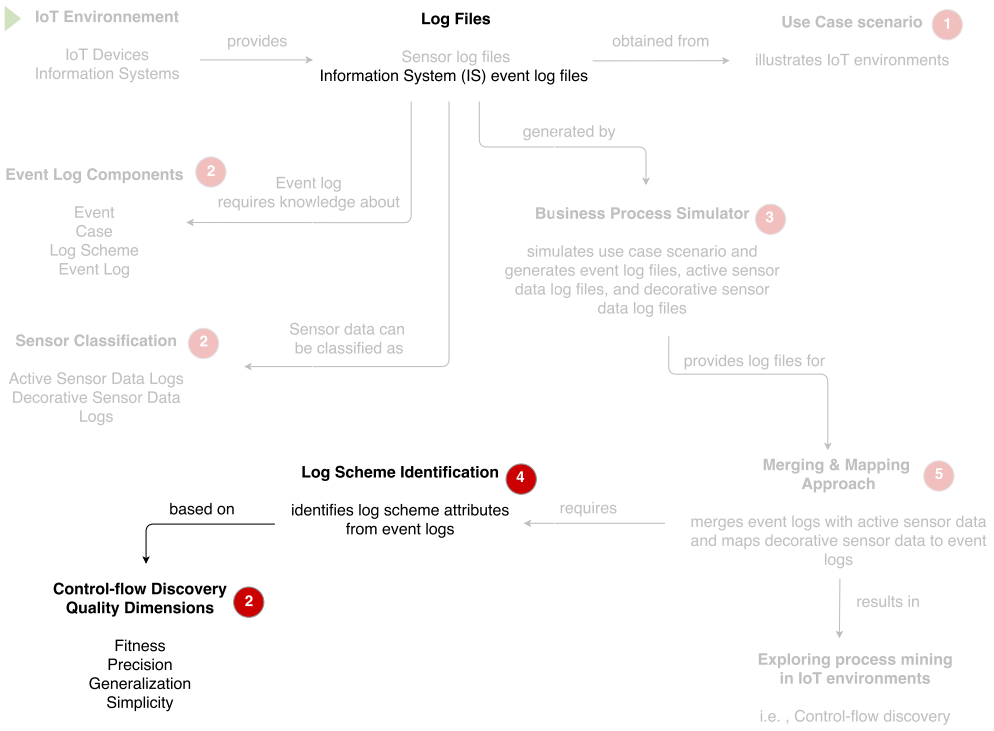
Moreover, a set of configuration parameters is proposed to ensure a prominent flexibility. The implementation of the BPS was conducted in Java. It consists of two main components that are the use case scenario component which implements the functional requirements of our use case, and the BPS component which implements the required simulation mechanisms.

Since the main goal of this simulator is to generate synthetic log files to evaluate the algorithms presented in Chapters 4, and 5. A set of assumptions are made to simplify the implementation and orient it to the desired purpose. The generated log files record the business process work-flow as well as the sensor data captured in real time. The next step is to merge those log files to explore the process mining capabilities.

[5]see `https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/LinkedBlockingQueue.html`

Chapter 4

Event Log Scheme Identification



This chapter aims at solving the event log scheme identification problem by introducing a new approach to label log files automatically. Section 4.1 introduces the problem and addresses its challenges. Section 4.2 discusses the related work. Section 4.3 presents the event log scheme identification approach along with the concepts used to explain it. Section 4.4 evaluates the

proposed approach, and finally Section 4.5 discusses the obtained results.

4.1 Introduction

The event logs produced by information systems provide insights about the executed process instances and allow to perceive the individual behaviour of each process instance. In process mining, the individual behaviour is implied from the recorded control-flow that represents the order in which events were executed, and the data-flow that represents the correlation between events' attributes. However, to explore process mining capabilities to mine the individual behaviour of each process instance from an event log, it is necessary to distinguish between the recorded process instances. This requirement gets more complicated in case of concurrent execution of process instances which is, by the way, one of the fundamental principles of designing modern BPM systems [43]. Consequently, the relation between events becomes uncertain, as two successive events in the log may belong to different process instances. As solution, a (*Case Id*) should be attributed to each single process execution.

The availability of case ids in an event log depends on its level of maturity, in fact, Van Der Aalst et al. in the process mining manifesto [40] proposed a maturity ranking of event logs depending on the level of logging information they provide. Nowadays, some process-oriented management systems (i.e., ERP systems, CRM systems) provide mature event logs where the case ids are explicitly mentioned. However, other management systems such as document management systems (DMS) still lack the notion of case Id in their generated log files [12]. Therefore it is essential to design a generic approach to infer the case ids from immature event log files to correlate events belonging the same process instance.

This chapter addresses the challenge of inferring case ids as an initial stride toward a generic approach allowing to infer all the event log scheme attributes (i.e., activity name, resource). By exploring the control-flow discovery quality dimensions described in Section 2.3 (cf. Chapter 2) a new approach to infer the case ids from event logs (ICI) is introduced and evaluated using both synthetic and real-world event logs.

4.2 Background and Related work

The challenge of inferring case ids from event logs has obtained little attention from the BPM community. The reason is that most of the literature introducing new process mining techniques assume the existence of labelled log files where the case id is known beforehand. This thesis report discards this assumption, and go on a quest for automatically inferring case ids. By looking at the existing related work, it is notable that few publications [20], [47], [12], [8] have already raised this challenge and proposed different approaches from different perspectives to solve the problem.

Ferreira et al. in [20] proposed an approach to transform an unlabeled log into a labelled log using the Expectation-Maximization technique that aims at finding a solution that converges to a local maximum of the likelihood function. This approach is considered by the authors as generic and executable in different environments. However, inferring case ids using this technique might be subject to uncertainty due to its probabilistic nature. Moreover, the first-order Markovian model used is unable to represent some work-flow patterns such as loops and parallelism. An enhanced

approach was suggested by Walicki and Ferreira [47], which suggests a sequence partitioning technique. However, the proposed technique shares the same limitations as the previous one [20] since it is limited to only simple work-flow patterns, thus it does not support loops and parallelism.

Bayomie et al. in [8] proposed an approach to infer the case identifiers from unlabeled event logs. The approach requires the reference process model used to document the business process, which is often part of the documentation package delivered at design time. Afterwards, the reference process model is used to generate a causal behavioural profile [48], which is similar to the footprint tables introduced by the Alpha algorithm in [42, p. 130]. The later is used together with some time heuristics inferred from the event log to build a decision tree where each node represents an event from the event log and carries its conditional probability of belonging to the same case as its parent node. Bayomie's approach aims to generate a set of labelled event logs listed according to a ranking score used to indicate their degree of trust. The method considers the problem of inferring case ids from a different perspective since it assumes that none of the existing events' attribute could be used as a case id. Thus it explores the data-flow correlations (time heuristics) and control-flow correlations (causal behavioural profile) between events to group them by case. Whereas the ICI approach digs into a more specific situation, where the case Id is assumed to be implicitly present in the event log.

Burattin and Vigo in [12] share the same assumption as the ICI approach, by considering the case id as a hidden attribute inside the log. The authors justify their assumption by the fact that it is general enough for a broad range of real-world event logs. Their proposed approach consists of filtering a set of event attributes considered as candidates for representing the case id in an event log. The filtering is done to reduce the search space by applying some selection heuristics such as selecting only the attributes with specific data types (i.e., ignoring timestamps), and using regular expression constraints as a selection criterion. Afterwards, the approach exploits the amount of data shared between attributes to construct chains, such that each chain links all similar attributes' values across the log. The case ids are then, inferred by choosing the chain with maximal length and minimal number of crossing attributes. By reducing the search space, the approach aims at finding a set of possible combinations of attribute that might represent the case id. However, it relies entirely on the similarity of attributes; thus, its accuracy is limited to a specific range of event logs.

The ICI approach overcomes all the challenges of the previously cited approaches. Indeed, it does not require any reference process model nor heuristics to infer the case ids. The aim is to introduce a generic approach to infer the case id from any event log using the four quality metrics described in Section 2.3 (cf. Chapter 2).

4.3 Methodological Approach

The ICI approach intents to automate the event log scheme identification process. Section 4.3.1 provides a formal definition of the notations used to describe the approach, Section 4.3.2 highlights the underlying assumptions, Section 4.3.3 presents the approach, and Section 4.3.4 illustrates the ICI approach by providing a running example.

4.3.1 Preliminaries

The event Definition 2.2.2 provided in Chapter 2 assumes the existence of a case id $c \in \mathcal{C}$ in order to construct the tuple $e = (a, c, t_s, t_e, d_1, \ldots, d_m)$. Since the ICI approach assumes that the case id is unknown a-priori, it is necessary to define a *Raw Event* as an event with unknown case id.

Definition 4.3.1 (Raw Event)**.** A *Raw Event* is a tuple $re = (a, t_s, t_e, d_1, \ldots, d_m)$, where:

- $a \in \mathcal{A}$ represents the activity associated to the event;
- $t_s \in \mathbb{N}$ represents the start timestamp;
- $t_e \in \mathbb{N}$ represents the end timestamp;
- $d_1, \ldots, d_m$ is a list of raw event attributes, where $\forall\, 1 \leq i \leq m, d_i \in \mathcal{D}_i$, $\mathcal{D}_i$ being the set of all possible attributes.

$\mathcal{RE} = \mathcal{A} \times \mathbb{N} \times \mathbb{N} \times \mathcal{D}_1 \times \cdots \times \mathcal{D}_m$ is called the raw event universe. In a raw event re, the following projection functions are defined: $\pi_a(re) = a$, $\pi_{t_s}(re) = t_s$, $\pi_{t_e}(re) = t_e$, and $\pi_{d_i}(re) = d_i, \forall\, 1 \leq i \leq m$. If re does not contain the attribute value d_i for some $i \in [1, m] \subset \mathbb{N}$, $\pi_{d_i}(re) = \perp$.

According to Definition 2.2.5 (cf. Chapter 2), an event log is defined as a tuple $L = (S(\mathcal{E}), T)$ where $S(\mathcal{E})$ is an event log scheme and T is a set of traces. Given a set of *raw events* $\mathcal{ER}$, the research problem introduced in this chapter is to transform $\mathcal{ER}$ to a set of *events* $\mathcal{EN}$ to construct T and identify $S(\mathcal{E})$. For sake of simplicity, the research problem is reduced to the scale of identifying only the case id "*caseid*" $\in S(\mathcal{E})$ as an initial stride toward a generic approach allowing to automatically identify other event log scheme attributes.

4.3.2 Assumptions

The ICI approach is built upon the following assumptions:

- The case id is assumed to be implicitly mentioned in the event log. In other words, given a raw event $re = (a, t_s, t_e, d_1, \ldots, d_m)$ (Definition 4.3.1), a case id c is one of the raw event attributes $d_1, \ldots, d_m$.
- The case id is given by the same attribute of all raw events. In other words, if d_i is the case id attribute of raw event $re \in \mathcal{RE}$, then the case id attribute of all other events in $\mathcal{RE}$ is d_i.
- The event name attribute and the timestamp attribute of the event log are known apriori.
- Each event log file is assumed to record the control-flow of only one process model.

4.3.3 Approach

The preliminaries presented in Section 4.3.1 and the assumptions presented in Section 4.3.2 provide a good starting point to describe the ICI approach. In this Section, it is important to emphasis on the use of the four quality dimensions described in Section 2.3 (cf. Chapter 2) to infer the case id. The control-flow discovery allows discovering process models reflecting the behaviours seen in an event log [42, p. 125]. To ensure the correctness of the discovered model, one needs to identify three main important attributes among the other event attributes, namely the *case id*, the *activity name*, and the *timestamp*, which are important to define the control-flow of the discovered

process model. In case one of these attributes is wrongly identified, the obtained model would be inconsistent. For instance, by selecting a random event attribute as a case id it is most likely that the discovered control-flow would not represent the original process model. Consequently, the discovery algorithm used will produce some strange behaviours resulting in an inconsistent model. Luckily, the control-flow four quality dimensions allow quantifying those behaviours.

In principle, the control-flow discovery quality dimensions are meant to evaluate and compare the correctness of different discovery algorithms [9]. This chapter aims at going beyond the classical use of quality dimensions by exploiting their ability to evaluate and compare different process models obtained using the same process discovery algorithm but considering different event attributes as case id.

To quantify the four quality dimensions of a process model, the ICI approach relies on the metrics presented in Section 2.3 (cf. Chapter 2). The fitness score is calculated using the *Alignment-based Replay Fitness* technique which quantifies the extent to which the event log traces can be replayed on the discovered model (ref. Section 2.3.1). The precision score is calculated using the *Escaping edges* technique which estimates the behaviours allowed by a process model (ref. Section 2.3.2). The generalization score is calculated using the *Frequency of use* technique that is built on the assumption that a generic model is a model whose parts are all used with the same frequency (ref. Section 2.3.3). The simplicity score is calculated using the *Simplicity by activity occurrence* approach which quantifies the simplicity of a process model by the number of duplicate activities it contains (ref. Section 2.3.4).

The ICI approach includes the following steps: (a) Compute grouping ratio for each attribute. (b) Compute the quality score for each attribute.

4.3.3.1 Compute Grouping Ratio for Each Attribute

To get close insight about which event attribute is more likely to represent the *case id*, the ICI approach computes a grouping ratio for each attribute. It is used to quantify the extent to which an attribute can be used to split events into groups, such that each group can be identified by a unique value of the attribute. For instance, a case id attribute is used to group events belonging to the same process execution by assigning them similar case id value, thus by grouping events by case id, the obtained number of groups will correspond to the number of process executions (cases). However, by choosing a different attribute to represent the case id (i.e., timestamp, event id, resource), the obtained event groups might have smaller or larger size.

Let $\mathcal{ER}$ be a set of raw events. According to Definition 4.3.1, a raw event re is a tuple $re = (a, t_s, t_e, d_1, \ldots, d_m)$, with $d_1, \ldots, d_m$ being the list of raw event attributes, where $\forall\, 1 \leq i \leq m, d_i \in \mathcal{D}_i$. $\mathcal{N}_{d_i}$ is defined as the set of unique values for attribute d_i such that $\mathcal{N}_{d_i} = \{\pi_{d_i}(re) \mid re \in ER\}$. Then, the *Grouping Ratio* for attribute d_i is defined as $g_i = 1 - \frac{|\mathcal{N}_{d_i}|}{|\mathcal{ER}|}$ such that $|\mathcal{N}_{d_i}|$ is the size of set $\mathcal{N}_{d_i}$, and $|\mathcal{ER}|$ is the size of the set $\mathcal{ER}$ that is the total number of raw events it contains.

4.3.3.2 Compute Quality Score for Each Attribute

In this step *raw events* are transformed into *events* with known case id that is one of the event attributes, and then an event log L is generated. Afterwards, a process discovery algorithm can be applied to the event log L to discover the corresponding process model. Once the model is obtained, the quality dimension metrics are used to measure fitness, precision, generalization, and simplicity.

The measurements are summed up together with the *the distance to the average grouping ratio* then averaged to get a *quality score*, which is used to rank the process model corresponding to each candidate attribute. Finally, the candidate attribute with the highest rank is selected to be the real case id attribute.

Let $\mathcal{W}$ be the set of candidate raw event attributes, and $\mathcal{ER}$ be a set of raw events. $\mathcal{ER}$ can be transformed into a set of events $\mathcal{EN}$ as follows: for each candidate attribute $w \in \mathcal{W}$, a raw event $re \in \mathcal{ER}$ is transformed to an event $e \in \mathcal{EN}$ such that $\pi_a(e) = \pi_a(re)$, $\pi_{t_s}(e) = \pi_{t_s}(re)$, $\pi_{t_e}(e) = \pi_{t_e}(re)$, $\pi_{d_i}(e) = \pi_{d_i}(re), \forall\, 1 \leq i \leq m$ (m the number of additional attributes), and $\pi_c(e) = w$. After creating the set of events $\mathcal{EN}$, an event log can be generated using definition 2.2.5 (cf. Chapter 2).

Let $R_{\mathcal{W}}$ be the average grouping ratio of the candidate attributes in $\mathcal{W}$, and R_i the grouping ratio of a candidate attribute $w \in \mathcal{W}$. Then the distance to the average grouping ratio for a candidate attributes $w \in \mathcal{W}$ is defined as $DTAG = 1 - Abs(R_i - R_{\mathcal{W}})$ where $Abs()$ is an absolute value function.

Algorithm 1 describes the *inferCaseId()* function used to infer the case id. The function takes as input $\mathcal{ER}$ the set of raw events, and $\mathcal{W}$ the set the candidate event attributes, and returns the real case id attribute $c \in \mathcal{W}$. The algorithm uses the following functions:

- $generateLog(\mathcal{ER}, w)$: This function transforms the set of raw events $\mathcal{ER}$ to a set of events $\mathcal{EN}$ where the case id $c = w \in \mathcal{W}$, then generate an event log Log using Definition 2.2.5 (cf. Chapter 2).
- $DoMiner(Log)$: This function takes as input the event log, and apply a process discovery algorithm (i.e., Inductive Miner) to generate a process model.
- $DTAGRatio(\mathcal{W}, w)$: This function computes the distance to average grouping ratio for a candidate attribute $w \in \mathcal{W}$.
- $ComputeFitness(Model)$, $ComputePrecision(Model)$, $ComputeGeneralization(Model)$, and $ComputeSimplicity(Model)$: These functions are used to compute the fitness score, the precision score, the generalization score, and simplicity score for a process model respectively.

Besides the algorithm defines G as a mapping from the set of candidate events $\mathcal{W}$ to the set of real numbers $\mathbb{R}$ such that each $w \in \mathcal{W}$ is mapped to a real number $r \in \mathbb{R}$ representing its quality score.

Algorithm 1: Infer Case id

```
1  Function inferCaseId (ER, W)
   Input  : ER the set of raw events, W the set the candidate attributes
   Output: real case id attribute c
2  begin
3      highestScore← 0                                  // initialize highest score
4      c ← null                                         // initialize real case id
5      G ← null                                         // initialize map G
6      /* Iterate over all candidate attributes in W                              */
7      foreach w ∈ W do
8          Log ← generateLog(ER,w)                      // generate log file
9          Model ← DoMiner(Log)                         // apply process discovery algorithm
10         Gr ← DTAGRatio(W,w)                          // compute the distance to average grouping ratio
11         Ft ← ComputeFitness(Model)                   // compute fitness score
12         Pr ← ComputePrecision(Model)                 // compute precision score
13         Ge ← ComputeGeneralization(Model)            // compute generalization score
14         Si ← ComputeSimplicity(Model)                // compute simplicity score
15         QualityScore ← (Gr + Ft + Pr + Ge + Si)/5    // compute quality score
16         G(w) ← QualityScore                          // Map the obtained quality score to w
17     /* Iterate over all the keys of Map G                                      */
18     foreach key w ∈ G do
19         if G(w) > highestScore then
20             highestScore ← G(w)                      // assign G(w) to highestScore
21             c ← w                                    // consider w as the real case id
22     return c
```

4.3.4 Running Example

To illustrate the ICI approach the *Robot Process* log file generated by the BPS is used. A small portion of the robot process log is depicted in Figure 3.3b (cf. Chapter 3). The log file used comprises 7250 events. Assuming that the case id column in the log file is unknown, the aim is to infer it using the ICI approach explained in Section 4.3.3.

The first step is to compute the grouping ratios using the formula shown in Section 4.3.3.1 used to calculate the grouping ratio of each event attribute (log column) in the log file. The results are shown in Table 4.1.

The grouping ratios presented in Table 4.1 provide a brief insight about which event attributes might represent the case id. By analyzing the obtained grouping ratios, one can notice the following: *Event Id* attribute has score 0, which is trivial since the event Id is a unique identifier for each event. *Start Timestamp* and *End Timestamp* have very low grouping ratios because some events happened concurrently. However, *Event Name*, *Event type*, *Subject Group*, and *Object Group* have very high grouping ratios. Assuming that a case id should group not too many and too few events one would expect that only *Case Id*, *Subject Id*, and *Object Id* might represent the real case Id column.

The second step is to compute the quality score for the candidate attributes in the $\mathcal{W}$. Hence, each

Event Attribute	Grouping Ratio
CaseId	**0.7694**
EventId	0.0000
StartT	0.0079
EndT	0.1696
Event Name	0.9993
Event Type	0.9999
Subject Group	0.9999
Subject Id	**0.9862**
Object Group	0.9999
Object Id	**0.9334**

Table 4.1: Grouping ratios for robot process log (cf. Figure 3.3b Chapter 3)

candidate attribute is considered as a case id. For the sake of simplicity it is assumed that the event name attribute and the start timestamp attribute are known; thus they are excluded from the set of candidate attributes $\mathcal{W}$. The corresponding process model is generated using a process discovery algorithm. In this example, the "Inductive Miner with Infrequent and all operators (IMfa)" [28] is used to discover the process model. IMfa is considered as one of the least bias discovery algorithms toward the four quality dimensions [28]. Finally, the distance to the average grouping ratio and the four quality dimension metrics explained in Section 2.3 (cf. Chapter 2) (Alignment-based Replay Fitness, Escaping edges, Frequency of use, Simplicity by activity occurrence) are computed for each candidate attribute and the quality score is derived as shown in Table 4.2.

