

DISCOVERING DATA MODELS FROM EVENT LOGS

DORINA BANO

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

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

DATE OF DEFENSE: ...

August, 2022

Unless otherwise indicated, this work is licensed under a Creative Commons License Attribution 4.0 International.

This does not apply to quoted content and works based on other permissions.

To view a copy of this licence visit:

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

Supervisor: Prof. Dr. Mathias Weske, University of Potsdam

Reviewers:

Prof. Dr. Stefanie Rinderle-Ma, Technical University of Munich, and

Prof. Dr. Matthias Weidlich, Humboldt University of Berlin

Dorina Bano: Discovering Data Models from Event Logs,

© 2022

Published online on the

Publication Server of the University of Potsdam:

<https://doi.org/10.25932/publishup-58542>

<https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-585427>

ABSTRACT

In the last two decades, process mining has developed from a niche discipline to a significant research area with considerable impact on academia and industry. Process mining enables organisations to identify the running business processes from historical execution data. The first requirement of any process mining technique is an event log, an artifact that represents concrete business process executions in the form of sequence of events. These logs can be extracted from the organization's information systems and are used by process experts to retrieve deep insights from the organization's running processes. Considering the events pertaining to such logs, the process models can be automatically discovered and enhanced or annotated with performance-related information. Besides behavioral information, event logs contain domain specific data, albeit implicitly. However, such data are usually overlooked and, thus, not utilized to their full potential.

Within the process mining area, we address in this thesis the research gap of discovering, from event logs, the contextual information that cannot be captured by applying existing process mining techniques. Within this research gap, we identify four key problems and tackle them by looking at an event log from different angles. First, we address the problem of deriving an event log in the absence of a proper database access and domain knowledge. The second problem is related to the under-utilization of the implicit domain knowledge present in an event log that can increase the understandability of the discovered process model. Next, there is a lack of a holistic representation of the historical data manipulation at the process model level of abstraction. Last but not least, each process model presumes to be independent of other process models when discovered from an event log, thus, ignoring possible data dependencies between processes within an organization.

For each of the problems mentioned above, this thesis proposes a dedicated method. The first method provides a solution to extract an event log only from the transactions performed on the database that are stored in the form of redo logs. The second method deals with discovering the underlying data model that is implicitly embedded in the event log, thus, complementing the discovered process model with important domain knowledge information. The third method captures, on the process model level, how the data affects the running process instances. Lastly, the fourth method is about the discovery of the relations between business processes (i.e., how they exchange data) from a set of event logs and explicitly representing such complex interdependencies in a business process architecture.

All the methods introduced in this thesis are implemented as a prototype and their feasibility is proven by being applied on real-life event logs.

ZUSAMMENFASSUNG

In den letzten zwei Jahrzehnten hat sich Process Mining von einer Nischendisziplin zu einem bedeutenden Forschungsgebiet mit erheblichen Auswirkungen auf Wissenschaft und Industrie entwickelt. Process Mining ermöglicht es Unternehmen, die laufenden Geschäftsprozesse anhand historischer Ausführungsdaten zu identifizieren. Die erste Voraussetzung für jede Process-Mining-Technik ist ein Ereignisprotokoll (Event Log), ein Artefakt, das konkrete Geschäftsprozessausführungen in Form einer Abfolge von Ereignissen darstellt. Diese Protokolle (Logs) können aus den Informationssystemen der Unternehmen extrahiert werden und ermöglichen es Prozessexperten, tiefe Einblicke in die laufenden Unternehmensprozesse zu gewinnen. Unter Berücksichtigung der Abfolge der Ereignisse in diesen Protokollen (Logs) können Prozessmodelle automatisch entdeckt und mit leistungsbezogenen Informationen erweitert werden. Neben verhaltensbezogenen Informationen enthalten Ereignisprotokolle (Event Logs) auch domänenspezifische Daten, wenn auch nur implizit. Solche Daten werden jedoch in der Regel nicht in vollem Umfang genutzt. Diese Arbeit befasst sich im Bereich Process Mining mit der Forschungslücke der Extraktion von Kontextinformationen aus Ereignisprotokollen (Event Logs), die von bestehenden Process Mining-Techniken nicht erfasst werden.