Candidate Col.	Grouping Ratio	DTAG ratio	Fitness	Precision	Simplicity	Generalization	Quality Score	Rank
Case Id	0.7694	0.9172	0.9998	1.0000	1.0000	0.9726	**0.9779**	**1**
Event Id	0.0000	0.3135	1.0000	1.0000	1.0000	0.9747	0.8576	5
End Timesstamp	0.1697	0.4831	0.9543	0.9171	1.0000	0.9767	0.8663	4
Event Type	0.9999	0.6867	0.7975	0.4927	1.0000	0.8450	0.7644	6
Subject Group	0.9999	0.6867	0.7975	0.4927	1.0000	0.8450	0.7644	6
Subject Id	0.9862	0.7003	0.9997	0.8077	1.0000	0.9596	0.8935	2
Object Group	0.9999	0.6867	0.7975	0.4927	1.0000	0.8450	0.7644	6
Object Id	0.9334	0.7532	0.9991	0.7343	1.0000	0.9738	0.8921	3

Table 4.2: Quality score for each candidate attribute, calculated from the quality dimension metrics presented in Section 2.3 (cf. Chapter 2)

By ranking the obtained quality scores shown in Table 4.2, it is clear that the "Case Id" attribute represents the real case Id attribute in the event log because it has the highest quality score. This example demonstrates that the ICI approach provides accurate results and allows to infer the case id on a synthetic log generated by the BPS. The next section uses real-world event logs to evaluate the ICI approach.

4.4 Evaluation

To evaluate the ICI approach on a larger scale, several real-world event logs were obtained from the 4TU public database [1]. With the purpose of obtaining a ground truth to evaluate the accuracy of ICI approach, the event logs used are all labelled; thus the real case id attribute is known. This section reports the results obtained by applying the ICI approach on several real-world log files. Section 4.4.1 describes the implementation, Section 4.4.2 explains the evaluation procedure, Section 4.4.3 briefly describes the used data sets, and Section 4.4.4 presents the evaluation results.

4.4.1 Implementation

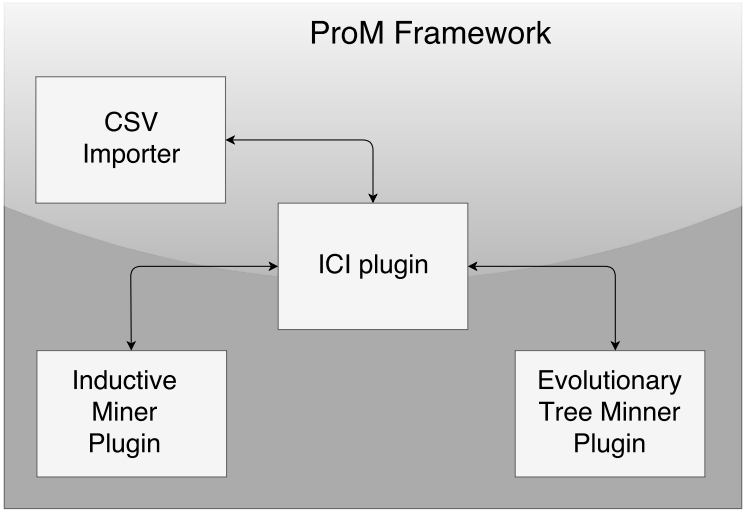


Figure 4.1: ICI plugin Architecture

The ICI plugin integrates two main packages from the open-source process mining framework ProM [2] that are the *Inductive Miner* [3] which implements the IMfa algorithm, and the *Evolutionary Tree Minner* [4] which implements the necessary metrics used to evaluate the control-flow four quality dimensions. The ICI plugin requires as input a log file in CSV format, and the index columns of the event name, and the timestamp attributes. The plugin generates a set of candidate case ids, and then executes the algorithm to infer the real case id attribute. An overview of the ICI plugin architecture is depicted in Figure 4.1.

The ICI implementation is available at the Github Directory `https://github.com/aminobest/ActiveAndDecorativeSensorMerging/tree/master/src/iciPlugin`.

[1] see the collection of real-world event logs at 4TU Center for Research Data `http://data.4tu.nl/repository/collection:event_logs_real`

[2] see `www.promtools.org/`

[3] see `https://svn.win.tue.nl/repos/prom/Packages/InductiveMiner/`

[4] see `https://svn.win.tue.nl/repos/prom/Packages/EvolutionaryTreeMiner/`

4.4.2 Evaluation Procedure

Most of the event logs obtained from the 4TU database are available in XES format, thus, the event log scheme is already known. A process mining tool named "Disco"[5] was used to convert the event logs from XES to CSV format. Also, to reduce the plugin execution time and avoid memory overheads only the first 10,000 events (ordered by timestamp) in the event logs were considered.

The procedure starts with iterating through each candidate attributes and computing its grouping ratio, before considering it as a case id. Then, by manually providing the event name and the timestamp columns index, the IMfa variant of the inductive miner is used to generate a process tree, that is used to compute the four quality metrics.

During the initial testing of the ICI plugin, a memory overhead was noticed causing the plugin to crash. The overhead is caused when attributes with very high or very low grouping ratios are considered as case ids. Consequently, the inductive miner consumes a huge amount of memory to mine the corresponding process model, which results in a memory overhead causing the whole plugin to crash. As solution, a light-weighted version of the plugin was developed to evaluate the ICI approach in a semi-automated way. This version returns the grouping ratio and the four quality metrics for an individual candidate attribute. Therefore, for each candidate attribute a plugin instance is wrapped into a job script and submitted to a high-performance computing machine (DTU HPC [6]). This technique allows preventing memory overheads from propagating and to try different benchmarks. After a set of trials, the following optimal system configuration was chosen: 12Gb of RAM, and 1 processor with 4 cores.

4.4.3 Data Sets

The data sets considered to evaluate the ICI approach are the following:

- BPI challenge 2012 event log [7] records a loan application process.
- BPI challenge 2013 event log [8] records Volvo IT incidents and management problems.
- BPI challenge 2014 event log [9] records the incidents encountered in one of the RaboBank systems.
- BPI challanges 2017 event log [10] records the loan application process of a Dutch finacial institute.
- Hospital billing event log [11] is obtained from the financial modules of an ERP system used by a regional hospital.
- Credit requirements event log [12] records information regarding a credit requirement process in a bank.
- Helpdesk anonymized event log [13] records the work-flow of a call center

[5] see `https://fluxicon.com/disco/`
[6] see `http://www.hpc.dtu.dk`
[7] See `http://data.4tu.nl/repository/uuid:3926db30-f712-4394-aebc-75976070e91f`
[8] See `http://data.4tu.nl/repository/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07`
[9] See `http://data.4tu.nl/repository/uuid:c3e5d162-0cfd-4bb0-bd82-af5268819c35`
[10] See `http://data.4tu.nl/repository/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b`
[11] See `http://data.4tu.nl/repository/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741`
[12] See `http://data.4tu.nl/repository/uuid:453e8ad1-4df0-4511-a916-93f46a37a1b5`
[13] See `https://data.mendeley.com/datasets/nm9xkzhpm4/1`

- Sepsis event log [14] records the treatment process of hospital patients with sepsis symptoms.
- Receipt phase of environmental permit event log [15] records the application process for an environmental permit.

Since only the first 10,000 events are considered for the evaluation, a shorter version of the real-world event log files is generated, and made available online [16].

4.4.4 Results

The evaluation results are reported in Tables 4.3, 4.4, and 4.5. For each log file event attribute the following ratios are computed: grouping ratio (GR), distance to average grouping ratio (DTAG), alignment-based replay fitness (Fr), precision using escaping edges improved technique (Pi), generalization using frequency of use technique (Gv), Simplicity by activity occurrence technique (Sm), and quality score. To demonstrate the accuracy of the ICI approach, the real case id attribute (considered as ground truth) should always have the highest quality score among the other attributes. Indeed, the evaluation results show that the case id attribute always has the highest quality score. The results are discussed in details in Section 4.5.

[14] See `http://data.4tu.nl/repository/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460`

[15] See `http://data.4tu.nl/repository/uuid:a07386a5-7be3-4367-9535-70bc9e77dbe6`

[16] See `https://github.com/aminobest/ActiveAndDecorativeSensorMerging/tree/master/ICI%20evaluation/eventsLogsWithTop1000events`.

Log file	Candidate Col.	GR	DTAG	Fr	Pi	Sm	Gv	Quality Score	Rank
BPI challenge 2012	Case ID	0.9565	0.8995	0.9097	0.7346	1.0000	0.9170	**0.8922**	**1**
	Activity	0.9964	0.8596	N/A	N/A	N/A	N/A	0.0000	6
	Resource	0.9948	0.8612	0.9918	0.2393	1.0000	0.8326	0.7850	5
	Complete Timestamp	0.0563	0.2003	N/A	N/A	N/A	N/A	0.0000	6
	(case) AMOUNT_REQ	0.9907	0.8653	0.9998	0.4074	1.0000	0.9076	0.8360	2
	concept:name	0.9976	0.8584	0.9109	0.6615	1.0000	0.5169	0.7895	3
	lifecycle:transition	0.9997	0.8563	0.9967	0.2737	1.0000	0.8184	0.7890	4
BPI challenge 2013	Case ID	0.9438	0.9655	0.9926	0.4574	1.0000	0.9678	**0.8766**	**1**
	Activity	0.9989	0.9104	N/A	N/A	N/A	N/A	0.0000	12
	Resource	0.9278	0.9815	0.9910	0.4358	1.0000	0.9640	0.8744	2
	Complete Timestamp	0.0181	0.1088	N/A	N/A	N/A	N/A	0.0000	12
	concept:name	0.9997	0.9096	0.7763	0.6627	1.0000	0.5665	0.7830	11
	impact	0.9997	0.9096	0.9588	0.6001	1.0000	0.8457	0.8628	3
	lifecycle:transition	0.9989	0.9104	1.0000	0.6683	1.0000	0.6604	0.8478	5
	org:group	0.9625	0.9468	0.9795	0.3874	1.0000	0.9345	0.8496	4
	org:role	0.9978	0.9115	0.9792	0.2891	1.0000	0.8772	0.8114	9
	organization country	0.9982	0.9111	0.9792	0.2889	1.0000	0.8621	0.8083	10
	organization involved	0.9977	0.9116	0.9792	0.2894	1.0000	0.8796	0.8120	8
	product	0.9801	0.9292	0.9788	0.3571	1.0000	0.9588	0.8448	6
	resource country	0.9975	0.9118	0.9792	0.2903	1.0000	0.8814	0.8125	7
BPI challenge 2014	Incident ID	0.9696	0.7900	0.9977	0.3541	1.0000	0.8922	**0.8068**	**1**
	DateStamp	0.3970	0.6374	N/A	N/A	N/A	N/A	0.0000	4
	IncidentActivity_Number	0.0000	0.2404	1.0000	1.0000	1.0000	0.7271	0.7935	2
	IncidentActivity_Type	0.9964	0.7632	N/A	N/A	N/A	N/A	0.0000	4
	Assignment Group	0.9891	0.7705	0.9861	0.2568	1.0000	0.8846	0.7796	3
	KMnumber	0.9902	0.7694	N/A	N/A	N/A	N/A	0.0000	4
	Interaction ID	0.9749	0.7847	N/A	N/A	N/A	N/A	0.0000	4
BPI challanges 2017	Case ID	0.9464	0.8905	0.9806	0.7591	1.0000	0.9370	**0.9134**	**1**
	Activity	0.9976	0.8393	N/A	N/A	N/A	N/A	0.0000	19
	Resource	0.9927	0.8442	0.9980	0.3487	1.0000	0.9305	0.8243	5
	Start Timestamp	0.0001	0.1632	N/A	N/A	N/A	N/A	0.0000	19
	Complete Timestamp	0.0001	0.1632	1.0000	0.9966	1.0000	0.9235	0.8167	7
	(case) ApplicationType	0.9998	0.8371	0.9818	0.2648	1.0000	0.9082	0.7984	10
	(case) LoanGoal	0.9987	0.8382	0.9932	0.3230	1.0000	0.9255	0.8160	8
	(case) RequestedAmount	0.9916	0.8453	0.9858	0.5070	1.0000	0.9489	0.8574	2
	Accepted	0.9974	0.8395	0.9917	0.2344	1.0000	0.8623	0.7856	16
	Action	0.9993	0.8376	0.9644	0.2690	1.0000	0.7672	0.7676	18
	CreditScore	0.9819	0.8550	0.9918	0.2428	1.0000	0.8967	0.7972	12
	EventID	0.0943	0.2574	1.0000	1.0000	1.0000	0.9135	0.8342	3
	EventOrigin	0.9026	0.9343	0.9383	0.3726	1.0000	0.8503	0.8191	6
	FirstWithdrawalAmount	0.9837	0.8532	0.9918	0.2446	1.0000	0.8967	0.7973	11
	MonthlyCost	0.9653	0.8716	0.9919	0.2500	1.0000	0.8975	0.8022	9
	NumberOfTerms	0.9861	0.8508	0.9918	0.2436	1.0000	0.8951	0.7962	14
	OfferID	0.9222	0.9147	0.9952	0.3271	1.0000	0.8964	0.8267	4
	OfferedAmount	0.9814	0.8555	0.9918	0.2431	1.0000	0.8955	0.7972	13
	Selected	0.9967	0.8402	0.9917	0.2344	1.0000	0.8623	0.7857	15
	lifecycle:transition	0.9995	0.8374	0.9991	0.1837	1.0000	0.9059	0.7852	17

Table 4.3: Quality score for each event log candidate attribute (Part 1)

Log file	Candidate Col.	GR	DTAG	Fr	Pi	Sm	Gv	Quality Score	Rank
Sepsis	Case ID	0.9313	0.9567	0.9064	0.4008	1.0000	0.9238	**0.8375**	**1**
	Activity	0.9984	0.9762	N/A	N/A	N/A	N/A	0.0000	31
	Complete Timestamp	0.3560	0.3814	N/A	N/A	N/A	N/A	0.0000	31
	Age	0.9984	0.9762	0.0000	0.0000	1.0000	0.0000	0.3952	7
	CRP	0.9662	0.9916	0.0000	0.0000	1.0000	0.0000	0.3983	3
	Diagnose	0.9883	0.9863	0.0000	0.0000	1.0000	0.0000	0.3973	5
	DiagnosticArtAstrup	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	DiagnosticBlood	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	DiagnosticECG	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	DiagnosticIC	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	DiagnosticLacticAcid	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	DiagnosticLiquor	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	DiagnosticOther	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	DiagnosticSputum	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	DiagnosticUrinaryCulture	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	DiagnosticUrinarySediment	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	DiagnosticXthorax	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	DisfuncOrg	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	Hypotensie	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	Hypoxie	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	InfectionSuspected	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	Infusion	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	LacticAcid	0.9923	0.9823	0.0000	0.0000	1.0000	0.0000	0.3965	6
	Leucocytes	0.9659	0.9913	0.0000	0.0000	1.0000	0.0000	0.3983	4
	Oligurie	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	SIRSCritHeartRate	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	SIRSCritLeucos	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	SIRSCritTachypnea	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	SIRSCritTemperature	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	SIRSCriteria2OrMore	0.9997	0.9749	0.0000	0.0000	1.0000	0.0000	0.3950	8
	lifecycle:transition	0.9999	0.9747	0.0000	0.0000	1.0000	0.0000	0.3949	30
	org:group	0.9974	0.9772	0.9806	0.4605	1.0000	0.7337	0.8304	2
Hospital billing	Case ID	0.7968	0.8497	0.9544	0.9233	1.0000	0.8196	**0.9094**	**1**
	Activity	0.9997	0.9473	N/A	N/A	N/A	N/A	0.0000	19
	Resource	0.9907	0.9564	0.9942	0.5711	1.0000	0.9082	0.8860	2
	Complete Timestamp	0.0093	0.0623	N/A	N/A	N/A	N/A	0.0000	19
	actOrange	1.0000	0.9471	0.8370	0.4489	1.0000	0.6616	0.7789	11
	actRed	1.0000	0.9471	0.8370	0.4489	1.0000	0.6616	0.7789	11
	blocked	1.0000	0.9471	0.9805	0.4262	1.0000	0.6615	0.8031	7
	caseType	0.9999	0.9472	0.0000	0.0000	1.0000	0.0000	0.3894	17
	closeCode	0.9996	0.9475	0.9362	0.4777	1.0000	0.8035	0.8330	4
	diagnosis	0.9879	0.9592	0.0000	0.0000	1.0000	0.0000	0.3918	16
	flagA	1.0000	0.9471	0.9805	0.4262	1.0000	0.6615	0.8031	7
	flagB	1.0000	0.9471	0.9805	0.4262	1.0000	0.6615	0.8031	7
	flagC	1.0000	0.9471	0.8370	0.4489	1.0000	0.6616	0.7789	11
	flagD	1.0000	0.9471	0.0000	0.0000	1.0000	0.0000	0.3894	18
	isCancelled	1.0000	0.9471	0.9686	0.5058	1.0000	0.6711	0.8185	5
	isClosed	1.0000	0.9471	0.9809	0.4396	1.0000	0.7038	0.8143	6
	lifecycle:transition	1.0000	0.9471	N/A	N/A	N/A	N/A	0.0000	19
	msgCode	0.9999	0.9471	N/A	N/A	N/A	N/A	0.0000	19
	msgCount	0.9999	0.9472	0.8371	0.4489	1.0000	0.5749	0.7616	15
	msgType	1.0000	0.9471	N/A	N/A	N/A	N/A	0.0000	19
	speciality	0.9996	0.9474	0.9805	0.4316	1.0000	0.8185	0.8356	3
	state	0.9998	0.9472	0.9812	0.5063	1.0000	0.5779	0.8025	10
	version	0.9999	0.9472	0.8370	0.4489	1.0000	0.6065	0.7679	14

Table 4.4: Quality score for each event log candidate attribute (Part 2)

Log file	Candidate Col.	GR	DTAG	Fr	Pi	Sm	Gv	Quality Score	Rank
Credit requirements	Case ID	0.8750	0.7289	0.9916	1.0000	1.0000	0.9718	**0.9384**	**1**
	Activity	0.9992	0.6047	N/A	N/A	N/A	N/A	0.0000	4
	Resource	0.9993	0.6046	0.9442	0.6677	1.0000	0.4905	0.7414	3
	Start Timestamp	0.1339	0.5300	N/A	N/A	N/A	N/A	0.0000	4
	Complete Timestamp	0.0120	0.4081	0.9983	0.8900	1.0000	0.9821	0.8557	2
Helpdesk anonymized	Case ID	0.8167	0.8859	0.9643	0.9729	1.0000	0.9494	**0.9545**	**1**
	Activity	0.9991	0.9317	N/A	N/A	N/A	N/A	0.0000	14
	Resource	0.9978	0.9330	0.9988	0.1464	1.0000	0.7702	0.7697	9
	Complete Timestamp	0.1891	0.2583	N/A	N/A	N/A	N/A	0.0000	14
	concept:name	0.9991	0.9317	1.0000	0.6680	1.0000	0.4680	0.8136	8
	customer	0.9686	0.9622	0.9302	0.5364	1.0000	0.8786	0.8615	2
	lifecycle:transition	0.9998	0.9310	0.0000	0.0000	1.0000	0.0000	0.3862	13
	org:resource	0.9978	0.9330	0.9988	0.1464	1.0000	0.7702	0.7697	9
	product	0.9979	0.9329	0.9401	0.4979	1.0000	0.8761	0.8494	3
	responsible_section	0.9993	0.9315	0.9439	0.3858	1.0000	0.8335	0.8189	6
	seriousness	0.9995	0.9313	0.0000	0.0000	1.0000	0.0000	0.3863	11
	service_level	0.9995	0.9313	0.0000	0.0000	1.0000	0.0000	0.3863	11
	service_type	0.9996	0.9312	0.9420	0.4402	1.0000	0.8291	0.8285	4
	support_section	0.9993	0.9315	0.9436	0.3855	1.0000	0.8335	0.8188	7
	workgroup	0.9995	0.9313	0.8955	0.4410	1.0000	0.8353	0.8206	5
Receipt env. Permit	Case ID	0.8328	0.9850	0.9843	0.7017	1.0000	0.8838	**0.9110**	**1**
	Activity	0.9969	0.8209	N/A	N/A	N/A	N/A	0.0000	10
	Resource	0.9944	0.8234	1.0000	0.0717	1.0000	0.7770	0.7344	6
	Complete Timestamp	0.0000	0.1822	N/A	N/A	N/A	N/A	0.0000	10
	(case) channel	0.9994	0.8184	0.0000	0.0000	1.0000	0.0000	0.3637	8
	(case) department	0.9997	0.8181	N/A	N/A	N/A	N/A	0.0000	10
	(case) group	0.9991	0.8187	0.0000	0.0000	1.0000	0.0000	0.3637	7
	(case) responsible	0.9955	0.8223	0.9967	0.2582	1.0000	0.8714	0.7897	2
	concept:instance	0.0000	0.1822	1.0000	1.0000	1.0000	0.7638	0.7892	4
	concept:name	0.9969	0.8209	0.9999	0.6836	1.0000	0.3957	0.7800	5
	lifecycle:transition	0.9999	0.8179	0.0000	0.0000	1.0000	0.0000	0.3636	9
	org:group	0.9988	0.8189	0.9795	0.3507	1.0000	0.7992	0.7897	3

Table 4.5: Quality score for each event log candidate attribute (Part 3)

The material used to evaluate the ICI approach is available at the Github Directory `https://github.com/aminobest/ActiveAndDecorativeSensorMerging/tree/master/ICI%20evaluation`.