Innerhalb dieser Forschungslücke identifizieren wir vier Schlüsselprobleme, bei denen wir die Ereignisprotokolle (Event Logs) aus verschiedenen Perspektiven betrachten. Zunächst befassen wir uns mit dem Problem der Erfassung eines Ereignisprotokolls (Event Logs) ohne hinreichenden Datenbankzugang. Das zweite Problem ist die unzureichende Nutzung des in Ereignisprotokollen (Event Logs) enthaltenen Domänenwissens, das zum besseren Verständnis der generierten Prozessmodelle beitragen kann. Außerdem mangelt es an einer ganzheitlichen Darstellung der historischen Datenmanipulation auf Prozessmodellebene. Nicht zuletzt werden Prozessmodelle häufig unabhängig von anderen Prozessmodellen betrachtet, wenn sie aus Ereignisprotokollen (Event Logs) ermittelt wurden. Dadurch können mögliche Datenabhängigkeiten zwischen Prozessen innerhalb einer Organisation übersehen werden.

Für jedes der oben genannten Probleme schlägt diese Arbeit eine eigene Methode vor. Die erste Methode ermöglicht es, ein Ereignisprotokoll (Event Log) ausschließlich anhand der Historie der auf einer Datenbank durchgeführten Transaktionen zu extrahieren, die in Form von Redo-Logs gespeichert ist. Die zweite Methode befasst sich mit der Entdeckung des zugrundeliegenden Datenmodells, das implizit in dem jeweiligen Ereignisprotokoll (Event Log) eingebettet ist, und ergänzt so-

mit das entdeckte Prozessmodell mit wichtigen, domänenspezifischen Informationen. Bei der dritten Methode wird auf der Ebene des Prozessmodells erfasst, wie sich die Daten auf die laufenden Prozessinstanzen auswirken. Die vierte Methode befasst sich schließlich mit der Entdeckung der Beziehungen zwischen Geschäftsprozessen (d.h. deren Datenaustausch) auf Basis der jeweiligen Ereignisprotokolle (Event Logs), sowie mit der expliziten Darstellung solcher komplexen Abhängigkeiten in einer Geschäftsprozessarchitektur.

Alle in dieser Arbeit vorgestellten Methoden sind als Prototyp implementiert und ihre Anwendbarkeit wird anhand ihrer Anwendung auf reale Ereignisprotokolle (Event Logs) nachgewiesen.

PUBLICATIONS

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

- Dorina Bano and Mathias Weske. “Discovering data models from event logs”. In Gillian Dobbie, Ulrich Frank, Gerti Kappel, Stephen W. Liddle, and Heinrich C. Mayr, editors, *Conceptual Modeling - 39th International Conference, ER 2020, Vienna, Austria, November 3 – 6, 2020, Proceedings*, volume 12400 of *Lecture Notes in Computer Science*, pages 62–76. Springer, 2020. doi : 10.1007/978–3–030–62522–1_5.
- Dorina Bano, Francesca Zerbato, Barbara Weber, and Mathias Weske. “Enhancing discovered process models with data object lifecycles”. In *25th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2021, Gold Coast, Australia, October 25 – 29, 2021*, pages 124–133. IEEE, 2021. doi : 10.1109/EDOC52215.2021.00023.
- Dorina Bano, Adriatik Nikaj, and Mathias Weske. “Discovering business process architectures from event logs”. In Artem Polyvyanyy, Moe Thandar Wynn, Amy Van Looy, and Manfred Reichert, editors, *Business Process Management Forum - BPM Forum 2021, Rome, Italy, September 06 – 10, 2021, Proceedings*, volume 427 of *Lecture Notes in Business Information Processing*, pages 162–177. Springer, 2021. doi : 10.1007/978–3–030–85440–9_10.
- Dorina Bano, Tom Lichtenstein, Finn Klessascheck, and Mathias Weske. “Database-less extraction of event logs from redo logs”. In Witold Abramowicz, Sören Auer, and Elzbieta Lewanska, editors, *24th International Conference on Business Information Systems, BIS 2021, Hannover, Germany, June 15 – 17, 2021*, pages 73–82, 2021. doi : 10.52825/bis.v1i.66.
- Tom Lichtenstein, Dorina Bano, and Mathias Weske. “Attribute-driven case notion discovery for unlabeled event logs”. In Andrea Marrella and Barbara Weber, editors, *Business Process Management Workshops - BPM 2021 International Workshops, Rome, Italy, September 6 – 10, 2021, Revised Selected Papers*, volume 436 of *Lecture Notes in Business Information Processing*, pages 111–122. Springer, 2021. doi : 10.1007/978–3–030–94343–1/9.
- Dorina Bano, Judith Michael, Bernhard Rumpe, Simon Varga, and Mathias Weske. *Process-aware digital twin cockpit synthesis from event logs*. *Journal of Computer Languages*, 70 : 101121, 2022.

Additionally, the author was involved in the following publications, which are not directly related to the topic of this thesis:

- Maximilian König, Leon Bein, Caterina Mandel, Kerstin Andree, Marc Rosenau, Carla Terboven, Stephan Haarmann, Dorina Bano and Mathias Weske. Design-Time Support for Fragment-Based Case Management. DEC2H 2022: 10th International Workshop on DECLarative, DECision and Hybrid approaches to processes, Munster, Germany.
- Leon Bein, Maximilian König, Caterina Mandel, Kerstin Andree, Marc Rosenau, Carla Terboven, Stephan Haarmann, Dorina Bano and Mathias Weske. fcm-js: A Tool for Design-Time Support of Fragment-Based Case Management. BPM (Demos) 2022, Munster, Germany.

ACKNOWLEDGMENTS

During my PhD journey, I have been surrounded by many people who have supported me in several aspects. Therefore, I would like to thank many of them, some of whom were crucial for undertaking this journey.

I would like to express my most profound appreciation to my supervisor, Mathias Weske, for his unlimited support and weekly discussions about my work during the last three years. In addition, I would like to thank him for his continuous encouragement, guidance, and the pleasant working environment.

I would like to thank Matthias Weidlich and Stefanie Rinderle-Ma for reviewing my thesis and for their thoughtful comments and efforts towards improving the structure and the content of this thesis.

In addition, I would like to express my gratitude to all people who collaborated with me during this journey for their fruitful and enjoyable discussions: Prof. Barbara Weber, Prof. Bernhard Rumpe, Francesca Zerbato, Judith Michael, Adriatik Nikaj, Tom Lichtenstein, Finn Klessascheck and all students of the Design-Time Support for Fragment-Based Case Management master project.

Furthermore, I would like to thank all my colleagues at the Business Process Technology group, Stephan Haarmann, Maximilian Völker, Tom Lichtenstein and Jonas Cremerius, for the great time and work or life-related discussions. In addition, I would like to thank them together with Maximilian Koenig for proofreading my thesis.

Last but not least, I would like to extend my sincere thanks to my family for their unwavering support and belief in me. I dedicate this thesis to my son Nolan and my husband Adriatik for their endless love, patience, and encouragement.

CONTENTS

1	INTRODUCTION	1
1.1	Story	2
1.2	Contributions	4
1.3	Structure of the Thesis	7
2	PRELIMINARIES	11
2.1	Business Process Management and Modeling	11
2.2	Data in Business Processes	16
2.2.1	Database's Redo Logs	16
2.2.2	Data Model	17
2.2.3	Data Object	18
2.3	Process Mining	19
2.3.1	Process Mining Position	19
2.3.2	Event Log	21
3	EVENT LOG EXTRACTION FROM REDO LOGS	25
3.1	Motivation	26
3.2	Extraction of Event Logs from Redo Logs	27
3.2.1	Overview of the Approach and Assumptions	27
3.2.2	Discovering the Data Model from Redo Logs	28
3.2.3	Event Log Extraction	31
3.3	Related Work	35
3.4	Evaluation	36
3.4.1	Evaluation Setup	37
3.4.2	Evaluation Results	38
3.5	Summary	43
4	DISCOVERING DATA MODELS FROM EVENT LOGS	45
4.1	Motivation	46
4.2	Overview of the Data Model Discovery Method	46
4.3	Derivation of the Activity Attribute Relationship Diagram	47
4.4	Data Model Discovery	49
4.5	Related Work	54
4.6	Evaluation	56
4.7	Application Scenarios	61
4.7.1	Towards Case Notion Identification from Unlabeled Event Logs	61
4.7.2	Towards Process Aware Digital Twin Generation from the Sensor Data	63
4.8	Summary	64
5	DISCOVERING DATA OBJECT LIFECYCLES FROM EVENT LOGS	67
5.1	Motivation	68
5.2	Scenario	68
5.3	Enhancing the Process Model with Data Object Lifecycles	71

5.3.1	Attribute-Access Event-Log Creation	72
5.3.2	Attribute Change Matrix Creation	72
5.3.3	Discover Data Object Lifecycles	74
5.3.4	Data Object Assignments in the Process Model	77
5.3.5	Holistic View Generation	79
5.4	Related Work	80
5.5	Evaluation	82
5.6	Summary	89
6	DISCOVERING BUSINESS PROCESS ARCHITECTURES FROM EVENT LOGS	91
6.1	Motivation	92
6.2	Business Process Architecture and Event Log Meta-Models	93
6.3	Event Log Awareness	94
6.4	Discovering the Business Process Architecture	96
6.4.1	Trigger Flow	96
6.4.2	Information Flow	98
6.5	The Extension of the BPA's Graphical Representation	101
6.6	Related Work	103
6.7	Evaluation	105
6.8	Summary	109
7	CONCLUSIONS	111
7.1	Contribution Summary	111
7.2	Limitations	115
7.3	Future Work	116
	BIBLIOGRAPHY	119