4.5 Discussion

This section discusses the evaluation results of the event logs shown in Tables 4.3, 4.4, 4.5. Clearly, the ICI approach demonstrates a high accuracy in inferring the case id in all the event logs considered for the evaluation. However, it is still important to highlight few cases where the quality score of other event attributes is very close to the case id attribute score. For instance, in BPI challenge 2013, the quality scores for *Case ID* attribute and *Resource* attribute are 0.8766 and 0.8744 respectively (less than 0.01 difference). To explain this inconsistency in the quality scores, the process mining tool Disco was used. By inspecting the statistics provided by the tool for the process model where the case id corresponds the real case id attribute, the number of cases and resources are 954 and 776 respectively, which can also be noticed from the grouping ratios of *Case ID* attribute and *Resource* attribute that are 0.9438 and 0.9278 respectively. This small difference in the quality score can be explained with the fact that the initial cut applied to the event log considers only the top 10,000 events which is not enough to perceive the overall behaviour of the

model. Alternatively, a cut of 40,000 events is applied to the BPI challenge 2013 event log, the corresponding quality scores are shown in Table 4.6.

Log file	Candidate Col.	GR	DTAG	Fr	Pi	Sm	Gv	Quality Score	Rank
BPI Chal. 2013 40,000 events	Case ID	0.9438	0.9581	0.9641	0.8205	1.0000	0.9480	**0.9381**	**1**
	Activity	0.9989	0.9030	N/A	N/A	N/A	N/A	0.0000	6
	Resource	0.9278	0.9741	0.9001	0.4056	1.0000	0.8938	**0.8347**	2
	Complete Timestamp	0.0181	0.1162	N/A	N/A	N/A	N/A	0.0000	6
	concept:name	0.9997	0.9022	0.0000	0.0000	1.0000	0.0000	0.3804	5
	impact	0.9997	0.9022	N/A	N/A	N/A	N/A	0.0000	6
	lifecycle:transition	0.9989	0.9030	N/A	N/A	N/A	N/A	0.0000	6
	org:group	0.9625	0.9394	1.0000	0.2431	1.0000	0.9354	0.8236	4
	org:role	0.9978	0.9041	N/A	N/A	N/A	N/A	0.0000	6
	organization country	0.9982	0.9037	N/A	N/A	N/A	N/A	0.0000	6
	organization involved	0.9977	0.9042	N/A	N/A	N/A	N/A	0.0000	6
	product	0.9801	0.9218	0.9996	0.2387	1.0000	0.9709	0.8262	3
	resource country	0.9975	0.9118	N/A	N/A	N/A	N/A	0.0000	6

Table 4.6: Quality scores for BPI challenge 2013 with a cut of 40,000 events

As shown in Table 4.6, by increasing the size of the cut from 10,000 events to 40,000 events, the difference between the quality scores for *Case ID* attribute and *Resource* attribute increased drastically (0.9381, and 0.8347 respectively). In a perfect scenario, one would avoid applying any cut to the event log to perceive the overall behaviour in the log, However, as explained in section 4.4.2, memory overhead issues are common especially while dealing with large event logs. Indeed, just by increasing the cut size to 40,000, the ICI plugin failed to compute the quality score for some candidate attributes (i.e., impact and organization country).

BPI challenge 2014 represents another example where the difference in quality scores between *Case ID* attribute and *IncidentActivityNumber* attribute is insignificant (0.8068, and 0.7935 respectively). However, the grouping ratio of *IncidentActivityNumber* attribute is 0; hence, the attribute contains only unique values. Consequently, the generated model would appear like a flower model where all the log traces can be replayed, thus, the fitness will certainly equal to 1. [42, p. 151]. To avoid such cases, the candidate attributes with a grouping ratio equals to 0 could be filtered out before running the ICI plugin.

In overall, the evaluation results demonstrate that the ICI approach can be potentially accurate on a wide range of event logs. However, the set of selected event logs is still considerably small, thus, the accuracy of the ICI approach cannot be generalized. Nevertheless, the results shown in Section 4.4.4 provide a clear insight into the aspects that should be enhanced. Mainly, the following two important challenges should be addressed: (a) Avoiding memory overhead by filtering out the candidate attributes with too low or too high grouping ratios. (b) Defining an optimal cut size proportional to each event log characteristics (i.e., number of resources).

4.6 Conclusion

To sum up this chapter, the ICI approach aims at proposing a new technique to automatically infer the case id from an event log by exploring the control-flow discovery quality dimensions capabilities. Unlike the existing techniques mentioned in Section 4.2, the ICI approach does not require any domain-specific heuristics. Under the assumption that a case id is implicitly mentioned in the

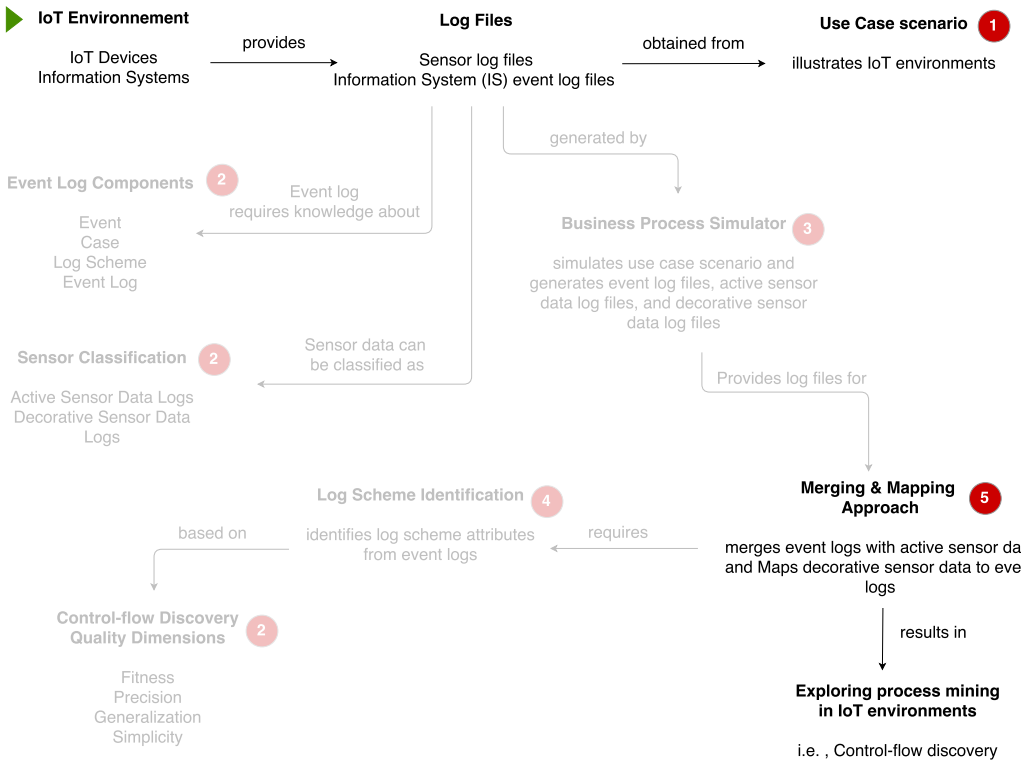
event log, the ICI approach allowed to correctly identify the case Id for the *Robot Process* event log generated by the BPS.

The approach was tested using several real-world event logs obtained from a public database to demonstrate its accuracy on a larger scale. The results show a high potential for inferring the case id despite the drawbacks discussed in Section 4.5. As future work, the drawbacks related to memory overhead and optimal cut size have the highest priority to ensure a more generic well-functioning of the ICI approach. Furthermore, several heuristics could be applied to filter out the candidate event attributes based on their data types (i.e., ignoring timestamps) and based on their grouping ratios.

Identifying the event log scheme is among the requirements of the data merging approach proposed in the next chapter. Indeed, by applying the ICI approach on the IS log files generated by the BPS, it is possible to iterate across all the log files and merge events belonging to the same case.

Chapter 5

Data Mapping and Merging



This chapter aims at solving the data mapping and merging challenge in order to generate a comprehensive event log file that includes all the events that occurred in an IoT environment, thus, enabling process mining capabilities to discover and analyze the overall process model. Section 5.1

provides an overview on the data data mapping and merging problem. Section 5.2 discusses the related work. Section 5.3 presents the data mapping and merging approach along with the concepts used to explain it. Section 5.4 describes the implementation. Section 5.5 evaluates the approach, and finally Section 5.6 discusses the obtained results.

5.1 Overview

Process mining allows the discovery and the analysis of process models in the presence of an event log. The availability of event logs depends on the maturity of the management systems used. Nowadays, the new era of distributed systems enables to design composite systems that operate in a distributed manner; thus, each system entity is entirely responsible for producing its own time-stamped documentation recording its execution trace. To explore the process mining capabilities on a larger scale, it is necessary to be able to reconstruct a comprehensive end-to-end business process by merging all event logs generated by the different system entities. In an ideal environment, all the system entities would agree upon a unique case id and provide similar well-structured event logs that facilitate the merging procedure. Unfortunately, it is not always the case, thus a pre-processing of the event logs is required to apply the merging successfully. In this context, it is to necessary to define the granularity level at which the merging should be applied.

Claes and Poels in [15] described the following levels of granularity: raw data level which includes raw data stored in database tables or files, structured data level which includes event logs produced by information systems, and model level which includes pre-defined process models. In an IoT environment where a network of IoT devices is deployed to support the work-flow of information systems, the sensor data generated by IoT devices are usually stored at the raw data level, whereas, the case events generated by information systems are typically stored at the structured data level. Therefore, the required merging technique should perform on both raw data level and structured data level taking into account the complex relationships between the interacting sub-processes, and the characteristics of the different sensor types. The sensors classification proposed in Section 2.4 (cf. Chapter 2) provides a good starting point to define a generic approach allowing to map each sensor type to the corresponding business process appropriately. Ultimately this mapping is only feasible with the availability of sophisticated merging techniques based on text mining and temporal relations allowing to maximize the matching between the events recorded across several log files.

The research problem addressed in this chapter is: How to merge raw data log files coming from sensors with structured data log files produced by information systems? The presented approach is built upon the event log merging concepts and sensor data to process model mapping techniques discussed in the literature presented in the next section to design a robust approach that enables merging log files with different granularity levels in an IoT context.

5.2 Background and Related work

The challenges addressed in this chapter are of great significance for the BPM community. Indeed, the process mining manifesto [40] accorded great importance to the event logs merging problem and lists it among the open challenges to be addressed to augment the usability of process mining techniques in distributed environments. Not surprisingly, the BPM-IoT manifesto [23] (C-13) published recently also emphasized the need to bridge the gap between sensor data and event logs to exploit the high volume of information provided by sensors to mine robust process models.

The literature has tackled the event log merging problem from different perspectives. At the structural data level, Cales and Poels in [15] proposed a rule-based merging approach that includes two main steps. The first step consists of defining a set of merging rules specifying the traces matching criteria. The rules are provided as statements indicating the common proprieties between traces or events belonging to the same process execution. To match two traces belonging to different log files, a set of criteria has to be satisfied. These criteria are usually expressed as relations between attribute values. The approach proposes a descriptive language that can be used to select attributes, containers (i.e., event, trace, log), and operators (i.e., add/or) to formulate a wide range of rules. The second step consists of merging the traces conforming the merging rules provided by the user. By this mean, all the possible traces combinations are evaluated using the *Method Evaluation Model* (MEM) proposed by Moody in [32] to find an optimal match. The MEM model uses an Actual effectiveness function that quantifies the extent to which two merged event logs respects the specified merging rules. In fact, the function computes the number of insertions and deletions required on events/traces to obtain a full matching with merging rules. The less are the operations; the highest is the chance that the two traces/events belong to the same process execution. The drawback of this approach is that it assumes the presence of similar case ids among the log files, which contradicts with real-world situations where each distributed system entity uses its own case id. Furthermore, the rule-based approach requires experts domain knowledge to formulate consistent merging rules.

Cales and Poels in another work [14] initiated a new approach to cope with the event logs merging problem by using genetic algorithms. Precisely, the authors suggest to perform merging according to an *Artificial Immune System* (AIS) algorithm which is inspired by the vertebrate immune system. The algorithm implements an affinity function to identify the generation with the highest matching score. The affinity function aims at maximizing the values of the indicators used to judge the coherence and the correctness of merged logs. The authors performed a series of tests to evaluate the AIS approach; however, the results showed that the approach faces some drawbacks, such as the inability to cope with concurrent traces and the likelihood to converge towards a local optima. Xu et al. in [51] proposed an enhanced version of the AIS approach by designing a *Hybrid Artificial Immune System* which integrates a simulated annealing that ensures a diversity for the coming generations, thus avoiding the local optima issue. The simulated annealing is the key feature since not only the solutions with higher affinity are considered for the clonal of next generations but also other solutions with medium affinity are considered, which allows maintaining a certain level of diversity in the population. By enhancing the genetic algorithm part, the hybrid AIS approach demonstrated a higher performance compared to the former AIS approach. Moreover the hybrid AIS approach optimizes the affinity function by considering Allen's interval algebra [4] to use temporal relations between process instances as a coherence indicator for merged logs. However, the approach assumes a one-to-one mapping between events; thus it does not consider other complex relations such as one-to-many relation. Furthermore, genetic algorithms are known to be time consuming [31].

Nooijen et al. in [33] proposed an event log merging technique built upon the data-centric and artifact-centric approach. The work relates to the data-centric systems use case, where relational databases are used to keep track of process execution information. In fact, ERP systems such as SAP Business [1], Microsoft Dynamics AX [2] , or Oracle E-business [3] lacks the notion of case id, thus, the data need to be pre-processed to obtain standard event logs that can be used for process mining. As relational databases allow to explore relations between data objects using primary

[1] See `https://www.sap.com/products/business-one.html`
[2] See `https://community.dynamics.com/ax`
[3] See `http://www.oracle.com/us/products/applications/ebusiness/overview/index.html`

keys and foreign keys, the proposed merging approach aims at discovering the business process entities, their scheme, and the life-cycle model implied by each process entity. Given a structured database, the corresponding scheme is extracted, then using a *K-means Clustering* algorithm, the artifact scheme can be identified such that all the tables containing similar artifact information are grouped together. Afterwards, an event log scheme mapping describing the time evolution for each artifact is generated to build an event log compatible with any process discovery algorithm. The artifact-centric approach has a potential to be used in the IoT context, especially if the IoT sensor interactions can be organized in a data-centric structure. However, the presented use case faces some challenges in term of primary/foreign keys mapping. Namely, the log files generated by the BPS lacks primary keys, since each log file has its own case ids. Moreover, the notion of foreign keys does not apply to the BPS log files because no unique shared event attribute allows gathering all traces belonging to the same process execution.

Raichelson and Soffer in [35] proposed a state-of-art event logs merging technique that overcomes most of the issues mentioned in the previous works. The approach digs into the text mining field to perform a merging that abstracts from the four types of relationships (one-to-one, one-to-many, many-to-one, many-to-many). Moreover, it does not require any experts domain knowledge to tuneup the mapping. As initial step, the approach considers event logs as plain text files and uses the *Term Frequency-Inverse Document Frequency* (TF-IDF) technique to filter out the frequent words in each trace. The output from this step is a "bag of words" representing the important keywords for each trace. Then by using a simple similarity function that counts the common words between two strings of different plain text files, it is possible to obtain the similarity score for each combination of two traces. To strengthen the effectiveness of this approach, the authors use the temporal relations introduced in Allen's interval algebra to ignore cases that are not matching temporally. The work presented in this chapter extends this approach by filling several gaps to build a robust log merging approach. Precisely, the approach presented in this chapter aims at enhancing the similarity function by trying several techniques. Moreover, the approach is extended to operate in larger scale by enabling merging event logs with sensor data logs. To do so, the original algorithm is revised and strengthened. Furthermore, the Log Scheme Identification technique introduced in Chapter 4 is used to automate the merging process by implicitly inferring the case id for the event logs participating in the merging process.

As mentioned in Section 2.4 (Cf. Chapter 2), there exists two types of sensors: *Active Sensors* and *Decorative Sensors*. The merging of Active sensor data is built upon Raichelson's approach as explained previously since the Active sensor data are considered as atomic events; thus, they can be directly merged. However, the Decorative sensor data require a different approach since they are only meant to enrich the events attributes by providing extra information (i.e., temperature at certain time/location). Senderovich et al. in [6] proposed an interaction mining approach to map location data with business process instances. In fact, the authors introduced a new "knowledge layer" between sensor data and event logs. The approach was implemented in a hospital environment where employees and patients got equipped with 900 real-time location sensors to track the subjects involved in business processes. Then by using the location data recorded by the sensors, their timestamps, and the profile of the interacting subject (employees and patients) the approach allowed to infer the current process activities. The approach is formulated as an *Optimal matching* problem where an Integer Linear program (ILP) was deployed to map subjects interactions to process instances. The evaluation of the approach shows that the accuracy depends on the amount of process knowledge provided. Indeed it is bounded to domain-specific applications where a well-documented business process is available apriori; thus the optimal matching problem task is to find the best alignment between the subjects interactions and the documented process.

Eck et al. in [19] addressed the challenge of applying process discovery techniques to mine human behaviours using sensor data. They propose an approach to generate event logs from sensor data. The key aspect of this work is about segmenting sensor data and mapping each segment to a corresponding process activity. The segmentation of sensor data problem has been raised by several authors in the past [7] [16], most of these techniques rely on classifiers used to recognize process activities. In this context, a considerable amount of data is required to train the classifiers. Eck's approach overcomes this requirement by dividing sensor data time series into small windows with a specific length. By comparing the time windows, it is possible to detect points-of-change where the observed behaviour changes suddenly. The points-of-change allow identifying the end and start of process activities. This approach allows to distinguish between the different activities included in a business process from the observed points-of-change; however, it is still necessary to label the activities to generate an event log, which apparently requires domain-specific knowledge from the user. The approach was evaluated in a specific case study where new prototypes of Philips baby bottles were equipped with sensors to evaluate their ease of usability. The approach allowed to successfully mine the users' behaviour and provided great insights into the design of the new smart device. However, since the approach requires experts knowledge to identify process activities, its application remains limited to domain-specific areas.

Dimaggio et al. in [17] introduced an approach to mine human habits, using a well-known process discovery algorithm called *Fuzzy Miner* [13]. The reason behind this choice is the unstructured nature of human habits; thus, any other discovery algorithm will result in a "spaghetti" model [42] where the high number of connections between activities increases the complexity of the process and makes it impossible to analyze. Luckily, the fuzzy miner allows to cope with unstructured processes using the zoom in/out feature inspired from the cartography field. Dealing with unstructured processes while using sensor data to track user behaviours is a challenge [17]; thus the fuzzy miner can always be considered as an effective process discovery technique. Nevertheless, the need to transform sensor data logs to event logs persists. In this context, the authors emphasis on segmenting different process instances from sensor data logs by trying to find correlations between sensor events belonging to the same process instance. Since the approach deals with human habits, each process instance represents the behaviour of one user. It starts when an event is triggered by that user (i.e., wake up), and ends when another event is triggered by the same user (sleep). Then, by using the topological information of sensors and by analyzing the users' movements, it is possible to distinguish between the different process instances. Once the different process instances are identified, generating an event log becomes straight-forward since the approach considers only active sensor data; thus each sensor data entry recorded in the sensor log is regarded as a distinct atomic event. Dimaggio approach has two main contributions that are: dealing with unstructured processes, and splitting sensor data into process instances, the approach was tested using real-world sensor data and demonstrated a good accuracy. However, the assumptions implied by this approach make its usage limited to only active sensors type.