LIST OF FIGURES

Figure 1	Methods provided to John to support him during his daily work	3
Figure 2	Overview of the contributions presented in this thesis	5
Figure 3	The structure of the thesis	9
Figure 4	The BPM lifecycle [139]	12
Figure 5	A subset of the BPM notions covered in this thesis	13
Figure 6	BPA representing trigger and information flow patterns [50]	14
Figure 7	The visual notations of catching (start/intermediate event) and throwing events (intermediate/end event) in a BPA	15
Figure 8	An example of the data model illustrated as a UML class diagram	18
Figure 9	An example of a process model (represented as BPMN) and the data object/repository associated with its activities	19
Figure 10	Process Mining position [72]	20
Figure 11	An event log example, its cases and the discovered process model	23
Figure 12	Event log extraction from redo log through the data model	25
Figure 13	Overview of the event log extraction method	28
Figure 14	Extraction of tables, attributes and values in each redo log entry	30
Figure 15	Example of an entity relation generation table extracted from the Patients and Admissions table, which are in a foreign key relation	32
Figure 16	An example of extracting the event log cases after applying the entity-based case collection step on the discovered database schema. The Patients table is picked as a root class by the domain expert	34
Figure 17	Evaluation setup based on two synthetic redo logs (MIMIC and Ticket Selling). The dashed lines annotate the compared artifacts	38

- Figure 18 The discovered database schema after applying the first step of our approach. The redo logs are extracted from the MIMIC database 39
- Figure 19 The discovered process model from the extracted event log after applying the method presented in this chapter 40
- Figure 20 The discovered database schema after applying the first step of our approach. The redo logs are taken from the provided data in the literature 42
- Figure 21 Discovering a data model from event log 45
- Figure 22 Overview of the data model discovery method 46
- Figure 23 Derivation process of the activity attribute relationship diagram 47
- Figure 24 An example of the A2A diagram derived from the event log 48
- Figure 25 Rules organized based on two aspects (non/isolated attributes and non/isolated activities) for deriving the data model classes 50
- Figure 26 Decomposed A2A diagram into three fragments after applying the fourth rule 53
- Figure 27 Process model discovered by the Inductive Miner algorithm [76] from the RTFM event log [87] 58
- Figure 28 The A2A diagram generated from the RTFM event log after applying Algorithm 1 presented in Section 4.3 58
- Figure 29 The rules applied to the A2A diagram of the RTFM event log and the resulting data model classes 59
- Figure 30 Resulting data model discovered from the RTFM event log 60
- Figure 31 Two data model application scenarios. In the first scenario (annotated with AS1) the data model is discovered from an unlabeled event log and used to correlate such events with the event log cases. In the second scenario (annotated with AS2) the data model discovered from an event log (extracted from the sensor data) is used to automate the process-aware digital twin generation through the model-to-code transformation approach 62
- Figure 32 Discovering data objects from event log 67

- Figure 33 Process model excerpt represented as BPMN inspired by the process management system within a hospital 69
- Figure 34 An overview of the approach used to discover the data object lifecycles and enhance the discovered process model with such information 71
- Figure 35 An example of Attribute-Change matrix based on the running example. The numbers in the matrix indicates how many times each activity (over)writes a specific event log attribute 73
- Figure 36 Two cases of discovered Attribute-Access behavior 76
- Figure 37 Attribute-change matrix created for the Hospital Billing event log (the color figure is available online) 84
- Figure 38 Discovered data objects and their Attribute-Access behavior for the Hospital Billing event log (the color figure is available online) 85
- Figure 39 Process model (discovered by the Inductive Visual Miner), data objects and their lifecycles for the Hospital Billing event log (the color figure is available online) 86
- Figure 40 Discovered data objects and their Attribute-Access behavior for the Road Traffic Fine Management event log (the color figure is available online) 87
- Figure 41 Process model (discovered by the Inductive Visual Miner), data objects and their lifecycles for the Road Traffic Management event log (the color figure is available online) 88
- Figure 42 Discovering business process architectures from a set of event logs 91
- Figure 43 The overall approach that discovers the business process relations (represented by dashed lines) in the form of a BPA by just considering a set of event logs as input. Each event log is beforehand extracted from the organization's database by having in mind a different business goal 93
- Figure 44 The meta-model for describing the relation between the business process architecture and the event log 94
- Figure 45 Example of a case from event log El_i being aware of another case pertaining to the event log El_j 95
- Figure 46 The overall method for discovering the trigger flow patterns between two pairs of event logs 97

Figure 47	The overall method for discovering the information flow patterns between two pairs of event logs	99
Figure 48	Complex behaviour of trigger and information flow	102
Figure 49	The discovered trigger flow (simple representation) of the BPI Challenge 2017	106
Figure 50	The discovered trigger flow (detailed representation) of the BPI Challenge 2017	106
Figure 51	The discovered information flow (simple representation) of the BPI Challenge 2017	108
Figure 52	The discovered information flow (detailed representation) of the BPI Challenge 2017	108
Figure 53	The discovered trigger flows of the BPI Challenge 2020	109
Figure 54	The overall contributions of this thesis	115

LIST OF TABLES

Table 1	An overview of the related work dealing with the data model generation	56
Table 2	The description of the RTFM event log attributes	57
Table 3	The description of the Hospital Billing event log attributes [86]	83

INTRODUCTION

Business Process Management (BPM) plays an important role in designing, implementing, controlling and improving the business processes of an organization [139]. The main artifact of BPM is the process model, which captures the main steps performed within an organization that achieve a certain business goal [124, 139]. One option to extract process-related information is to rely on the event data of the running information systems [128] and apply process mining.

In the last two decades, process mining has developed from a niche discipline to a significant research area with considerable impact on the industry [98]. Organizations can gain deep insights about their running business processes by applying different process mining techniques like discovery [76], conformance checking [107, 138], and performance analysis [130]. All these techniques require as input an event log — a list of timestamped events that mark meaningful happenings in an organization [128]. These events are created from the organization's running information systems and are usually extracted from the respective databases beforehand.

The extraction of an event log usually requires access to the whole database [18], specific knowledge about its structure [35] and domain expertise [44]. This becomes challenging in a real-world setting because holistic domain expertise is hard to find [101] and organization databases are very large and complex [34]. Once these challenges are overcome, the extracted event logs are very valuable, in that, besides capturing the sequences of events, they are a rich source of domain-specific knowledge. However, this knowledge is only implicitly captured in the data manipulated by the events. Extracting this knowledge from the event logs is usually overlooked, especially when the underlying databases are too large and complex to understand from a process perspective.

Process experts can use the event logs to discover the process model, i.e., by applying any process mining discovery algorithms like the Alpha Miner [96] or the Heuristic Miner [116]. However, the discovered process model is not sufficient to be able to understand how the process affects its contextual environment and the impact of the environment on the process. In the context of this thesis, the environment can be data or other processes that are intertwined with the process at hand. There are two formal ways to represent data and process dependencies within a single organization, respectively: data models and business process architectures.

Data models provide key information about the main entities involved in the process and the relation between these entities. The standard language for capturing data models is Unified Modeling Language UML [108]. Data models can be linked to data objects in a process model, which complements the control-flow with data-flow [94]. However, there is a research gap in understanding the underlying data model that is implicitly embedded in an event log, even more specific, when it comes to understanding and visualizing how this data affects the running process instances on the process model level.

Nevertheless, the data within one organization can not always be isolated in that different processes can access to common data. This means that having a narrow view of the data each process accesses or modifies is not sufficient to understand how data changes and, most importantly, how these business processes are interconnected. This is where Business Process Architectures (BPA) prove to be useful [46]. They provide a holistic view of all the processes in an organization's process repository and the relation between these processes. However, the state-of-the-art research on designing Business Process Architecture relies only on information found at the process model level of abstraction, thus neglecting processes execution information like event logs [49, 50]. The information store in these logs can reveal complex relations that are otherwise not discoverable only by looking at the process model. For example, a process model can trigger 1-to-n instances of another process model or, depending on certain decision within a process models, information can flow to different target process models.

1.1 STORY

To make the problems in this thesis more tangible, we present the following story, which we will repeatedly visit to showcase our solution. Suppose that John is a process expert within an organization. One of the tasks assigned to him has to answer a set of questions related to the organization's running business processes. Assuming that no business process model is captured in the corresponding organization implies the need to apply process mining discovery techniques to discover such models. The first requirement of any process mining technique is an event log. If these logs are unavailable, John has to extract them from the organization's database. The premise is that John does not have direct access to or holistic knowledge of the database but only access to the Redo Logs. Redo logs are used within an organization to store the data operations and bring the database into a consistent state in case of system failure (e.g., power failure).