The decorative sensor mapping approach proposed in this chapter is valid for both types of sensor data, as well as it does not require any knowledge from domain experts. By this mean, decorative sensor data are mapped according to a multi-dimensional framework, where each dimension refers to a field (i.e., time, location) of possible intersections between decorative sensor data and process events. Concerning active sensor data, as described previously, they are merged with IS events using text mining techniques and Allen's interval algebra. More insights about this approach will be provided in the coming sections.

5.3 Methodological Approach

The *Active and Decorative Sensor Merging* (ADSM) approach proposed in this chapter aims at defining two phases approach that (a) merges active sensor data with IS event log files, (b) and maps decorative sensor data to merged events. The approach described is structured as follows: Section 5.3.1 lists the set of assumptions considered by the ADSM approach. Section 5.3.2 presents a running example that will be referred to in the coming sections to illustrate the techniques used in the ADSM approach. Section 5.3.3 explains the various concepts and techniques used to design the active sensor data merging phase. Section 5.3.4 describes the decorative sensor data mapping phase and the concepts implied to explain it.

5.3.1 General Assumptions

This section presents the assumptions considered while designing the ADSM approach. The following assumptions are made:

- The IoT environment studied is assumed to contain the following data sources: IS data, active sensor data, and decorative sensor data.
- The devices deployed in the IoT environment all have synchronized clocks.
- A main event log file generated by an information system is assumed to be available and known.
- Event logs use different case identifiers.
- No common event attribute can be used as a foreign key to map traces across different logs.
- The time granularity of events is available in a fine-grained order.
- Each event is handled by only one resource.
- Sensors are assumed to report data instantly with no delay.

5.3.2 Running Example

To illustrate the various steps and techniques used to enable data mapping and merging, this running example is provided. As mentioned in Section 5.3.1, three data sources will be considered in this running example (IS data, active sensor data, decorative sensor data). Using the BPS (Cf. Chapter 3), and the use case scenario described in Section 1.4 (Cf. Chapter 1), small data sets representing each data source type are generated.

To illustrate the event logs merging techniques, the *Clerk Process Log* (Table 5.1) and the *Robot Process Log* (Table 5.2) are used. Both logs are assumed to be generated by information systems; thus, they respect the standard event log structure presented in Definition 2.2.5 (Cf. Chapter 2). Besides, as mentioned in the general assumptions in Section 5.3.1, the two event logs have different case ids. However, to have a ground truth on which it is possible to check the accuracy of the merging approaches, the case id values are composed of an integer and a suffix. The integer part allows identifying traces belonging to the same process execution across the two event logs. For example, in the *Clerk Process Log* (Table 5.1) the events identified with the event ids 001, 002, 003, and 004 belong to the case identified with case id 0CP. Meanwhile, in the *Robot Process Log*

(Table 5.2) events 010, 011, 012, 013 belong to the case 0RP. Hence, one can intuitively infer that case 0CP in the *Clerk Process Log*, and case 0RP in the *Robot Process Log* belongs to the same process execution. To avoid any bias especially with the similarity score function, the case id attributes are not considered in calculating the similarity score.

Case Id	Event Id	Start Time.	End Time.	Event Name	Event type	Subject Group	Subject Id	Object Group	Object Id
0CP	001	2017-11-05 08:00:00	2017-11-05 08:05:00	Request product	process activity	clerk	1c	product	0p
0CP	002	2017-11-05 08:10:15	2017-11-05 08:10:40	Assign robot	process activity	robot	1r	shelf	7s
0CP	003	2017-11-05 08:25:00	2017-11-05 08:30:00	Check if the product was checked	process activity	clerk	1c	product	0p
0CP	004	2017-11-05 08:32:00	2017-11-05 08:32:15	Product Delivered	process activity	clerk	1c	product	0p
1CP	005	2017-11-05 08:02:00	2017-11-05 08:04:30	Request product	process activity	clerk	1c	product	1p
1CP	006	2017-11-05 08:11:15	2017-11-05 08:12:20	Assign robot	process activity	robot	2r	shelf	32s
1CP	007	2017-11-05 08:44:00	2017-11-05 08:50:00	Check if the product was checked	process activity	clerk	1c	product	1p
1CP	008	2017-11-05 08:55:00	2017-11-05 08:59:15	Product Delivered	process activity	clerk	1c	product	1p
2CP	035	2017-11-05 08:51:00	2017-11-05 08:53:00	Request product	process activity	clerk	1c	product	2p
2CP	036	2017-11-05 08:54:00	2017-11-05 08:55:00	Assign robot	process activity	robot	3r	shelf	61s
2CP	037	2017-11-05 09:40:00	2017-11-05 09:50:00	Check if the product was checked	process activity	clerk	1c	product	2p
2CP	038	2017-11-05 09:53:00	2017-11-05 09:57:00	Product Delivered	process activity	clerk	1c	product	2p

Table 5.1: Clerk Process Log example used for the running example

Case Id	Event Id	Start Time.	End Time.	Event Name	Event type	Subject Group	Subject Id	Object Group	Object Id
0RP	010	2017-11-05 08:11:00	2017-11-05 08:13:00	Putdown current shelf	process activity	robot	1r	shelf	5s
0RP	011	2017-11-05 08:14:00	2017-11-05 08:17:23	Go to appropriate shelf	process activity	robot	1r	shelf	7s
0RP	012	2017-11-05 08:18:00	2017-11-05 08:23:00	Move shelf to dock	process activity	robot	1r	shelf	7s
0RP	013	2017-11-05 08:37:31	2017-11-05 08:42:00	Move shelf from dock	process activity	robot	1r	shelf	7s
1RP	018	2017-11-05 08:12:00	2017-11-05 08:13:11	Putdown current shelf	process activity	robot	2r	shelf	24s
1RP	019	2017-11-05 08:14:15	2017-11-05 08:22:10	Go to appropriate shelf	process activity	robot	2r	shelf	32s
1RP	020	2017-11-05 08:23:00	2017-11-05 08:35:00	Move shelf to dock	process activity	robot	2r	shelf	32s
1RP	021	2017-11-05 08:25:00	2017-11-05 08:25:05	Reduce Speed	process activity	robot	2r	shelf	32s
1RP	022	2017-11-05 08:43:31	2017-11-05 08:48:00	Move shelf from dock	process activity	robot	2r	shelf	32s
2RP	210	2017-11-05 08:56:00	2017-11-05 08:57:00	Putdown current shelf	process activity	robot	3r	shelf	75s
2RP	211	2017-11-05 09:00:00	2017-11-05 09:10:00	Go to appropriate shelf	process activity	robot	3r	shelf	61s
2RP	212	2017-11-05 09:11:00	2017-11-05 09:25:00	Move shelf to dock	process activity	robot	3r	shelf	61s
2RP	213	2017-11-05 09:30:31	2017-11-05 09:35:00	Move shelf from dock	process activity	robot	3r	shelf	61s

Table 5.2: Robot Process Log example used for the running example

In order to illustrate the merging of active sensor data with IS events, the *Clerk RFID Sensor Log* file (Table 5.3) is considered. This file represents the active sensor data source, it contains several entries where each entry is identified with an *Entry Id* representing the entry unique identifier, a *Chip Name* referring to which sensor is used, a *Timestamp* recording the moment in time where the sensor is triggered, and an *Item Id* representing the value captured by the sensor. The ground truth for the *Clerk RFID Sensor Log* can be inferred from an extra attribute named *Ground Truth Case Id* used specifically for this purpose. This attribute value is not considered by any of the approaches described in this chapter, the only reason for using it is to have a ground truth to check the accuracy of the merging approaches. For example, the entries 041 and 042 have *Ground Truth Case Id* value 0 which means that they belong the same process execution containing the cases 0CP and 0RP from *Clerk Process Log* and *Robot Process Log* respectively.

Entry Id	Chip Name	Timestamp	Item Id	Ground Truth Case Id
041	collector	2017-11-05 08:24:45	0p	0
042	packageDone	2017-11-05 08:31:45	0p	0
055	collector	2017-11-05 08:43:30	1p	1
056	packageDone	2017-11-05 08:54:01	1p	1
057	collector	2017-11-05 09:39:35	2p	2
058	packageDone	2017-11-05 09:52:25	2p	2

Table 5.3: Clerk RFID Sensor Log example used for the running example

Lastly, to illustrate decorative sensor data source, the *Accelerometer Sensor Data Log* (Table 5.4) is used. The two important inputs provided by each data entry are the *Shelf Id* that is the identifier of the shelf being shacked, and the *Timestamp* that represents the moment in time the shake was recorded. As mentioned in Section 3.4, the accelerometer coordinates' values are abstracted, and only the timestamps when shelves are shaken are recorded. Similarly to the active sensor data log, the ground truth is obtained from an extra attribute named *Ground Truth Case Id* used specifically for this purpose.

EntryId	Shelf Id	Timestamp	Ground Truth Case Id
456	24s	2017-11-05 08:19:10	5
457	32s	2017-11-05 08:24:59	1
458	11s	2017-11-05 09:14:42	6
460	7s	2017-11-05 09:10:12	7

Table 5.4: Accelerometer Sensor Data Log example used for the running example

5.3.3 Active Sensor Data Merging

The merging of active sensor data is performed according to a set of generic techniques allowing to correlate several events stored in different log files. In this context, a main event log file is required; indeed, it is considered as the skeleton of the new merged event log. As stated in the general assumptions Section 5.3.1, the main event log is obtained from the primary information system supporting the IoT environment. Given a main event log and a set of log files (sensor logs and event logs), Allen's interval algebra [4] is used to select the candidate traces matching temporally with the main trace, then by using text mining approaches it is possible to correlate events belonging to the same process execution. Before presenting the full merging approach, it is necessary to explain each component independently. Section 5.3.3.1 describes Allen's temporal rules, Section 5.3.3.2 highlights several similarity scoring functions to be used in order to ensure a high matching accuracy, Section 5.3.3.3 recapitulates the scheme identification approach in the context of event log merging, and Section 5.3.3.4 presents the hierarchical merging approach.

5.3.3.1 Temporal Relations Rules

Allen in [4] introduced a temporal representation that includes 13 temporal relations to cope with temporal intervals in a hierarchical manner based on constraints propagation techniques. In

the events merging context, the temporal relations are used to decide whether two traces match temporally under the following assumptions: (a) each trace represents the execution flow of a distinct process, (b) a trace either belongs to the main process X or belongs to the sub-process Y , (c) the sub-process Y is triggered by the main process X. Table 5.5 depicts the 13 temporal relations, and point out the cases where a match is positive. Note that according to the general assumption defined in Section 5.3.1, the time granularity of events is assumed to be available in a fine-grained order; otherwise relations 9, 10, 13 might not hold.

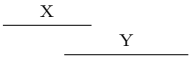
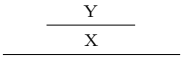
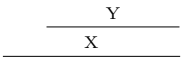
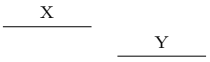
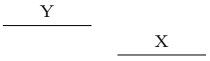
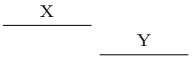
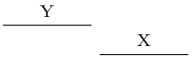
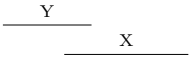
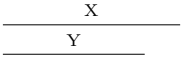
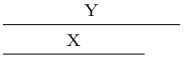
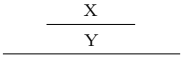
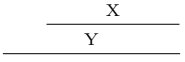
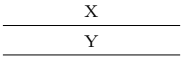
Id	Case	Relation	Illustration	Match	Comment
1	Main process overlaps with sub-process with	X *o* Y		**True**	Match found
2	Sub-process starts and ends during main process	Y *d* X		**True**	Match found
3	Sub-pr. starts during main pr., finishes with main pr.	Y *f* X		**True**	Match found
4	Main process ends before sub-process process starts	X < Y		False	Y cannot provide feedback to X
5	Sub-process ends before main process starts	Y > X		False	X starts after Y, no triggering
6	Main process meets sub-process	X *m* Y		False	Y cannot provide feedback to X
7	Sub-process meets main process	Y *m* X		False	X starts after Y, no triggering
8	sub-process overlaps with main process	Y *o* X		False	Y starts before X, no triggering
9	Main process starts simultaneously with sub-process	X *s* Y		False	Both processes started at same time, no triggering
10	Sub-process starts simultaneously with main process	Y *s* X		False	Both processes started at same time, no triggering
11	Main process starts and ends during sub-process	X *d* Y		False	Y starts before X, no triggering
12	Main pr. starts during sub-pr., finishes with sub-pr.	X *f* Y		False	Y starts before X, no triggering
13	Main process equal sub-process	X = Y		False	X and Y start and end at the same time, no triggering

Table 5.5: Allen's Temporal Relations

In the process of merging IS event logs, the ADSM approach iterates over the traces of each event log in order to temporally match traces belonging to different event logs. Given a main event log $L_m = (S(\mathcal{E}_m), T_m)$, where $S(\mathcal{E}_m)$ is an event log scheme, and T_m is a set of traces, and a sub event log $L_s = (S(\mathcal{E}_s), T_s)$, where $S(\mathcal{E}_s)$ is an event log scheme, and T_s is a set of traces (cf.

Definition 2.2.2, Chapter 2). For each trace $t_m \in T_m$, and for each trace $t_s \in T_s$, t_m and t_s are checked against the 13 temporal relations. In case a positive match is found, t_s is seen as a candidate matching trace with t_m.

The process of merging active sensor data with IS event logs is different from the previous one. Since the active sensor data cannot be grouped into traces, traces from the main event log are checked against single sensor data entries from the sensor log. Given a main event log $L_m = (S(\mathcal{E}_m), T_m)$, where $(S(\mathcal{E}_m)$ is an event log scheme , and T_m is a set of traces (cf. Definition 2.2.2, Chapter 2), and a sensor log $\mathcal{O}_s$. For each trace $t_m \in T_m$, and for each sensor data entry $o_s \in \mathcal{O}_s$ (cf. Definition 2.2.6, Chapter 2), if o_s starts within the time interval of trace t_m, then o_s is seen as a candidate matching event with t_m

The running example introduced in Section 5.3.2 is used to the illustrate the temporal relations. In this context, the *Clerk Process Log* (Table 5.1) is set as main log, and the aim is to find the corresponding temporally matching traces in *Robot Process Log* (Table 5.2) , and *Clerk RFID Sensor Log* (Table 5.3). Since both *Clerk Process Log*, and *Robot Process Log* are event logs, the matching will be done according to Allen's relations. Consequently, only the cases respecting relations 1, 2, or 3 on Table 5.5 are selected. Table 5.6 shows the start-time and end-time of each trace, note that each trace corresponds to one case (cf. Definition 2.2.3, Chapter 2). Table 5.7 illustrates the selection process.

Case Id	Start-time	End-time
0CP	2017-11-05 08:00:00	2017-11-05 08:32:15
1CP	2017-11-05 08:02:00	2017-11-05 08:59:15
2CP	2017-11-05 08:51:00	2017-11-05 09:57:00
0RP	2017-11-05 08:11:00	2017-11-05 08:42:00
1RP	2017-11-05 08:12:00	2017-11-05 08:48:00
2RP	2017-11-05 08:56:00	2017-11-05 09:35:00

Table 5.6: Time Intervals for traces in *Clerk Process Log* (Table 5.1) and *Robot Process Log* (Table 5.2)

Main L. trace	Sub L. trace	Allen's Rule	Illustration	Selected
0CP	0RP	1- X *o* Y	0CP (08:00:00 - 08:32:15) 0RP (08:11:00 - 08:42:00)	True
0CP	1RP	1- X *o* Y	0CP (08:00:00 - 08:32:15) 1RP (08:12:00 - 08:48:00)	True
0CP	2RP	4- X < Y	0CP (08:00:00 - 08:32:15) 2RP (08:56:00 - 09:35:00)	False
1CP	0RP	2- Y *d* X	0RP (08:11:00 - 08:42:00) 1CP (08:02:00 - 08:59:15)	True
1CP	1RP	2- Y *d* X	1RP (08:12:00 - 08:48:00) 1CP (08:02:00 - 08:59:15)	True
1CP	2RP	1- X *o* Y	1CP (08:02:00 - 08:59:15) 2RP (08:56:00 - 09:35:00)	True
2CP	0RP	5- Y > X	0RP (08:11:00 - 08:42:00) 2CP (08:51:00 - 09:57:00)	False
2CP	1RP	5- Y > X	1RP (08:12:00 - 08:48:00) 2CP (08:51:00 - 09:57:00)	False
2CP	2RP	2- Y *d* X	2RP (08:56:00 - 09:35:00) 2CP (08:51:00 - 09:57:00)	True

Table 5.7: Selection process according to Allen's temporal relations

According to Table 5.7 the candidate traces for 0CP are: 0RP, 1RP, the candidate traces for 1CP are 0RP, 1RP, 2RP, and the only candidate trace for 2CP is 2RP.

To match sensor data entries from the *Clerk RFID Sensor Log* with traces from *Clerk Process Log*, for each trace in *Clerk Process Log*, all the active sensor data entries starting within the trace time interval are selected as candidate matching events. The selection results are shown in Table 5.8.

Main log Case Id	Start-time	End-time	Sensor Log Matching Data Entries Id
0CP	2017-11-05 08:00:00	2017-11-05 08:32:15	041, 042
1CP	2017-11-05 08:02:00	2017-11-05 08:59:15	041, 042, 055, 056
2CP	2017-11-05 08:51:00	2017-11-05 09:57:00	056, 057, 058

Table 5.8: Time interval selection results between main event log *Clerk Process Log* (Table 5.1) and sensor log *Clerk RFID Sensor Log* (Table 5.3)

According to Table 5.8 the candidate data entry ids for 0CP are: 041, 042, the candidate data entry ids for 1CP are: 041, 042, 055, 056 and data entry ids for 2CP are: 056, 057, 058. The next step is to use text mining techniques to find the optimal matching.

5.3.3.2 Text Mining and Similarity Scoring

After knowing the potentially matching traces/data-entries with the main event log, text mining can be used to define a similarity scoring function to quantify the similarity between the matching

traces/data-entries. In this context, *Term frequency–inverse document frequency* (TF-IDF) is used.

TF-IDF is a numerical statistical approach that allows evaluating the importance of words in documents [29, p. 8]. The primary usage of this techniques comes out to solve the document categorization problem where several documents are required to be classified based on their subjects of interest. TF-IDF approach proposes to assign a weight to each word based on its occurrence in the document, which allows identifying keywords with highest weights, which are used to infer documents' subjects. In contrast to Raichelson approach where TF-IDF is used to filter out irrelevant words in the event log, the ADSM approach uses TF-IDF as a weighting factor in the similarity scoring function. In other words, each candidate trace/data-entry is transformed to plain text, then each word in the plaintext is assigned a weight representing its importance in the corresponding log, which is considered by the similarity function.

The similarity function using TF-IDF is defined as follows: Let pe_i and pe_j be two plain text strings resulting from converting two candidate traces/data-entries to plain-text, and log_i, log_j be the documents containing pe_i and pe_j respectively. The similarity function $simTF(pe_i, pe_j, log_i, log_j)$ is sum of the weights of the common words in pe_i and pe_j. Algorithm 2 allows to compute $simTF(pe_i, pe_j)$, using the function $getWeights(word, log)$ that takes as input a word, and a log and provides as output a weight value that represents the word importance in the log.

Algorithm 2: Smilarity function using TF-IDF

```
1  Function simTF(pe_i, pe_j, log_i, log_j)
   Input  : pe_i, and pe_j are two plain text events, and log_i, and log_j are the documents containing
            pe_i and pe_j respectively
   Output: score Similarity score using TF-IDF weights
2  begin
3      C ← ∅                                   // Set of common words between pe_i and pe_j
4      score ← 0                                                  // initialize the score
5      /* Iterate over the words in pe_i                                                  */
6      foreach w_i ∈ pe_i do
7          /* Iterate over the words in pe_j                                              */
8          foreach w_j ∈ pe_j do
9              if w_i = w_j and w_i ∉ C then
10                 score ← score + (getWeights(w_i, log_i) + getWeights(w_j, log_j)) / 2   // update score
11                 C ← C ∪ w_i                          // add w_i to the set of common words C
12     return score
```

Another approach to quantify similarity is the *Jaccard index* [29, p. 94], which measures the similarity between two finite sets. Let A and B be two finite sets, the Jaccard index is the size of the intersection of the two sets, divided by the union of the two sets. Let A and B be two finite sets, the *Jaccard index* is defined as follows: $J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A|+|B|-|A \cap B|}$, such that $0 \leq J(A,B) \leq 1$, and if $A \leftarrow \emptyset$ and $B \leftarrow \emptyset$ then $J(A,B) = 1$. The *Jaccard index* is used by the ADSM approach as follows: Let pe_i and pe_j be two plain text strings resulting from converting two candidate traces/data-entries to plain-text, the similarity function using Jaccard index is $J(pe_i, pe_j)$.