Once John can derive an event log from the Redo log (see Figure 1), he can apply any process mining discovery algorithm to discover the process model and start answering his process-related questions. Deriving a process model alone is not sufficient for John to understand how the

process model affects its environment and, in addition, the impact of the process model on the environment. The environment can be data or other processes that are connected with the process at hand. This step is proven to be time-consuming [140] and the complexity increases on the size of the event log in terms of the number of cases, attributes and activities. Hence, John requires a method that can aid him in identifying the underlying data model and how the running process is affected by the data (see Figure 1).

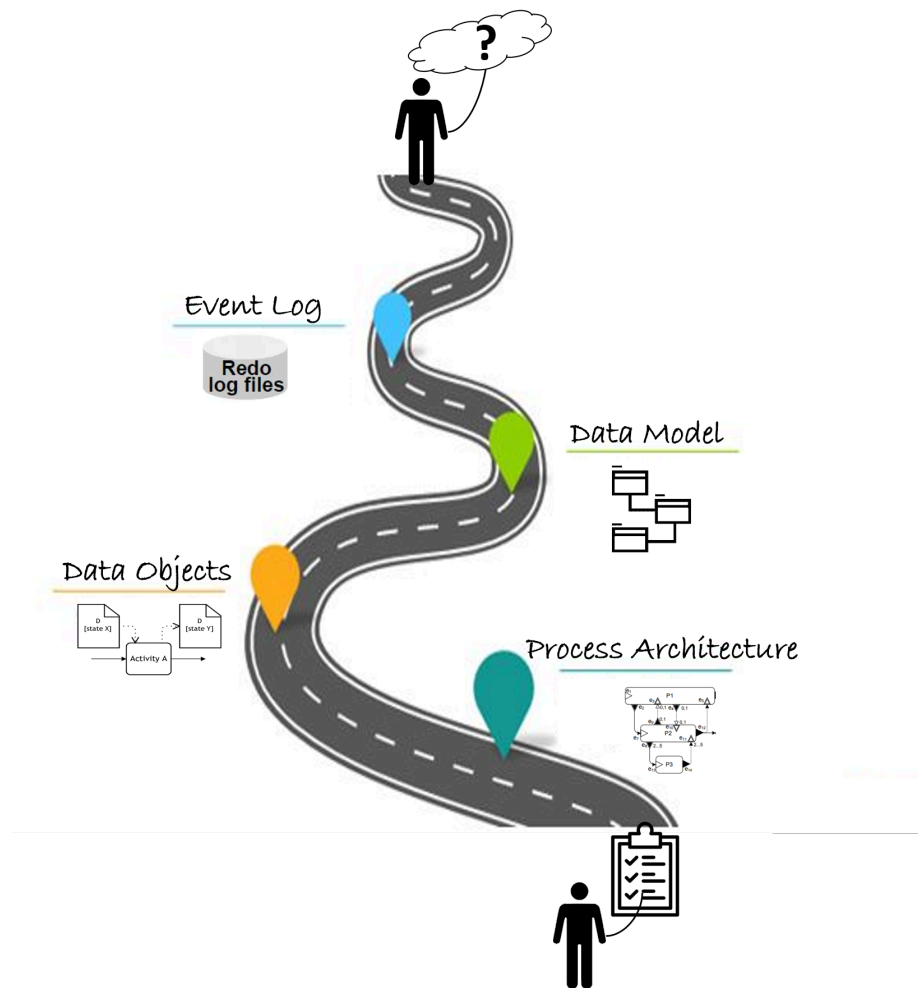


Figure 1: Methods provided to John to support him during his daily work

However, this data is not always isolated in that other processes can access it. John should not have a narrow view of the relation between a single process and its data because it limits his visibility of the overall data dependency between different business processes. It is possible for John to 1) extract several event logs, each pertaining to a different business goal, 2) discover a process model for each event log, and 3) derive the business process architecture (BPA) directly from the process models. However, the derivation of the BPA would solely be based on the activity labels' matching between the process models [56]. To re-

ally understand the dependencies between these process models, John needs to consider historical execution data, which can be found only by looking at all corresponding event logs together.

This thesis aims to support John in deriving an event log in the absence of a proper database, extract the underlying data model from an event log and, finally, understand how the data impacts the process not only in isolation but also how it manifests dependencies between co-existing business processes.