Once the similarity scores are computed using $simTF(pe_i, pe_j, log_i, log_j)$ or $J(pe_i, pe_j)$, it is crucial to set a selection ratio to define which traces/data-entries represents a good match. To do so, the only the top $n\%$ traces/data-entries with high similarity scores are selected. In the running

example, the selection ratio is set to 0.2; thus, only the top 20% traces/data-entries with high similarity scores are selected.

In order to illustrate the two similarity functions $simTF(pe_i, pe_j, log_i, log_j)$, and $J(pe_i, pe_j)$, the candidate traces/data-entries selected from the example of the previous section (Tables 5.7, and 5.8) are used. Table 5.7 shows the candidate traces for merging *Clerk Process Log* (Table 5.1) with *Robot Process Log* (Table 5.2). After converting each trace in *Clerk Process Log* and *Robot Process Log* to plain text, the similarity scores are calculated as reported in Table 5.9.

Main log Case Id	Sub log Candidate Case Id	$simTF(pe_i, pe_j, log_i, log_j)$	$J(pe_i, pe_j)$
0CP	0RP	**1.7931**	**0.1625**
0CP	1RP	1.5151	0.0278
1CP	0RP	1.6129	0.0278
1CP	1RP	**1.6774**	**0.0295**
1CP	2RP	1.6129	0.0278
2CP	2RP	**1.7931**	**0.0295**

Table 5.9: Similarity scores for the candidate traces matching temporally between *Clerk Process Log* (Table 5.1) and *Robot Process Log* (Table 5.2)

According to the results shown in Table 5.9, the similarity scores using $simTF(pe_i, pe_j, log_i, log_j)$ for *1CP-0RP*, *1CP-1RP*, and *1CP-2RP* are *1.6129*, *1.6774*, and *1.6129* respectively. By setting the selection ratio to *0.2*, only *1CP-1RP* is selected. Similarly for *0CP-0RP*, and *2CP-2RP* using both similarity functions. Hence the results align with the ground truth provided in the Section 5.3.2. Table 5.10 shows the event log obtained by merging *Clerk Process Log* (Table 5.1) and *Robot Process Log* (Table 5.2) using the Jaccard similarity function $J()$.

New C.Id	Case Id	Event Id	Start Time.	End Time.	Event Name	Event type	Subject Group	Subject Id	Object Group	Object Id
26109	1CP	5	2017-11-05 08:02:00	2017-11-05 08:04:30	Request product	clerk	1c	product	1p	process activity
26109	1CP	6	2017-11-05 08:11:15	2017-11-05 08:12:20	Assign robot	robot	2r	shelf	32s	process activity
26109	1RP	18	2017-11-05 08:12:00	2017-11-05 08:13:11	Putdown current shelf	robot	2r	shelf	24s	process activity
26109	1RP	19	2017-11-05 08:14:15	2017-11-05 08:22:10	Go to appropriate shelf	robot	2r	shelf	32s	process activity
26109	1RP	20	2017-11-05 08:23:00	2017-11-05 08:35:00	Move shelf to dock	robot	2r	shelf	32s	process activity
26109	1RP	20	2017-11-05 08:25:00	2017-11-05 08:25:05	Reduce Speed	robot	2r	shelf	32s	process activity
26109	1RP	21	2017-11-05 08:43:31	2017-11-05 08:48:00	Move shelf from dock	robot	2r	shelf	32s	process activity
26109	1CP	7	2017-11-05 08:44:00	2017-11-05 08:50:00	Check if the product was checked	clerk	1c	product	1p	process activity
26109	1CP	8	2017-11-05 08:55:00	2017-11-05 08:59:15	Product Delivered	clerk	1c	product	1p	process activity
8864	0CP	1	2017-11-05 08:00:00	2017-11-05 08:05:00	Request product	clerk	1c	product	0p	process activity
8864	0CP	2	2017-11-05 08:10:15	2017-11-05 08:10:40	Assign robot	robot	1r	shelf	7s	process activity
8864	0RP	10	2017-11-05 08:11:00	2017-11-05 08:13:00	Putdown current shelf	robot	1r	shelf	5s	process activity
8864	0RP	11	2017-11-05 08:14:00	2017-11-05 08:17:23	Go to appropriate shelf	robot	1r	shelf	7s	process activity
8864	0RP	12	2017-11-05 08:18:00	2017-11-05 08:23:00	Move shelf to dock	robot	1r	shelf	7s	process activity
8864	0CP	3	2017-11-05 08:25:00	2017-11-05 08:30:00	Check if the product was checked	clerk	1c	product	0p	process activity
8864	0CP	4	2017-11-05 08:32:00	2017-11-05 08:32:15	Product Delivered	clerk	1c	product	0p	process activity
8864	0RP	13	2017-11-05 08:37:31	2017-11-05 08:42:00	Move shelf from dock	robot	1r	shelf	7s	process activity
73408	2CP	35	2017-11-05 08:51:00	2017-11-05 08:53:00	Request product	clerk	1c	product	2p	process activity
73408	2CP	36	2017-11-05 08:54:00	2017-11-05 08:55:00	Assign robot	robot	3r	shelf	61s	process activity
73408	2RP	210	2017-11-05 08:56:00	2017-11-05 08:57:00	Putdown current shelf	robot	3r	shelf	75s	process activity
73408	2RP	211	2017-11-05 09:00:00	2017-11-05 09:10:00	Go to appropriate shelf	robot	3r	shelf	61s	process activity
73408	2RP	212	2017-11-05 09:11:00	2017-11-05 09:25:00	Move shelf to dock	robot	3r	shelf	61s	process activity
73408	2RP	213	2017-11-05 09:30:31	2017-11-05 09:35:00	Move shelf from dock	robot	3r	shelf	61s	process activity
73408	2CP	37	2017-11-05 09:40:00	2017-11-05 09:50:00	Check if the product was checked	clerk	1c	product	2p	process activity
73408	2CP	38	2017-11-05 09:53:00	2017-11-05 09:57:00	Product Delivered	clerk	1c	product	2p	process activity

Table 5.10: Event log obtained by merging *Clerk Process Log* and *Robot Process Log* using the Jaccard similarity function $J()$

The same approach can be applied to compute the similarity scores between the traces from *Clerk Process Log*, and the candidate sensor data entries from *Clerk RFID Sensor Log* (Table 5.8). After

converting the traces/data-entries to plain text, the similarity scores are calculated as reported in Table 5.11.

Main log Case Id	Sub log Candidate Entry id	$simTF(pe_i, pe_j, log_i, log_j)$	$J(pe_i, pe_j)$
0CP	041	**2.0000**	**0.0025**
0CP	042	**2.0000**	**0.0025**
1CP	041	1.7727	0.0
1CP	042	1.7727	0.0
1CP	055	**2.0000**	**0.0025**
1CP	056	**2.0000**	**0.0025**
2CP	056	1.6363	0.0
2CP	057	**1.8571**	**0.0025**
2CP	058	**1.8571**	**0.0025**

Table 5.11: Similarity scores between the traces from *Clerk Process Log* (Table 5.1), and the candidate sensor data entries from *Clerk RFID Sensor* Log (Table 5.3)

According to the results shown in Table 5.11, the similarity scores using $simTF(pe_i, pe_j, log_i, log_j)$ for *1CP-041*, *1CP-042*, *1CP-055* and *1CP-056* are *1.7727*, *1.7727*, *2*, and *2* respectively. By setting the selection ratio to *0.2*, only *1CP-055*, and *1CP-056* are selected. Similarly for the others using both similarity functions. Hence the results align with the ground truth provided in the Section 5.3.2. Table 5.12 shows the event log obtained by merging *Clerk Process Log* (Table 5.1) and *Clerk RFID Sensor Log* (Table 5.3) using Jaccard similarity function.

New C.Id	Case Id	Event Id	Start Time.	End Time.	Event Name	Event type	Subject Group	Subject Id	Object Group	Object Id
87562	0CP	1	2017-11-05 08:00:00	2017-11-05 08:05:00	Request product	clerk	1c	product	0p	process activity
87562	0CP	2	2017-11-05 08:10:15	2017-11-05 08:10:40	Assign robot	robot	1r	shelf	7s	process activity
87562	41	41	2017-11-05 08:24:45	2017-11-05 08:24:45	collector				0p	
87562	0CP	3	2017-11-05 08:25:00	2017-11-05 08:30:00	Check if the product was checked	clerk	1c	product	0p	process activity
87562	42	42	2017-11-05 08:31:45	2017-11-05 08:31:45	packageDone				0p	
87562	0CP	4	2017-11-05 08:32:00	2017-11-05 08:32:15	Product Delivered	clerk	1c	product	0p	process activity
62996	1CP	5	2017-11-05 08:02:00	2017-11-05 08:04:30	Request product	clerk	1c	product	1p	process activity
62996	1CP	6	2017-11-05 08:11:15	2017-11-05 08:12:20	Assign robot	robot	2r	shelf	32s	process activity
62996	55	55	2017-11-05 08:43:30	2017-11-05 08:43:30	collector				1p	
62996	1CP	7	2017-11-05 08:44:00	2017-11-05 08:50:00	Check if the product was checked	clerk	1c	product	1p	process activity
62996	56	56	2017-11-05 08:54:01	2017-11-05 08:54:01	packageDone				1p	
62996	1CP	8	2017-11-05 08:55:00	2017-11-05 08:59:15	Product Delivered	clerk	1c	product	1p	process activity
86355	2CP	35	2017-11-05 08:51:00	2017-11-05 08:53:00	Request product	clerk	1c	product	2p	process activity
86355	2CP	36	2017-11-05 08:54:00	2017-11-05 08:55:00	Assign robot	robot	3r	shelf	61s	process activity
86355	57	57	2017-11-05 09:39:35	2017-11-05 09:39:35	collector				2p	
86355	2CP	37	2017-11-05 09:40:00	2017-11-05 09:50:00	Check if the product was checked	clerk	1c	product	2p	process activity
86355	58	58	2017-11-05 09:52:25	2017-11-05 09:52:25	packageDone				2p	
86355	2CP	38	2017-11-05 09:53:00	2017-11-05 09:57:00	Product Delivered	clerk	1c	product	2p	process activity

Table 5.12: Event log obtained by merging *Clerk Process Log* and*Clerk RFID Sensor Log* using Jaccard similarity function.

5.3.3.3 Event Log Scheme Identification

The application of the ADSM approach requires knowledge about some attributes shared among all the log files, namely, the timestamp, and the case id. The timestamp (i.e., start/end time) is crucial to apply Allen's temporal relations to check whether two traces match temporally. Therefore, this attribute is mandatory to have in any log file (i.e., event log, sensor log).

The other important attribute is the case id because it allows to group events belonging to the same process execution in each event log, for the purpose to identify the traces to be processed by both Allen's temporal relations and the similarity scoring function. In this context, case ids can

be inferred from event logs using the ICI approach introduced in Chapter 4, whereas concerning sensor log files no case id attribute is required.

Another optional attribute is the event identifier which allows distinguishing between events. However, since this attribute is not always present in log files, the ADSM implementation can generate internal event ids.

5.3.3.4 Hierarchical Merging

The ADSM approach allows for a hierarchical merging of log files, by this means it aims at finding the best merging order to merge several log files. The hierarchical merging is built upon the scores computed using the similarity functions introduced in Section 5.3.3.2, such that among all the possible merging combinations the combination with the highest average similarity score is considered.

Precisely, let L_s be the set of log files, and $ls(i) = L_{s_i}$, $\forall\, 1 \leq i \leq |L_s|$ be a selection operator on L_s, $p = (m, s, a, score)$ be a merging tuple where $m \in L_s$ represents the main log, $s \in L_s$ represents the sub log with $s \neq m$, a represents the log file generated by merging m with s, and $score \in \mathbb{R}$ represents the average similarity score of a that is the sum of the similarity scores obtained by merging m with s using the Jaccard Similarity function or the TF-IDF similarity function (Section 5.3.3.2) divided by the number of candidate traces/log-entries selected by Allen's temporal rules. In a merging tuple p, the following projection functions are defined: $\pi_m(p) = m$, $\pi_s(p) = s$, $\pi_a(p) = a$, and $\pi_{score}(p) = score$. Also Let P be a set of merging tuples p, $max(P)$ be the tuple with the highest similarity score $\pi_{score}(p)$, $mergeFiles(m, s)$ be a merging function that takes as input a main log file $m \in L_s$ and a sub log file $s \in L_s$ and returns a tuple p.

Given j the index of the main log file in L_s, then $m = ls(j)$ correspond to the main log file in L_s. (1) For each log file $s \in L_s$, if $s \neq m$ then a merging tuple $p = mergeFiles(m, s)$ is generated, and appended to the set of merging tuples P. Once (1) is completed, the merging tuple with the highest score $p_h = max(P)$ is obtained. Afterwards, the logs $\pi_m(p_h)$, $\pi_s(p_h)$ contributing in the merging process are subtracted from the log set L_s, and $\pi_a(p_h)$ the new generated log is appended to L_s and set as main event log. The same approach is done iteratively until the size of L_s is equal

to 1. Algorithm 3 implements the hierarchical merging approach.

Algorithm 3: Hierarchical Merging Algorithm

```
1  Function Hmerger(mainlogIndex, L_s)
   Input  : mainlogIndex the index of the initial main log file in L_s , and L_s the set of log files
   Output: log Merged log file
2  begin
3      m ← ls(mainlogIndex)      // get the main log using the selection operator ls(i)
4      /* Iterate until the size of L_s is equals to 1                                    */
5      while |L_s| ≠ 1 do
6          P ← ∅                                        // initialize set of merging tuples
7          /* Iterate over L_s the set of logs                                            */
8          foreach s ∈ L_s do
9              if s ≠ m then
10                 p ← mergeFiles(m,s)                          // merge logs m, and s
11                 P ← P ∪ p                  // append p to the set of merging tuples P
12         p_h ← max(P)          // get the merging tuple with the highest merging score
13         L_s ← L_s \ π_m(p_h)                        // remove main log π_m(p_h) from L_s
14         L_s ← L_s \ π_s(p_h)                         // remove sub log π_s(p_h) from L_s
15         L_s ← L_s ∪ π_a(p_h)                       // append merged log π_a(p_h) to L_s
16         m ← π_a(p_h)                               // set the merged log as main log
17     return m
```

To illustrate the hierarchical merging approach, the logs files presented in the running example Section 5.3.2 are used. Given CP the *Clerk Process Log* (Table 5.1), RP the *Robot Process Log* (Table 5.2), and CR the *Clerk RFID Sensor Log* (Table 5.3), then the logs set $L_s = \{CP, RP, CR\}$.

Suppose that CP is the main log file, in the first while loop iteration (Lines 5 - 16), the first foreach iteration (Line 8 - Line 11) CP cannot be merged with itself (Line 9). The second iteration CP is merged with RP, the resulting merging tuple is $p_0 = (CP, RP, CP\text{-}RP, 1.6502)$ where $\pi_{score}(p_0) = 1.6502$ is obtained using the similarity function $J()$ (Line 10). The third iteration CP is merged with CR, the resulting merging tuple is $p_1 = (CP, CR, CP\text{-}CR, 1.8773)$ where $\pi_{score}(p_1) = 1.8773$ is obtained using the similarity function $J()$ (Line 10).

Clearly $\pi_{score}(p_1) > \pi_{score}(p_0)$, which means that the average score from merging CP and CR is the highest, thus, $p_h = p_1$ (Line 12). Afterwards, $\pi_m(p_h) = CP$, and $\pi_s(p_h) = CR$ are removed from L_s (Lines 13 - 14), and the new merged log $\pi_a(p_h) = CP\text{-}CR$ is appended to L_s, also $\pi_a(p_h)$ is set as main log for the next iteration (Line 16). At this stage the logs set $L_s = \{RP, CP\text{-}CR\}$.

In the next while loop iteration $CP\text{-}CR$ and RP are merged, and the generated merging tuple is $p_0 = (CP\text{-}CR, RP, CP\text{-}CR\text{-}RP, 1.7941)$ where $\pi_{score}(p_0) = 1.7941$ is obtained using the similarity function $J()$. in this case there exists only one merging tuple, thus $p_h = p_0$. Afterwards, both $CP\text{-}CR$ and RP are removed from L_s and the new merged log $CP\text{-}CR\text{-}RP$ is appended to L_s, also $CP\text{-}CR\text{-}RP$ is set as main log for the next iteration. However, this time the size of L_s is 1 ($L_s = \{CP\text{-}CR\text{-}RP\}$). therefore no more while loop iterations are possible. Finally the log $CP\text{-}CR\text{-}RP$ is returned (Line 17). Table 5.14 shows the final log $CP\text{-}CR\text{-}RP$.

New C.Id	Case Id	Event Id	Start Time.	End Time.	Event Name	Event type	Subject Group	Subject Id	Object Group	Object Id
53269	0CP	1	2017-11-05 08:00:00	2017-11-05 08:05:00	Request product	clerk	1c	product	0p	process activity
53269	0CP	2	2017-11-05 08:10:15	2017-11-05 08:10:40	Assign robot	robot	1r	shelf	7s	process activity
53269	0RP	10	2017-11-05 08:11:00	2017-11-05 08:13:00	Putdown current shelf	robot	1r	shelf	5s	process activity
53269	0RP	11	2017-11-05 08:14:00	2017-11-05 08:17:23	Go to appropriate shelf	robot	1r	shelf	7s	process activity
53269	0RP	12	2017-11-05 08:18:00	2017-11-05 08:23:00	Move shelf to dock	robot	1r	shelf	7s	process activity
53269	41	41	2017-11-05 08:24:45	2017-11-05 08:24:45	collector				0p	
53269	0CP	3	2017-11-05 08:25:00	2017-11-05 08:30:00	Check if the product was checked	clerk	1c	product	0p	process activity
53269	42	42	2017-11-05 08:31:45	2017-11-05 08:31:45	packageDone				0p	
53269	0CP	4	2017-11-05 08:32:00	2017-11-05 08:32:15	Product Delivered	clerk	1c	product	0p	process activity
53269	0RP	13	2017-11-05 08:37:31	2017-11-05 08:42:00	Move shelf from dock	robot	1r	shelf	7s	process activity
70923	2CP	35	2017-11-05 08:51:00	2017-11-05 08:53:00	Request product	clerk	1c	product	2p	process activity
70923	2CP	36	2017-11-05 08:54:00	2017-11-05 08:55:00	Assign robot	robot	3r	shelf	61s	process activity
70923	2RP	210	2017-11-05 08:56:00	2017-11-05 08:57:00	Putdown current shelf	robot	3r	shelf	75s	process activity
70923	2RP	211	2017-11-05 09:00:00	2017-11-05 09:10:00	Go to appropriate shelf	robot	3r	shelf	61s	process activity
70923	2RP	212	2017-11-05 09:11:00	2017-11-05 09:25:00	Move shelf to dock	robot	3r	shelf	61s	process activity
70923	2RP	213	2017-11-05 09:30:31	2017-11-05 09:35:00	Move shelf from dock	robot	3r	shelf	61s	process activity
70923	57	57	2017-11-05 09:39:35	2017-11-05 09:39:35	collector				2p	
70923	2CP	37	2017-11-05 09:40:00	2017-11-05 09:50:00	Check if the product was checked	clerk	1c	product	2p	process activity
70923	58	58	2017-11-05 09:52:25	2017-11-05 09:52:25	packageDone				2p	
70923	2CP	38	2017-11-05 09:53:00	2017-11-05 09:57:00	Product Delivered	clerk	1c	product	2p	process activity
92494	1CP	5	2017-11-05 08:02:00	2017-11-05 08:04:30	Request product	clerk	1c	product	1p	process activity
92494	1CP	6	2017-11-05 08:11:15	2017-11-05 08:12:20	Assign robot	robot	2r	shelf	32s	process activity
92494	1RP	18	2017-11-05 08:12:00	2017-11-05 08:13:11	Putdown current shelf	robot	2r	shelf	24s	process activity
92494	1RP	19	2017-11-05 08:14:15	2017-11-05 08:22:10	Go to appropriate shelf	robot	2r	shelf	32s	process activity
92494	1RP	20	2017-11-05 08:23:00	2017-11-05 08:35:00	Move shelf to dock	robot	2r	shelf	32s	process activity
92494	1RP	21	2017-11-05 08:25:00	2017-11-05 08:25:05	Reduce Speed	robot	2r	shelf	32s	process activity
92494	55	55	2017-11-05 08:43:30	2017-11-05 08:43:30	collector				1p	
92494	1RP	22	2017-11-05 08:43:31	2017-11-05 08:48:00	Move shelf from dock	robot	2r	shelf	32s	process activity
92494	1CP	7	2017-11-05 08:44:00	2017-11-05 08:50:00	Check if the product was checked	clerk	1c	product	1p	process activity
92494	56	56	2017-11-05 08:54:01	2017-11-05 08:54:01	packageDone				1p	
92494	1CP	8	2017-11-05 08:55:00	2017-11-05 08:59:15	Product Delivered	clerk	1c	product	1p	process activity

Table 5.13: Log *CP-CR-RP* obtained using the hierarchical merging algorithm

5.3.4 Decorative Sensor Data Mapping

The mapping of decorative sensor data is performed according to an n-dimensional framework, where each dimension represents a field of intersection between decorative sensor data log and event log. Generally, the dimensions can be obtained from the attributes of the decorative sensor log file. For instance, the *Accelorometer Sensor Data* log shown in Table 5.4 has 2 attributes (*Shelf Id* and *Timestamp*) that can be represented in the framework. Thus, the framework will have two dimensions that are *Shelf Id* and *Timestamp*

The *Decorative Sensor Data Mapping* (DSDM) approach aims at finding the intersection between the decorative sensor data, and the log events according to the dimensions inferred from the decorative sensor data log attributes. To do so, both decorative sensor data and log events are projected on the n-dimensional framework. Every time a decorative data entry intersects with an event, the event is decorated with an extra attribute to emphasize on the intersection. The intersection dimensions are categorized into the following two types: (a) discrete dimensions (i.e., *Shelf Id* attribute in *Accelorometer Sensor Data* Log), (b) continuous dimensions (i.e., *Timestamp* attribute in *Accelorometer Sensor Data* Log).

Given the set of decorative sensor log attributes corresponding to the framework dimensions, the first step is to find all log events that can be represented in the framework, knowing that an event can only be represented in the framework, if it contains common attributes with the decorative sensor data log. In this context, it is important to distinguish between discrete attributes (corresponding to discrete dimension), and continuous attributes (corresponding to continuous dimension). For discrete attributes, the matching events in the event log need to share the same attribute values as the ones in the decorative sensor data log. While, for continuous attributes, the events from the event log and the data entries from the decorative sensor log need to intersect within the common attribute domain. For instance, *Shelf Id* attribute in the *Accelorometer Sensor Data* Log is a discrete attribute corresponding to a discrete dimension. Therefore, to find events from the *CP-CR-RP* Log (Table 5.14) that can be represented on the *Shelf Id* dimension, it is

mandatory to look for those that have the same shelf Ids as the sensor data entries in the *Accelorometer Sensor Data* Log. Another example is the *Timestamp* attribute in the *Accelorometer Sensor Data* Log that is a continuous attribute corresponding to a continuous dimension. Therefore, to find events from the *CP-CR-RP* Log that can be represented in the *Timestamp* dimension, it is mandatory to look for those that meet in time with the sensor log data entries. For the sake of simplicity, only time is considered as a continuous attribute, however, the same approach can be applied to other continuous data types.

Precisely, Let LD be a set representing a decorative sensor log, $D \in LD$ be a set representing a decorative data entry, and $d \in D$ be a decorative attribute tuple such that $d = (v, t)$ where $t \in \{"discrete", "continuous"\}$ represents the decorative attribute types set, and $v \in \mathcal{V}$ represents the decorative attribute value in the values universe $\mathcal{V}$. In a decorative attribute tuple d, the following projection functions are defined: $\pi_v(d) = v$, $\pi_t(d) = t$. Also let *findEventsWithKeyWord(eventlog,keyword)* be a function that given an event log *eventlog* and a keyword *keyword*, it returns all events in *eventlog* that contains *keyword*, and *findEventsInTimeInterval(eventlog,timestamp)* be a function that given an event log *eventlog* and a time value *timestamp*, it returns all events in *eventlog* that were running at *timestamp*.

Given a decorative sensor log LD and an event log L (cf. Definition 2.2.5, Chapter 2), let ed be a set containing the events in L matching with the discrete attributes values of LD, and ec be a set containing events in L matching with the continuous attributes values in LD. For each data entry D in LD, and for each attribute tuple d in D, if $\pi_t(d) = "discrete"$ then *findEventsWithKeyWord(L,*$\pi_v(d)$*)* returns all events in L (which can be obtained by iterating over all cases in L) that contain the keyword $\pi_v(d)$, which are appended to the set ed. Otherwise if $\pi_t(d) = "continuous"$, then *findEventsInTimeInterval(L,* $\pi_v(d)$*)* returns all events in L that were running at $\pi_v(d)$, which are appended to the set ec.

Let $ei = ed \cap ec$ be a set containing the shared events between ed and ec. The set ei contains the candidate events for intersection with data entries in LD. Algorithm 4 summarizes the procedure

to obtain ei.

Algorithm 4: Finding events compatible with the n-dimensional framework

```
1  Function DecorativeMapping(LD, L)
   Input  : LD decorative sensor log, L event log
   Output: ei The set of events that can be projected into the n-dim framework
2  begin
3      ed ← ∅                    // events in L matching with discrete attributes in LD
4      ec ← ∅                  // events in L matching with continuous attributes in LD
5      /* Iterate over all data entries in LD                                    */
6      foreach D ∈ LD do
7          /* Iterate over all attribute tuples in D                             */
8          foreach d ∈ D do
9              if π_t(d) = "discrete" then
10                 /* append the events containing the value π_t(d) to ed        */
11                 ed ← ed ∪ findEventsWithKeyWord(L,π_v(d))
12             else if π_t(d) = "continuous" then
13                 /* append the events containing the value π_t(d) to ec        */
14                 ec ← ec ∪ findEventsInTimeInterval(L, π_v(d))
15     /* ei is the intersection of the sets ed and ec                           */
16     ei ← ed ∩ ec
17     return ei
```

Once ei is generated using Algorithm 4, it becomes possible to map the decorative data entries to the log events. Let *isKeyWordInEvent(event,keyword)* be a function that given an event *event*, and a keyword *keyword*, it returns *True* if *event* contains *keyword*, or *False* otherwise, and *isEventRunning(event,timestamp)* be a function that given an event *event* and a time value *timestamp*, it returns *True* if *event* was running during *timestamp*, or *False* otherwise. Also Let *addAtrribute(event, marking)* be a function that given an event *event* and a string value *marking*, it appends a new attribute to *event* containing the value *marking*, and returns *event*, and *replace(Log, eOld , eNew)* be a function that given an event log *Log*, an event *eOld*, and an event *eNew*, it replaces *eOld* with *eNew* in *Log*, and returns *Log*. Then, for each decorative data entry $D \in LD$, and for each event $e \in ei$, a flag (Boolean variable) *eventIntersecting = True* is initialized to know if e is intersecting with all attributes in D, then for each attribute tuple $d \in D$, if ($\pi_t(d)$ = "*discrete*" AND $isKeyWordInEvent(e, \pi_v(d))$) OR ($\pi_t(d)$ = "*continuous*" AND $isEventRunningAtTime(e, \pi_v(d))$) then event e is intersecting with attribute d, thus *eventIntersecting = eventIntersecting AND True*, otherwise *eventIntersecting = eventIntersecting AND False*. Since all the data entry attributes have to intersect with the event attributes the *AND* logical operator is used in the flag *eventIntersecting* such that once it get *False*, it remains *False* forever.

Finally, if *eventIntersecting = True*, e is saved in temporary variable *eOld*, and $eNew$ = *addAtrribute(e, marking)* is used to mark the event e, finally *replace(Log, eOld , eNew)* is used to replace *eOld* with $eNew$ in the event log. Algorithm summarizes the decorative mapping approach given LD a decorative sensor log, L an event log, ei a set of events that can be projected into the n-dim

framework, and *marking* a value to mark events in *ei* in case they match with data entries in *LD*.

Algorithm 5: Decorative Sensor Data Mapping

```
1  Function DecorativeMapping(LD,L, ei,marking)
   Input  : LD decorative sensor log, L event log, ei set of events that can be projected into the
            n-dim framework, marking value to mark events in ei in case they match with data
            entries in LD
   Output: L decorated event log
2  begin
3      /* Iterate over all data entries in LD                                          */
4      foreach D ∈ LD do
5          /* Iterate over all events in ei                                            */
6          foreach e ∈ ei do
7              /* Initialize a Boolean variable to know if e is intersecting with all
                 attributes D                                                         */
8              eventIntersecting← True
9              /* Iterate over all attributes in D                                     */
10             foreach d ∈ D do
11                 if (π_t(d) = "discrete" ∧ isKeyWordInEvent(e,π_v(d))) ∨ (π_t(d) = "continuous" ∧
                    isEventRunningAtTime(e,π_v(d)) ) then
12                     /* Event e is intersecting with attribute d                     */
13                     eventIntersecting ← (eventIntersecting ∧ True)
14                 else
15                     eventIntersecting ← ( eventIntersecting ∧ False)
16             /* check if e is intersecting with all attributes of data entry D       */
17             if eventIntersecting then
18                 eOld ← e                                          // save e temporary
19                 /* add extra attribute to e with value marking                      */
20                 eNew ← addAtrribute(e, marking)
21                 /* Replace Event eOld with Event eNew in Log L                      */
22                 L ← replace(L,eOld,eNew)
23     return L
```

To illustrate this approach, the decorative log *Accelerometer Sensor Data* (Table 5.4), and the event log *CP-CR-RP* (Table 5.14) are used. The first step is generate the sets *ed* and *ec* containing the events in *CP-CR-RP* matching with the discrete, and continuous attributes values in *Accelerometer Sensor Data* respectively. using Algorithm 4, the set *ed* is expected to have events containing one of the following shelf ids: 24s, 32s, 11s, 7s (Table 5.4), as result *ed* contains the following events: 18, 6, 19, 20, 21, 22, 2, 11, 12, 13 (Table 5.14), and the set is *ec* is expected to have events that were running one of the following timestamps: 2017-11-05 08:19:10, 2017-11-05 08:24:59, 2017-11-05 09:14:42, 2017-11-05 09:10:12 (Table 5.4), as result *ec* contains the following events: 12, 19, 20, 212 (Table 5.14).

Figure 5.1 depicts a 2-dimensional framework showing intersections between *Accelerometer Sensor Data* (Table 5.4) and *CP-CR-RP* (Table 5.14). The *Y axis* represents the shelfs of the set *ed*, and *X axis* represents timestamps projected into a continuous time interval. The Blue, Red, Green lines refers to events in set *ed*, events in set *ec*, and data entries in set *LD*. A log event and a sensor

data entry intersect in the framework if and only if the log event belongs to the set $ei = ed \cap ec$, and intersects in all dimensions with the sensor data entry attributes.

In this example, the events in sets ed and ec can be projects into the 2-dimensional framework as depicted in Figure 5.1 (Blue lines and Red lines respectively). The set $ei = ed \cap ec$ represents the candidate events for intersection with data entries in the *Accelerometer Sensor Data* (Table 5.4) that are 19, 20, and 12.

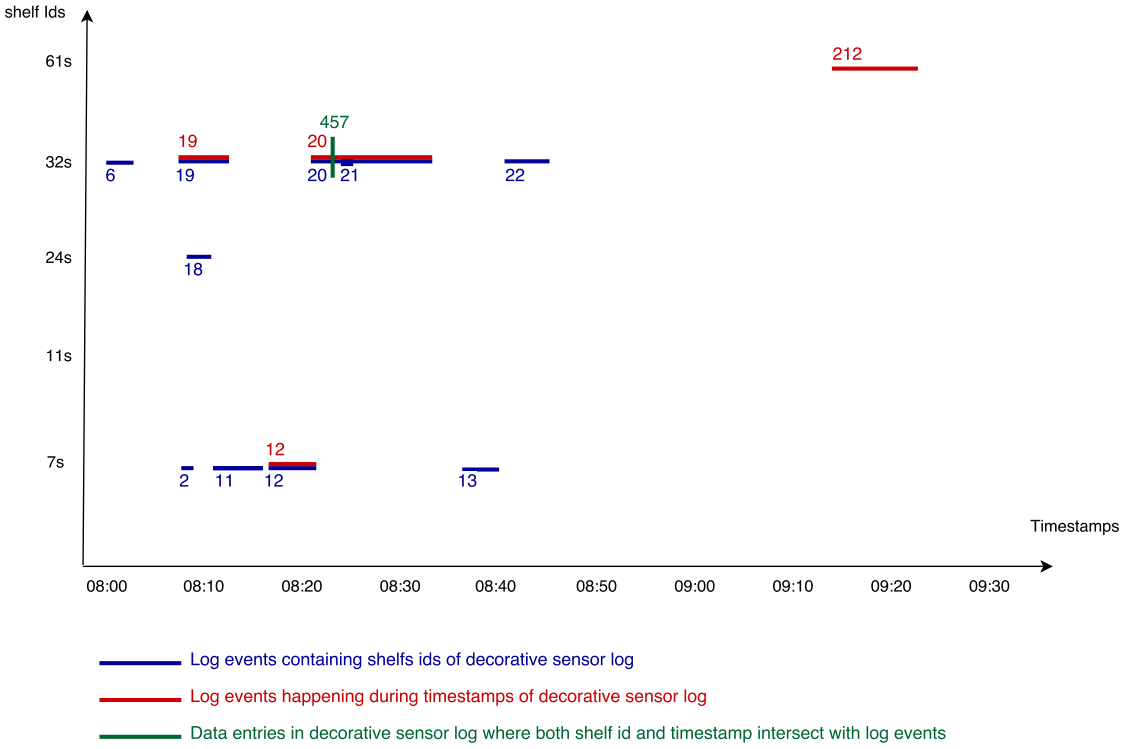


Figure 5.1: 2-dimensional framework showing the intersections between *Accelerometer Sensor Data* (Table 5.4) and *CP-CR-RP* (Table 5.14). The number next to each colored line refers to the event id/data entry id in the parent log file.

As shown in Figure 5.1, there exists only one intersection (Green line) where event 20 from *CP-CR-RP*, and event 457 from *Accelerometer Sensor Data* intersect on both *Timestamp*, and *Shelf Id* dimensions. As result event 20 will be decorated with a marking value. In this example, the marking value chosen is "shacked". The final event log generated by the ADSM approach (using the running example Section 5.3.2) is shown in Table 5.14. Note that event 20, is now decorated with value "shacked", which means that the shelf was shaken while it was moving to the dock.

New C.Id	Case Id	Event Id	Start Time.	End Time.	Event Name	Event type	Subject Group	Subject Id	Object Group	Object Id	Shacked
53269	0CP	1	2017-11-05 08:00:00	2017-11-05 08:05:00	Request product	clerk	1c	product	0p	process activity	
53269	0CP	2	2017-11-05 08:10:15	2017-11-05 08:10:40	Assign robot	robot	1r	shelf	7s	process activity	
53269	0RP	10	2017-11-05 08:11:00	2017-11-05 08:13:00	Putdown current shelf	robot	1r	shelf	5s	process activity	
53269	0RP	11	2017-11-05 08:14:00	2017-11-05 08:17:23	Go to appropriate shelf	robot	1r	shelf	7s	process activity	
53269	0RP	12	2017-11-05 08:18:00	2017-11-05 08:23:00	Move shelf to dock	robot	1r	shelf	7s	process activity	
53269	41	41	2017-11-05 08:24:45	2017-11-05 08:24:45	collector				0p		
53269	0CP	3	2017-11-05 08:25:00	2017-11-05 08:30:00	Check if the product was checked	clerk	1c	product	0p	process activity	
53269	42	42	2017-11-05 08:31:45	2017-11-05 08:31:45	packageDone				0p		
53269	0CP	4	2017-11-05 08:32:00	2017-11-05 08:32:15	Product Delivered	clerk	1c	product	0p	process activity	
53269	0RP	13	2017-11-05 08:37:31	2017-11-05 08:42:00	Move shelf from dock	robot	1r	shelf	7s	process activity	
70923	2CP	35	2017-11-05 08:51:00	2017-11-05 08:53:00	Request product	clerk	1c	product	2p	process activity	
70923	2CP	36	2017-11-05 08:54:00	2017-11-05 08:55:00	Assign robot	robot	3r	shelf	61s	process activity	
70923	2RP	210	2017-11-05 08:56:00	2017-11-05 08:57:00	Putdown current shelf	robot	3r	shelf	75s	process activity	
70923	2RP	211	2017-11-05 09:00:00	2017-11-05 09:10:00	Go to appropriate shelf	robot	3r	shelf	61s	process activity	
70923	2RP	212	2017-11-05 09:11:00	2017-11-05 09:25:00	Move shelf to dock	robot	3r	shelf	61s	process activity	
70923	2RP	213	2017-11-05 09:30:31	2017-11-05 09:35:00	Move shelf from dock	robot	3r	shelf	61s	process activity	
70923	57	57	2017-11-05 09:39:35	2017-11-05 09:39:35	collector				2p		
70923	2CP	37	2017-11-05 09:40:00	2017-11-05 09:50:00	Check if the product was checked	clerk	1c	product	2p	process activity	
70923	58	58	2017-11-05 09:52:25	2017-11-05 09:52:25	packageDone				2p		
70923	2CP	38	2017-11-05 09:53:00	2017-11-05 09:57:00	Product Delivered	clerk	1c	product	2p	process activity	
92494	1CP	5	2017-11-05 08:02:00	2017-11-05 08:04:30	Request product	clerk	1c	product	1p	process activity	
92494	1CP	6	2017-11-05 08:11:15	2017-11-05 08:12:20	Assign robot	robot	2r	shelf	32s	process activity	
92494	1RP	18	2017-11-05 08:12:00	2017-11-05 08:13:11	Putdown current shelf	robot	2r	shelf	24s	process activity	
92494	1RP	19	2017-11-05 08:14:15	2017-11-05 08:22:10	Go to appropriate shelf	robot	2r	shelf	32s	process activity	
92494	1RP	20	2017-11-05 08:23:00	2017-11-05 08:35:00	Move shelf to dock	robot	2r	shelf	32s	process activity	Shacked
92494	1RP	21	2017-11-05 08:25:00	2017-11-05 08:25:05	Reduce Speed	robot	2r	shelf	32s	process activity	
92494	55	55	2017-11-05 08:43:30	2017-11-05 08:43:30	collector				1p		
92494	1RP	22	2017-11-05 08:43:31	2017-11-05 08:48:00	Move shelf from dock	robot	2r	shelf	32s	process activity	
92494	1CP	7	2017-11-05 08:44:00	2017-11-05 08:50:00	Check if the product was checked	clerk	1c	product	1p	process activity	
92494	56	56	2017-11-05 08:54:01	2017-11-05 08:54:01	packageDone				1p		
92494	1CP	8	2017-11-05 08:55:00	2017-11-05 08:59:15	Product Delivered	clerk	1c	product	1p	process activity	

Table 5.14: Final event log generated by the ADSM approach (using the running example Section 5.3.2)

5.4 Implementation

The ADSM approach is implemented as a ProM 6 plugin. ProM offers a Process Mining open-source framework allowing to implement new algorithms as plugins that can be quickly loaded to the ProM work-space. The framework provides a tools kit to explore several process mining features in an easy-to-use graphical user interface [41]. Furthermore, ProM plugins can be used in chain such that the output provided by one plugin is reused as input for another plugin.

The event log obtained from the ADSM approach is converted to XES format and visualized by the XES plugin (Available in ProM 6) which provides a compact dashboard allowing to inspect log cases at the individual level and at the trace variants level. Moreover, the XES file can be used by a wide range of process discovery plugins enabling to generate the corresponding process model using different discovery algorithms (i,e Inductive miner [27], Genetic miner [45], Heuristic miner [49], Alpha Algorithm [3], Fuzzy miner [2], ILP-based miner [45]).

Figure 5.2 depicts the ADSM plugin architecture. The plugin uses the *CSV Importer* component to import Comma-Separated-Files (CSV) into ProM work-space. Using the plugin graphical interface *UI* component, the user is prompt to choose the log type corresponding to each CSV file (i.e., event log, active sensor data log, decorative sensor data log), and the merging parameters. The *UI* component interacts with the *ICI plugin* introduced in Chapter 4 to automatically infer case ids from event logs. Afterwards, the *Logs Processor* component process the log files based on their type and populate a *Data Model* to generate the data objects to be used by the *Merger & Mapper* component. The techniques described in Section 5.3 namely Temporal Relations Rules (Section 5.3.3.1), Text Mining and Similarity Scoring (Section 5.3.3.2), Hierarchical Merging (Section 5.3.3.4) , Decorative Sensor Data Mapping (Section 5.3.4) are implemented as *Temporal Relations Checker*, *Text Miner*, *Hierarchical Merger*, and *Intersections Mapper* in the *Merger & Mapper* component respectively. The output is then processed by the *Log Generator* component responsible for generating an XES log file compatible with a variety of process discovery plugins.