1.2 CONTRIBUTIONS

This thesis mainly focuses on the automated analysis of the event log attributes to derive context-aware information that can not be captured by solely considering the discovered process model. As a foundation, we shed light on the way how the organization's data are obtained and manipulated by the running processes. Based on this data, a set of methods are designed and provided to the process expert allowing her/him to exploit the event logs to their full capacity even when the access to the original database system and the domain knowledge is limited or unavailable. These methods help the process experts to better understand the discovered business processes by having a holistic view of the data involved with an organization and its business process models.

As it is illustrated in Figure 2, each method captures specific data aspects while involving different artifacts. These methods touch on five main artifacts: redo log, event log, data model, data object and business process architecture. Redo logs are used by several Data Base Management Systems (DBMSs), like Oracle RDBMS¹, to bring the database into a consistent state in case of system failure. In this thesis, we provide a method that considers as input this type of log and extracts an event log as a set of cases, each representing one business process execution instance. The event logs are considered as a starting point of any process mining technique and are de facto standard for capturing process information [122]. By applying any process mining technique to an event log, the process model representing the real execution of the running information system can be obtained as a set of activities executed in a specific order to achieve a certain business goal [48, 139]. Aside from the control-flow perspective, the process model can represent other perspectives captured as the data objects or data models. Data objects are defined as entities that are processed during the process execution and are annotated with states. Data models are used to group similar data into data model classes, which are connected together via relations. Within an organization, the process repository might contain hundreds or thousands of process modes. To handle the relations between these

¹ <https://www.oracle.com/database/technologies/>

processes, the business process architecture defines what information these process exchanges or how they instantiate each other.

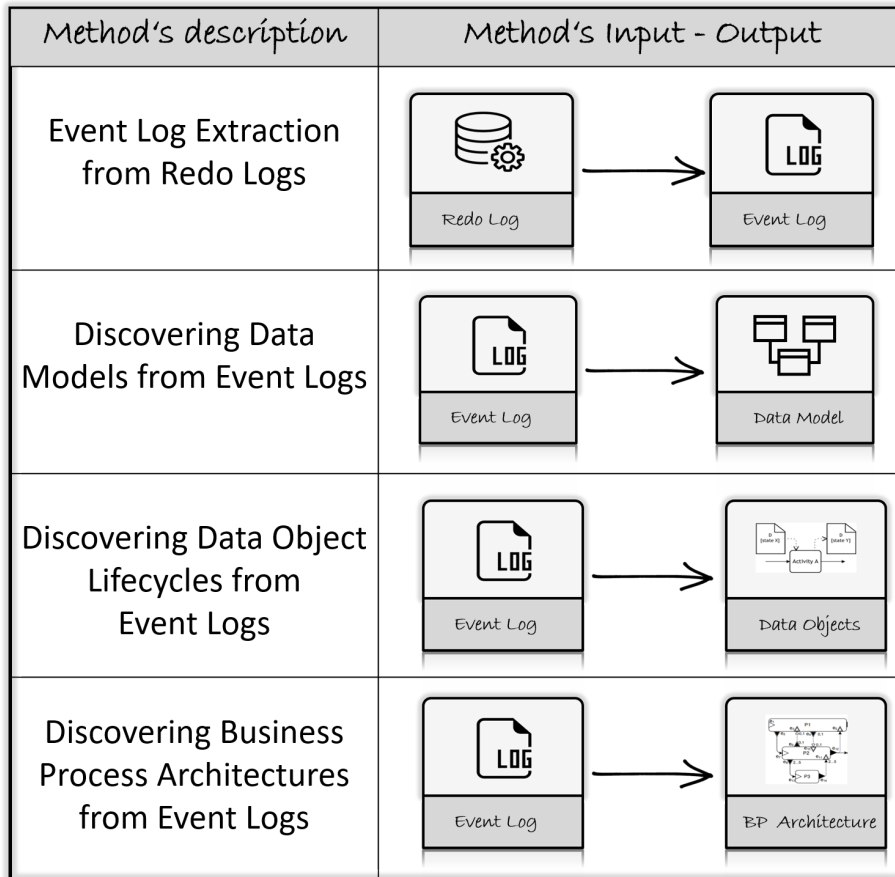


Figure 2: Overview of the contributions presented in this thesis

The insights about the processes and data within an organization and the presented methods lead to several contributions related to the event log extraction, process enhancement and process relation. We identify one contribution related to event log extraction from the insights obtained from the transactions made upon the running information system and stored as redo logs. Whereas, three other contributions are responsible for capturing the data perspective from an event log and enhancing the process model with complementary information, which is useful to better understand the process. Specifically, the four main contributions of this thesis are summarized below.