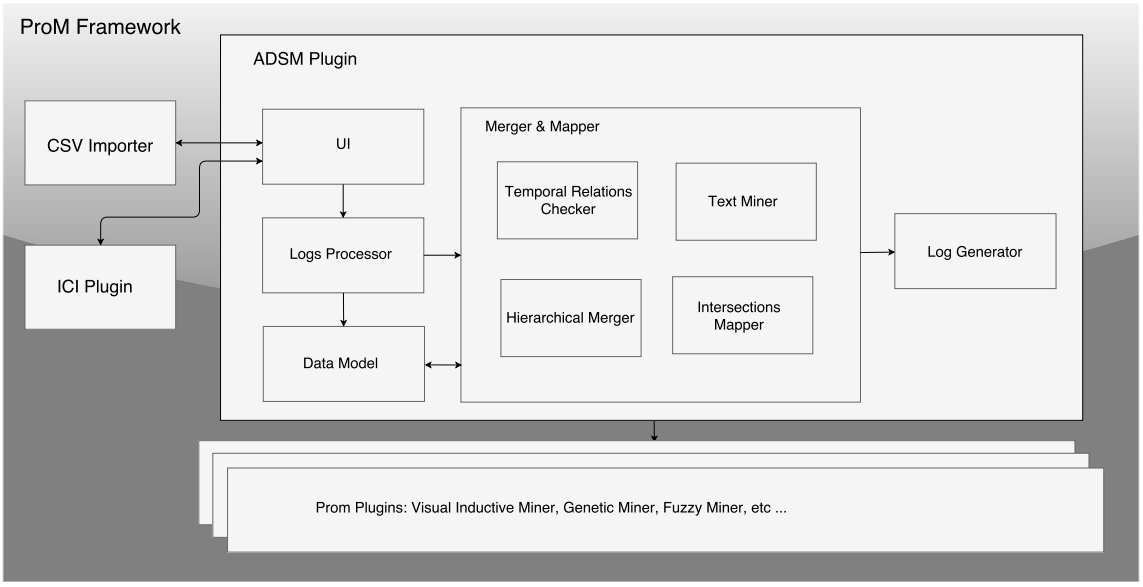


Figure 5.2: Overview on ADSM plugin Architecture

Figures 5.3, 5.4, 5.5 depicts screen-shots of the Active and Decorative Sensor Mapping Plugin developed as part of this thesis project. The plugin allows for an infinite number of logs file. The Merging and Mapping Parameters Panel (Figure 5.4) prompt the user to choose the similarity function to be used and the main log file. Moreover, for each log file, the plugin prompts the user to specify its type, then based on the type, extra parameters (i.e., timestamp, and case id) are required. To ensure a high accuracy of the ADSM approach, inferring case id is made optional to the user. In other words, the user can either input the case id index manually or infer it automatically by calling the ICI plugin (Chapter 4). In case the user chooses to infer the case id automatically, he/she has to input the event name index of the corresponding log file.

The implementation of the ADSM plugin, and an executable version of ProM with ADSM plugin are available at the Github Repository `https://github.com/aminobest/ActiveAndDecorativeSensorMerging/`.

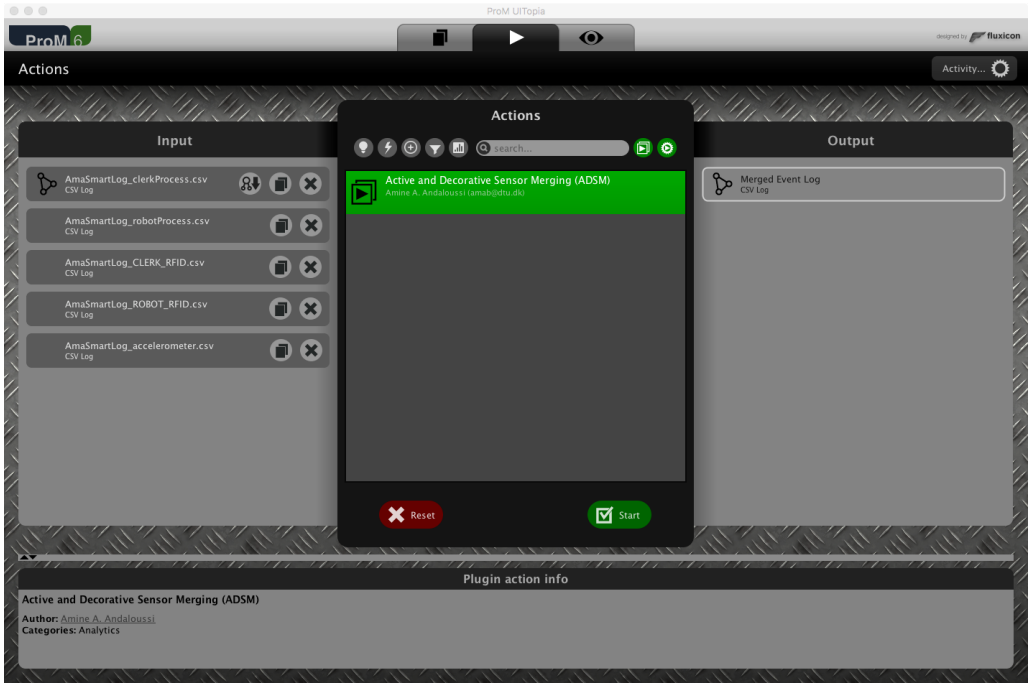


Figure 5.3: Entry point of the ADSM plugin

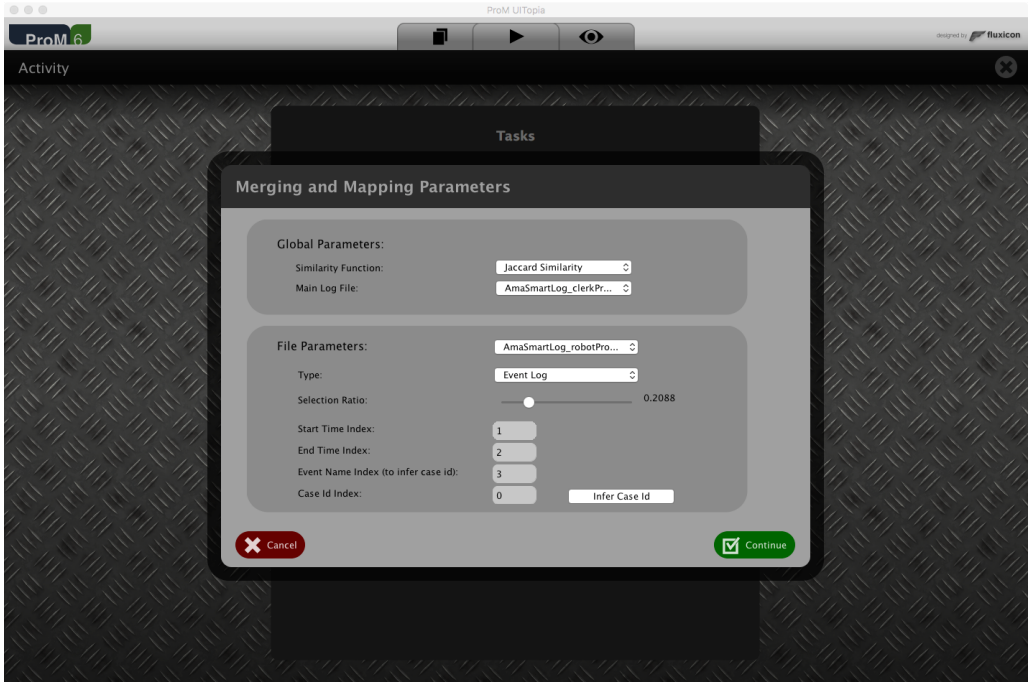
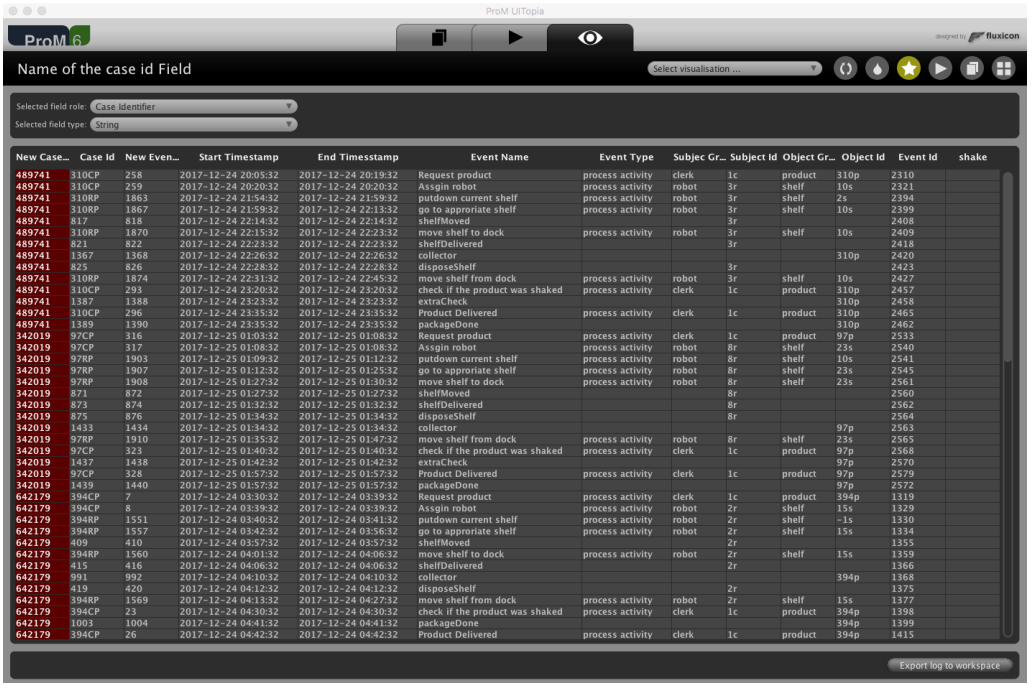


Figure 5.4: Merging and Mapping Parameters Panel of the ADSM plugin



New Case...	Case Id	New Even...	Start Timestamp	End Timesstamp	Event Name	Event Type	Subjec Gr...	Subject Id	Object Gr...	Object Id	Event Id	shake
489741	310CP	258	2017-12-24 20:05:32	2017-12-24 20:19:32	Request product	process activity	clerk	1c	product	310p	2310	
489741	310CP	259	2017-12-24 20:20:32	2017-12-24 20:20:32	Assgin robot	process activity	robot	3r	shelf	10s	2321	
489741	310RP	1863	2017-12-24 21:54:32	2017-12-24 21:59:32	putdown current shelf	process activity	robot	3r	shelf	2s	2394	
489741	310RP	1867	2017-12-24 21:59:32	2017-12-24 22:13:32	go to approriate shelf	process activity	robot	3r	shelf	10s	2399	
489741	817	818	2017-12-24 22:14:32	2017-12-24 22:14:32	shelfMoved			3r			2408	
489741	310RP	1870	2017-12-24 22:15:32	2017-12-24 22:23:32	move shelf to dock	process activity	robot	3r	shelf	10s	2409	
489741	821	822	2017-12-24 22:23:32	2017-12-24 22:23:32	shelfDelivered			3r			2418	
489741	1367	1368	2017-12-24 22:26:32	2017-12-24 22:26:32	collector					310p	2420	
489741	825	826	2017-12-24 22:28:32	2017-12-24 22:28:32	disposeShelf			3r			2423	
489741	310RP	1874	2017-12-24 22:31:32	2017-12-24 22:45:32	move shelf from dock	process activity	robot	3r	shelf	10s	2427	
489741	310CP	293	2017-12-24 23:20:32	2017-12-24 23:20:32	check if the product was shaked	process activity	clerk	1c	product	310p	2457	
489741	1387	1388	2017-12-24 23:23:32	2017-12-24 23:23:32	extraCheck					310p	2458	
489741	310CP	296	2017-12-24 23:35:32	2017-12-24 23:35:32	Product Delivered	process activity	clerk	1c	product	310p	2465	
489741	1389	1390	2017-12-24 23:35:32	2017-12-24 23:35:32	packageDone					310p	2462	
342019	97CP	316	2017-12-25 01:03:32	2017-12-25 01:08:32	Request product	process activity	clerk	1c	product	97p	2533	
342019	97CP	317	2017-12-25 01:08:32	2017-12-25 01:08:32	Assgin robot	process activity	robot	8r	shelf	23s	2540	
342019	97RP	1903	2017-12-25 01:09:32	2017-12-25 01:12:32	putdown current shelf	process activity	robot	8r	shelf	10s	2541	
342019	97RP	1907	2017-12-25 01:12:32	2017-12-25 01:25:32	go to approriate shelf	process activity	robot	8r	shelf	23s	2545	
342019	97RP	1908	2017-12-25 01:27:32	2017-12-25 01:30:32	move shelf to dock	process activity	robot	8r	shelf	23s	2561	
342019	871	872	2017-12-25 01:27:32	2017-12-25 01:27:32	shelfMoved			8r			2560	
342019	873	874	2017-12-25 01:32:32	2017-12-25 01:32:32	shelfDelivered			8r			2562	
342019	875	876	2017-12-25 01:34:32	2017-12-25 01:34:32	disposeShelf			8r			2564	
342019	1433	1434	2017-12-25 01:34:32	2017-12-25 01:34:32	collector					97p	2563	
342019	97RP	1910	2017-12-25 01:35:32	2017-12-25 01:47:32	move shelf from dock	process activity	robot	8r	shelf	23s	2565	
342019	97CP	323	2017-12-25 01:40:32	2017-12-25 01:40:32	check if the product was shaked	process activity	clerk	1c	product	97p	2568	
342019	1437	1438	2017-12-25 01:42:32	2017-12-25 01:42:32	extraCheck					97p	2570	
342019	97CP	328	2017-12-25 01:57:32	2017-12-25 01:57:32	Product Delivered	process activity	clerk	1c	product	97p	2579	
342019	1439	1440	2017-12-25 01:57:32	2017-12-25 01:57:32	packageDone					97p	2572	
642179	394CP	7	2017-12-24 03:30:32	2017-12-24 03:39:32	Request product	process activity	clerk	1c	product	394p	1319	
642179	394CP	8	2017-12-24 03:39:32	2017-12-24 03:39:32	Assgin robot	process activity	robot	2r	shelf	15s	1329	
642179	394RP	1551	2017-12-24 03:40:32	2017-12-24 03:41:32	putdown current shelf	process activity	robot	2r	shelf	1s	1330	
642179	394RP	1557	2017-12-24 03:42:32	2017-12-24 03:56:32	go to approriate shelf	process activity	robot	2r	shelf	15s	1334	
642179	409	410	2017-12-24 03:57:32	2017-12-24 03:57:32	shelfMoved			2r			1355	
642179	394RP	1560	2017-12-24 04:01:32	2017-12-24 04:06:32	move shelf to dock	process activity	robot	2r	shelf	15s	1359	
642179	415	416	2017-12-24 04:06:32	2017-12-24 04:06:32	shelfDelivered			2r			1366	
642179	991	992	2017-12-24 04:10:32	2017-12-24 04:10:32	collector					394p	1368	
642179	419	420	2017-12-24 04:12:32	2017-12-24 04:12:32	disposeShelf			2r			1375	
642179	394RP	1569	2017-12-24 04:13:32	2017-12-24 04:27:32	move shelf from dock	process activity	robot	2r	shelf	15s	1377	
642179	394CP	23	2017-12-24 04:30:32	2017-12-24 04:30:32	check if the product was shaked	process activity	clerk	1c	product	394p	1398	
642179	1003	1004	2017-12-24 04:41:32	2017-12-24 04:41:32	packageDone					394p	1399	
642179	394CP	26	2017-12-24 04:42:32	2017-12-24 04:42:32	Product Delivered	process activity	clerk	1c	product	394p	1415	

Figure 5.5: Output of the ADSM plugin

5.5 Evaluation

The evaluation of the ADSM approach is performed according to a ground-truth file generated by the BPS. The ground truth file gathers the events and data entries belonging to the same process execution together regardless of their parent log files.

The ADSM approach was tested on several data sets generated by the BPS. Each dataset contains log files similar to those depicted in Figures 3.3, and 3.4 (cf. Chapter 3). To evaluate the accuracy of the ADSM approach on different scales, the BPS generated ten random datasets simulating the work-flow of 20 cases, ten random datasets simulating the work-flow of 100 cases, and ten random datasets simulating the work-flow of 1000 cases. The reason for choosing multiple random data sets for each log size is to obtain an average accuracy. Since each data set has its own random characteristics, the merging and mapping algorithms are expected to face different merging scenarios.

The tests were executed in batch, the BPS, the ADSM plugin, and an Accuracy Checker were exported as Runnable JAR files, then run from a Shell Script file. The simulator generates a random data set, then the ADSM plugin applies the merging and mapping approach, and finally, the Accuracy Checker compares the ADSM output to the ground truth file generated by the BPS. The selection ratio used is 0.2, and the system configuration is 12GB of RAM, 1 processor with 4 cores.

The Runnable Jar files, the generated data sets, and ground truth files are available at the Github Directory `https://github.com/aminobest/ActiveAndDecorativeSensorMerging/tree/master/ADSM%20evaluation` .

		Cases Correctly Merged	Cases Wrongly Merged	Accuracy
Run 1	Jaccard	15	5	0.750
	TF-IDF	16	4	0.800
Run 2	Jaccard	14	6	0.700
	TF-IDF	16	4	0.800
Run 3	Jaccard	20	0	1.000
	TF-IDF	20	0	1.000
Run 4	Jaccard	12	8	0.600
	TF-IDF	14	6	0.700
Run 5	Jaccard	13	7	0.650
	TF-IDF	13	7	0.650
Run 6	Jaccard	14	6	0.700
	TF-IDF	15	5	0.750
Run 7	Jaccard	15	5	0.750
	TF-IDF	16	4	0.800
Run 8	Jaccard	18	2	0.900
	TF-IDF	18	2	0.900
Run 9	Jaccard	9	11	0.450
	TF-IDF	14	6	0.700
Run 10	Jaccard	11	9	0.550
	TF-IDF	18	2	0.900

Table 5.15: Merging accuracy of data sets containing 20 cases each using the ADSM approach. Average accuracy using Jaccard similarity function is 0.705, and average accuracy using TF-IDF similarity function is 0.800

		Cases Correctly Merged	Cases Wrongly Merged	Accuracy
Run 1	Jaccard	97	3	0.970
	TF-IDF	97	3	0.970
Run 2	Jaccard	73	27	0.730
	TF-IDF	99	1	0.990
Run 3	Jaccard	94	6	0.940
	TF-IDF	94	6	0.940
Run 4	Jaccard	90	10	0.900
	TF-IDF	100	0	1.000
Run 5	Jaccard	99	1	0.990
	TF-IDF	99	1	0.990
Run 6	Jaccard	98	2	0.980
	TF-IDF	98	2	0.980
Run 7	Jaccard	95	5	0.950
	TF-IDF	96	4	0.960
Run 8	Jaccard	77	23	0.770
	TF-IDF	95	5	0.950
Run 9	Jaccard	84	16	0.840
	TF-IDF	100	0	1.000
Run 10	Jaccard	81	19	0.810
	TF-IDF	99	1	0.990

Table 5.16: Merging accuracy of data sets containing 100 cases each using the ADSM approach. Average accuracy using Jaccard similarity function is 0.888, and average accuracy using TF-IDF similarity function is 0.977

		Cases Correctly Merged	Cases Wrongly Merged	Accuracy
Run 1	Jaccard	880	120	0.880
	TF-IDF	1000	0	1.000
Run 2	Jaccard	996	4	0.996
	TF-IDF	998	2	0.998
Run 3	Jaccard	989	11	0.989
	TF-IDF	995	5	0.995
Run 4	Jaccard	887	113	0.887
	TF-IDF	997	3	0.997
Run 5	Jaccard	713	287	0.713
	TF-IDF	998	2	0.998
Run 6	Jaccard	782	218	0.782
	TF-IDF	999	1	0.999
Run 7	Jaccard	999	1	0.999
	TF-IDF	1000	0	1.000
Run 8	Jaccard	783	217	0.783
	TF-IDF	1000	0	1.000
Run 9	Jaccard	888	112	0.888
	TF-IDF	1000	0	1.000
Run 10	Jaccard	916	84	0.916
	TF-IDF	1000	0	1.000

Table 5.17: Merging accuracy of data sets containing 1000 cases each using the ADSM approach. Average accuracy using Jaccard similarity function is 0.883, and average accuracy using TF-IDF similarity function is 0.999

Tables 5.15, 5.16, 5.17 show the merging accuracy of data sets containing 20, 100, and 1000 cases respectively. The results demonstrate a high accuracy of the ADSM approach using both Jaccard similarity and TF-IDF similarity functions. The results also show that the accuracy is higher with larger log sizes, which can be interpreted by the fact that the text mining approaches described in Section 5.3.3.2 provide better accuracy with the availability of more data. By considering different data sets with different log sizes the average accuracy using Jaccard similarity function and the average accuracy using TF-IDF similarity function as shown in Table 5.18 are is 0.825 and 0.925 respectively.

	20 Cases	100 Cases	1000 Cases	Average Accuracy
Jaccard	0.705	0.888	0.883	0.825
TF-IDF	0.800	0.977	0.999	0.925

Table 5.18: Average ADSM approach accuracy obtained from the results reported in Tables 5.15, 5.16, 5.17

5.6 Discussion

The results reported in Section 5.5 demonstrate a high accuracy in merging event logs with sensor data. Both Jaccard similarity function and TF-IDF Similarity function provide reliable accuracy in the merging and mapping process. As shown in table 5.18, the TF-IDF function outperforms the Jaccard similarity function in accuracy by an average of 10%.

By integrating the ADSM approach in ProM framework, it becomes easier to discover the process

model corresponding to the event log generated by the approach. In this context, the event log generated from Run 4 using TF-IDF similarity function[4] (cf. Table 5.16) was provided as input to the *Mine for a Fuzzy Model* Plugin[5], the resulting process model is depicted in Figure 5.6.

By comparing the original use case scenario model depicted in Figure 1.2 (cf. Chapter 1) with the Fuzzy Miner model depicted in Figure 5.6, one would notice a similar control-flow. Note that the activity *Record shake for affected products* from the original use case scenario model was not recorded by the BPS for sake of simplicity. Furthermore, some events are recorded in different orders by the BPS. This design decision was made explicit for the purpose of simulating distributed systems where not all events are synchronized. For instance, the activities *Check if the product was shacked* in the *Clerk Process*, and *Move shelf from clerk dock* in *Robot Process* (cf. Figure 1.2, Chapter 1) do not necessarily have to synchronize. In other words, *Move shelf from clerk dock* can happen either before or after *Check if the product was shacked* as long as the product is collected. Furthermore, since sensors are assumed to respond instantly, some sensor data entries and log events are expected to have similar timestamps. For instance, the sensor data entry *Dispose Shelf* (cf. Table 3.4 Chapter 3), and the event *Move shelf from clerk dock* (cf. Figure 1.2, Chapter 1) usually have the same timestamp.

To emphasis on such cases the *Inductive Visual Miner* Plugin[6] was used to mine the process model corresponding to the same event log. Unlike the Fuzzy Miner the Inductive Visual Miner is aware of concurrency, the resulting process model is depicted in Figure 5.7[7]. From the process model, it is clear that the events with the same timestamp such as *shelfmoved* and *move shelf to dock* appear as concurrent events. Also the events *Collector* and *move shelf from dock*, and *disposeShelf* appear as concurrent events which solves the lack of synchronization issue discussed previously.

[4]Available Online as XES file at `https://github.com/aminobest/ActiveAndDecorativeSensorMerging/tree/master/ADSM%20evaluation/run4_100cases_TFIDF_mergedlog`.

[5]See `http://www.processmining.org/online/fuzzyminer`. Available in ProM 6

[6]Available as part of the XES plugin in ProM 6

[7]The Figure is a Vector Image File, that can be zoomed-in on the electronic version of this document for better readability

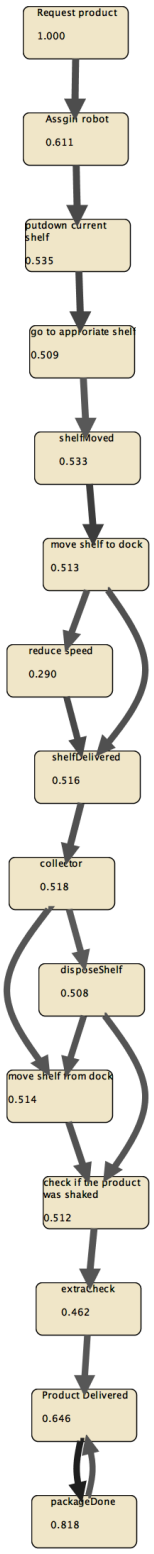


Figure 5.6: Fuzzy Miner model obtained from the event log generated from Run 4 using TF-IDF similarity function (cf. Table 5.16).

Figure 5.7: Inductive Visual Miner Process model obtained from the event log generated from Run 4 using TF-IDF similarity function (cf. Table 5.16) (zoom-in on the electronic version of this document for better readability).

5.7 Conclusion

The ADSM approach demonstrates high accuracy in solving the data mapping and merging challenge in IoT environments. The approach extends existing works related to event log merging, and sensor data mapping present in the literature. It proposes different similarity functions and provides a comprehensive approach allowing for the hierarchical merging of multiple event logs. The event log scheme identification approach presented in Chapter 4 is deployed in the context of this chapter to automate the case id identification from event logs. Furthermore, The ADSM approach allowed to map decorative sensor data to log events successfully using the multi-dimensional framework introduced in Section 5.3.4 to locate the intersections between decorative data entries and events according to pre-defined dimensions.

This chapter answers to the primary research problem presented in Chapter 1, the solution is developed as a ProM plugin. By implementing the ADSM approach in the ProM framework, it becomes possible to use the generated XES file as input for a variety of ProM plugins, thus exploring a wide range of process mining capabilities such as control-flow discovery and conformance checking. The ADSM plugin will be available soon in the ProM package manager after being tested to ensure its robustness.

The process models depicted in the Evaluation Section 5.5 show similar control-flow to the original use case model depicted in Figure 1.2 (cf. Chapter 1). However, it is necessary to note that the ADSM approach was tested only on synthetic log files generated by the BPS; thus, it is still necessary to try-out the approach on real-world log files, which is kept for future work.

Chapter 6

Conclusion

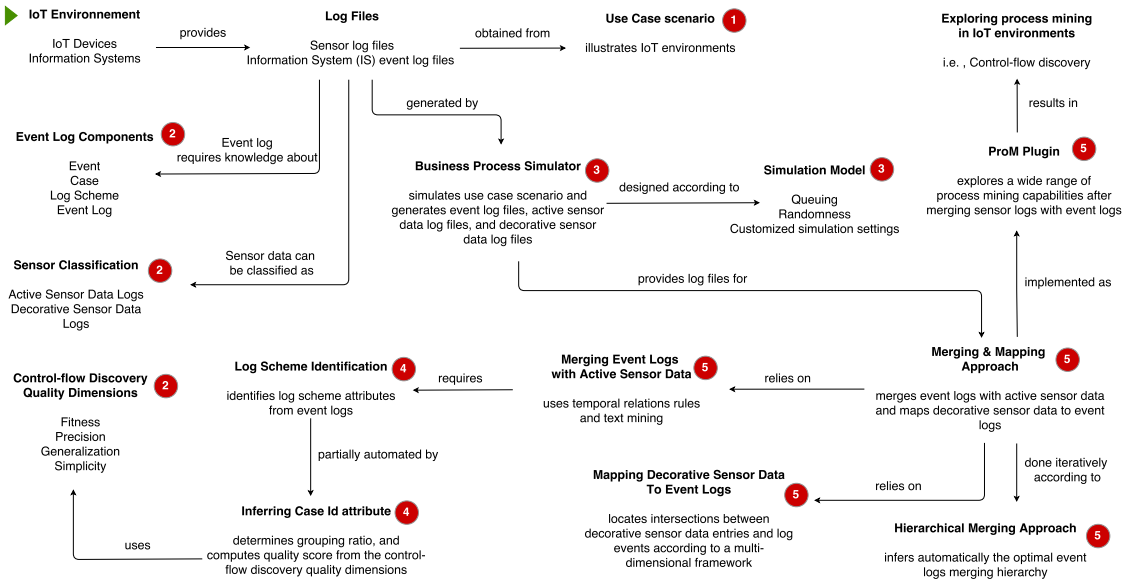


Figure 6.1: Detailed scope of this thesis project. The green triangle refers to the starting point of the chart, and the red circles show the chapter numbers discussing the corresponding topics.

The bottom-up approach described in Chapter 1 relies on process mining capabilities to bridge the gap between the IoT world and the BPM world. Figure 6.1 depicts a comprehensive scope including all the concepts and techniques discussed in this report. It starts from an IoT environment comprising IoT devices and information system interacting together in a distributed environment. The huge amount of data recorded by IoT sensors, and the general business process control-flow recorded by information systems supporting the IoT environment opens-up new challenges and opportunities for exploring process mining in a different setting.

As mentioned in Chapter 1 obtaining real-world logs files illustrating the interactions between IoT devices and information systems in an IoT environment is still a challenging task. Therefore, it was

mandatory to develop a use case scenario to illustrate the work-flow within an IoT environment. As source of inspiration the Kiva Robots deployed in Amazon warehouses were considered.

Exploring process Mining in IoT environments requires an advanced knowledge about process mining. To bring process mining to the IoT context, a new process mining scope was presented in Chapter 2. In this scope business processes are supported by both information systems and IoT devices allowing for a better and efficient business process management. In this new context, the challenge of merging sensor data with event logs raised.

As described in Chapter 2 IoT sensors have different characteristics. Therefore, by identifying these characteristics and categorizing sensor data accordingly, it becomes possible to treat them independently to design a robust approach allowing to map and merge sensor data with event logs.

The use case scenario introduced in Chapter 1 was simulated using the BPS proposed in Chapter 3. Despite the availability of business process simulators in the literature, it was necessary to develop a new business process simulator, because none of the existing simulators allows to mimic the interactions between sensors and process instances in a BPM context. As result, a new dynamic, stochastic, and continuous simulation model was designed and implemented in Java. Although its usage is limited to the use case scenario, it offers a variety of simulation characteristics such as queuing and randomness.

The log files generated by the simulator are used to evaluate the ADSM approach presented in Chapter 5. The aim of this approach is to build a robust algorithm allowing to merge event logs with active sensor data and to map decorative sensor data to event logs. Merging events logs with active sensor data is performed according to a set of temporal relations rules allowing to infer events and data entries happening within the same time interval efficiently, and text mining approaches presented in the context of similarity functions allowing to measure the extends two which events have similar attributes. The ADSM approach allows for an efficient hierarchical merging by automatically finding the optimal merging hierarchy that maximizes the similarity score between sensor data entries and log events. The approach was implemented as a ProM plugin and tested on several synthetic data sets generated by the BPS. Finally, an event log obtained from the ADSM plugin was used by two control-flow discovery plugins to discover and analyze the corresponding process model.

As complement to the ADSM approach, this work investigates a common challenge in the process mining community that is to identify event log schemes. Chapter 4 proposes a new technique to automatically infer case ids form event logs by exploring the control-flow discovery quality dimensions described in Chapter 2. The suggested approach was successful in demonstrating a high accuracy in inferring case ids from several real-world event logs.

Future work is planned on the following three main areas: Business process simulation, event log scheme identification, and sensor data mapping and merging. For the business process simulation area, the BPS provides an initial platform by developing a simulation model allowing to imitate the interactions between sensors and process instances. As next step the simulation model should be extended to cope with any business process, such that the user can draw or import her/his process model (i.e., as BPMN model) within the BPS framework in run-time. Concerning the event log scheme identification area, the ICI approach should be extended to allow inferring other event attributes such as the event name, and implement other heuristics to avoid overhead memory problems for the purpose to ensure a generic well-functioning of the approach. Finally, in the sensor data mapping and merging area, the ADSM approach has to be tried-out on real-world logs to evaluate its efficiency on a larger scale.

Bibliography

[1] Replaying History on Process Models for Conformance Checking and Performance Analysis. (1):1–18.

[2] V. D. Aalst. Fuzzy mining - adaptive process simplification based on multi-perspective metrics. 4714(Bpm 2007):24–28, 2007.

[3] W. M. P. V. D. Aalst, A. J. M. M. Weijters, and L. Maruster. Workflow Mining : Discovering process models from event logs.

[4] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, nov 1983.

[5] A. Aztiria, J. C. Augusto, R. Basagoiti, A. Izaguirre, and D. J. Cook. Discovering frequent user environment interactions in intelligent environments. *Personal and Ubiquitous Computing*, 16(1):91–103, jan 2012.

[6] A. S. B, A. Rogge-solti, A. Gal, J. Mendling, and A. Mandelbaum. The ROAD from Sensor Data to Process Instances via Interaction Mining. 1:257–273, 2016.

[7] O. Banos, J.-M. Galvez, M. Damas, H. Pomares, and I. Rojas. Window size impact in human activity recognition. *Sensors*, 14(4):6474–6499, 2014.

[8] D. Bayomie, I. M. Helal, A. Awad, E. Ezat, and A. ElBastawissi. Deducing case IDs for unlabeled event logs. *Lecture Notes in Business Information Processing*, 256:242–254, 2016.

[9] J. C. a. M. Buijs. *Flexible Evolutionary Algorithms for Mining Structured Process Models*. 2014.

[10] A. Burattin. Plg2: Multiperspective process randomization with online and offline simulations. *Proceedings of the BPM Demo Track 2016 Co-located with the 14th International Conference on Business Process Management BPM 2016), Rio de Janeiro, Brazil, September 21, 2016.*, 2:1–6, 2016.

[11] A. Burattin and A. Sperduti. PLG: A Framework for the Generation of Business Process Models and Their Execution Logs. pages 214–219. 2011.

[12] A. Burattin and R. Vigo. A framework for semi-automated process instance discovery from decorative attributes. In *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 176–183. IEEE, apr 2011.

[13] Christian Walter Gunther. *Process Mining in Flexible Environments*. University Press Facilities, Eindhoven, 2009.

[14] J. Claes and G. Poels. Merging computer log files for process mining: An artificial immune system technique. *CEUR Workshop Proceedings*, 800:49–50, 2011.

[15] J. Claes and G. Poels. Merging event logs for process mining: A rule based merging method and rule suggestion algorithm. *Expert Systems with Applications*, 41(16):7291–7306, 2014.

[16] D. J. Cook, N. C. Krishnan, and P. Rashidi. Activity Discovery and Activity Recognition : A New Partnership. 43(3):820–828, 2013.

[17] M. Dimaggio, F. Leotta, M. Mecella, D. Sora, I. Informatica, and A. Gestionale. Process-Based Habit Mining : Experiments and Techniques. 2016.

[18] M. Dumas, M. Rosa, J. Mendling, and H. Reijers. *Fundamentals of Business Process Management*. Springer Berlin Heidelberg, 2013.

[19] M. L. V. Eck, N. Sidorova, and W. M. P. V. D. Aalst. Enabling Process Mining on Sensor Data from Smart Products. In *Research Challenges in Information Science (RCIS), 2016*.

[20] D. R. Ferreira and D. Gillblad. Discovering Process Models from. *Bpm*, pages 143–158, 2009.

[21] K. Figl. Comprehension of Procedural Visual Business Process Models: A Literature Review. *Business and Information Systems Engineering*, 59(1):41–67, 2017.

[22] P. Fishwick. *Handbook of Dynamic System Modeling*. Chapman & Hall/CRC Computer and Information Science Series. CRC Press, 2007.

[23] C. Janiesch, A. Koschmider, M. Mecella, B. Weber, A. Burattin, C. Di Ciccio, A. Gal, U. Kannengiesser, F. Mannhardt, J. Mendling, A. Oberweis, M. Reichert, S. Rinderle-Ma, W. Song, J. Su, V. Torres, M. Weidlich, M. Weske, and L. Zhang. The Internet-of-Things Meets Business Process Management: Mutual Benefits and Challenges. pages 1–9, 2017.

[24] K. Jensen. *Coloured Petri Nets*. Monographs in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[25] D. E. Knuth, O.-J. Dahl, and I. Nygaard. SIMULA an ALGOL-Based Simulation Language. *Comunication of the ACM*, 9(9):671–678, 1966.

[26] M. Laguna and J. Marklund. *Business Process Modeling, Simulation, and Design*. Education Pearson, 2005.

[27] S. J. Leemans, D. Fahland, and W. M. Van Der Aalst. Process and deviation exploration with inductive visual miner. *CEUR Workshop Proceedings*, 1295:46–50, 2014.

[28] S. J. J. Leemans. *Robust process mining with guarantees*. SIKS Dissertation Series No. 2017-12, 2017.

[29] J. Leskovec, A. Rajaraman, and J. D. Ullman. Mining of Massive Datasets. 2014.

[30] C. F. L. Lewis. EE 5322 : Intelligent Control Systems: Petri nets. pages 1–19, 2004.

[31] Y. Liu and W.-j. Ye. Time Consuming Numerical Model Calibration Using Genetic Algorithm (GA), 1-Nearest Neighbor (1NN) Classifier and Principal Component Analysis (PCA) 1. pages 1208–1211, 2005.

[32] D. L. Moody. Association for Information Systems AIS Electronic Library (AISeL) The Method Evaluation Model: A Theoretical Model for Validating Information Systems Design Methods The Method Evaluation Model: A Theoretical Model for Validating Information Systems Design Methods. 2003.

[33] E. H. J. Nooijen, B. F. van Dongen, and D. Fahland. Automatic Discovery of Data-Centric and Artifact-Centric Processes. *Business Process Management Workshops*, 132:316–327, 2012.

[34] M. Polato, A. Sperduti, A. Burattin, and M. de Leoni. Time and Activity Sequence Prediction of Business Process Instances. feb 2016.

[35] L. Raichelson and P. Soffer. Merging Event Logs with Many to Many Relationships. pages 330–341, 2015.

[36] L. Raichelson, P. Soffer, and E. Verbeek. Merging event logs : Combining granularity levels for process flow analysis. *Information Systems*, 71:211–227, 2017.

[37] H. A. Reijers and J. Mendling. A Study Into the Factors That Influence the Understandability of Business Process Models. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 41(3):449–462, 2011.

[38] A. Rozinat and W. M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.

[39] F. Stertz, J. Mangler, and S. Rinderle-Ma. NFC-Based Task Enactment for Automatic Documentation of Treatment Processes. pages 34–48. 2017.

[40] W. Van Der Aalst, A. Adriansyah, A. K. A. De Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. Van Den Brand, R. Brandtjen, J. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani, M. De Leoni, P. Delias, B. F. Van Dongen, M. Dumas, S. Dustdar, D. Fahland, D. R. Ferreira, W. Gaaloul, F. Van Geffen, S. Goel, C. Günther, A. Guzzo, P. Harmon, A. Ter Hofstede, J. Hoogland, J. E. Ingvaldsen, K. Kato, R. Kuhn, A. Kumar, M. La Rosa, F. Maggi, D. Malerba, R. S. Mans, A. Manuel, M. McCreesh, P. Mello, J. Mendling, M. Montali, H. R. Motahari-Nezhad, M. Zur Muehlen, J. Munoz-Gama, L. Pontieri, J. Ribeiro, A. Rozinat, H. Seguel Pérez, R. Seguel Pérez, M. Sepúlveda, J. Sinur, P. Soffer, M. Song, A. Sperduti, G. Stilo, C. Stoel, K. Swenson, M. Talamo, W. Tan, C. Turner, J. Vanthienen, G. Varvaressos, E. Verbeek, M. Verdonk, R. Vigo, J. Wang, B. Weber, M. Weidlich, T. Weijters, L. Wen, M. Westergaard, and M. Wynn. Process mining manifesto. *Lecture Notes in Business Information Processing*, 99 LNBIP(PART 1):169–194, 2012.

[41] W. van der Aalst, P. van den Brand, B. F. van Dongen, D. Fahland, C. W. Gunther, E. Verbeek, and M. Westergaard. The Process Mining Toolkit: ProM. 2013.

[42] W. M. Van Der Aalst. *Process mining: Discovery, Conformance, and Enhancement of Business processes*. Springer International Publishing, 2010.

[43] W. M. Van Der Aalst. Mediating between modeled and observed behavior: The quest for the 'right' process: Keynote. *Proceedings - International Conference on Research Challenges in Information Science*, 2013.

[44] W. M. P. van der Aalst, J. Nakatumba, A. Rozinat, and N. Russell. Business Process Simulation: How to get it right? pages 317–342, 2010.

[45] J. M. E. M. van der Werf, B. F. van Dongen, C. A. J. Hurkens, and A. Serebrenik. *Process Discovery Using Integer Linear Programming*, pages 368–387. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[46] K. M. Van Hee and Z. Liu. Generating benchmarks by random stepwise refinement of Petri nets. *CEUR Workshop Proceedings*, 827:403–417, 2010.

[47] M. Walicki and D. R. Ferreira. Sequence partitioning for process mining with unlabeled event logs. *Data and Knowledge Engineering*, 70(10):821–841, 2011.

[48] M. Weidlich, A. Polyvyanyy, J. Mendling, and M. Weske. Causal behavioural profiles - Efficient computation, applications, and evaluation. *Fundamenta Informaticae*, 113(3-4):399–435, 2011.

[49] A. J. M. M. Weijters, W. M. P. Van Der Aalst, and A. K. A. D. Medeiros. Process Mining with the Heuristics Miner Algorithm. *Technische Universiteit Eindhoven, Tech. Rep. WP*, 166:1–34, 2006.

[50] P. R. Wurman, R. D. Andrea, and M. Mountz. Coordinating Hundreds of Cooperative , Autonomous Vehicles in Warehouses.

[51] Y. Xu, Q. Lin, and M. Q. Zhao. Merging Event Logs for Process Mining with Hybrid Artificial Immune Algorithm. 2:10–16.