- *A method that extracts an event log from the redo logs*

In their running information systems, organizations configure the database management systems to store their transactions in so-called redo logs, which brings the database into a consistent state in case of an instance failure. We argue that these types of logs contain useful information to derive the process instances stored in the form of event logs, which are considered the main artifact in

the process mining area. As part of Chapter 3, we provide insights into why sometimes it is necessary to use this type of log for event log extraction and not data structured in the organization's database, which is a commonly known source of information for extracting an event log. The extracted event log can be tailored to the process mining experts for applying different process mining techniques, e.g., process discovery and conformance checking.

- *A method that discovers a data model from the event log*
The organization uses data models as a common understanding language between different database designers or data modelers. They are defined as a fundamental concept within an organization and contain information about the data structure in the form of classes, which are associated together via relations. As part of Chapter 4, we are aiming to discover such data models from the historical data of the given organization stored as event logs. The event logs do not contain explicit information about the context and the data they are extracted from. Despite the fact that event logs capture behavior information captured by the event log cases, we argue that they are also a rich source of domain-specific information, albeit not explicit.
- *A method that discovers the data objects lifecycles from an event log*
During the business process execution, the activities read and write to different data objects. Therefore, these activities change the data object states during each process instance. Enhancing the process model with the data objects means providing additional information about the data-flow perspective, which can not be captured by the process model itself. (i.e., the process model provides information about the control-flow). Therefore, they play an important role in process model understandability. As part of Chapter 5, we are providing a method that can be used by any process expert to discover the data-flow perspective based on the information stored in the event log. This is realized by tracking how the event log activities write or modify the event log attribute values, propagating how the business process activities manipulate the data objects during the process execution. Connecting these two perspectives together and providing a holistic view increases the information which can be gained by solely analyzing the event log and the discovered process model.
- *A method that discovers business process architecture from event logs*
Business process repositories within an organization capture hundreds or even thousands of business processes. Having a holistic view of the interrelationship between these processes helps the organizations to organize, manage and better understand their processes. Typically, these relations are designed by process experts and usually involve manual tasks and extensive domain

knowledge [74] making the whole designing process not scalable. Chapter 6 shows the potential of automating this step using the event logs extracted from the organization's database to derive such relations. This is achieved by analyzing the causality relationships between the events pertaining to two different event logs but having access to common data.

Each method is totally independent from each other and can be used as standalone anytime the requirements are fulfilled.

1.3 STRUCTURE OF THE THESIS

In this thesis, we provide a set of methods that any process mining expert can use (e.g., John based on the story presented in Section 1.1) to discover not only the business process model from an event log but also context-aware information pertained to the event log attributes and activities. The structure of the thesis is illustrated in Figure 3 and explained in more details below.

- **Chapter 2 (Preliminaries).**

Recalls the basic notions that are necessary to set the stage for presenting the main contributions of this thesis. The main concepts are elaborated through the lens of process mining and business process management as fundamental areas of this thesis.

- **Chapter 3 (Event Log Extraction from Redo Logs).**

Presents a two-step method for obtaining a reliable event log by considering as input only the transactions made upon the database of the running information system and stored in the form of Redo Logs. In the first step, the data model, which serves as a substitute for the database schema, is discovered directly from a redo log. In the second step, the redo entries are correlated with the event log cases. By using the proposed method, it is possible to extract an event log from a different source of information besides the one generally used in the process mining, i.e., database

- **Chapter 4 (Discovering Data Models from Event Logs).**

Describes a two steps method that takes as input an event log and discovers the data model. In the first step, an intermediate representation called the Attribute-Access relationship diagram is obtained based on the access relation defined between the event log attributes and activities. In the second step, a set of rules are applied to the obtained diagram and the data model is discovered.

The discovered data model can be used as complementary information to the process model by increasing its understandability.

- **Chapter 5** (Discovering Data Object Lifecycles from Event Logs).
Describes a method that takes as input an event log and discovers the data objects and their lifecycles by tracking how the event log attributes are manipulated within a case. The discovered information enhances the process model (i.e., which is upfront discovered from the same event log by applying any process mining discovery algorithm) with a data-flow perspective revealing how the process activities manipulate the data during the execution.
- **Chapter 6** (Discovering Business Process Architectures from Event Logs).
Describes a method to discover the relations between several business processes, each of them discovered from an event log by having in mind a different business goal. This is realized by discovering considering two types of relationships defined in the scope of the BPA area, called trigger flow and information flow.
- **Chapter 7** (Conclusions).
Finally, the thesis is concluded in the last chapter. The contributions are summarized, and further research directions to improve or extend the presented methods are provided in the last chapter as well.

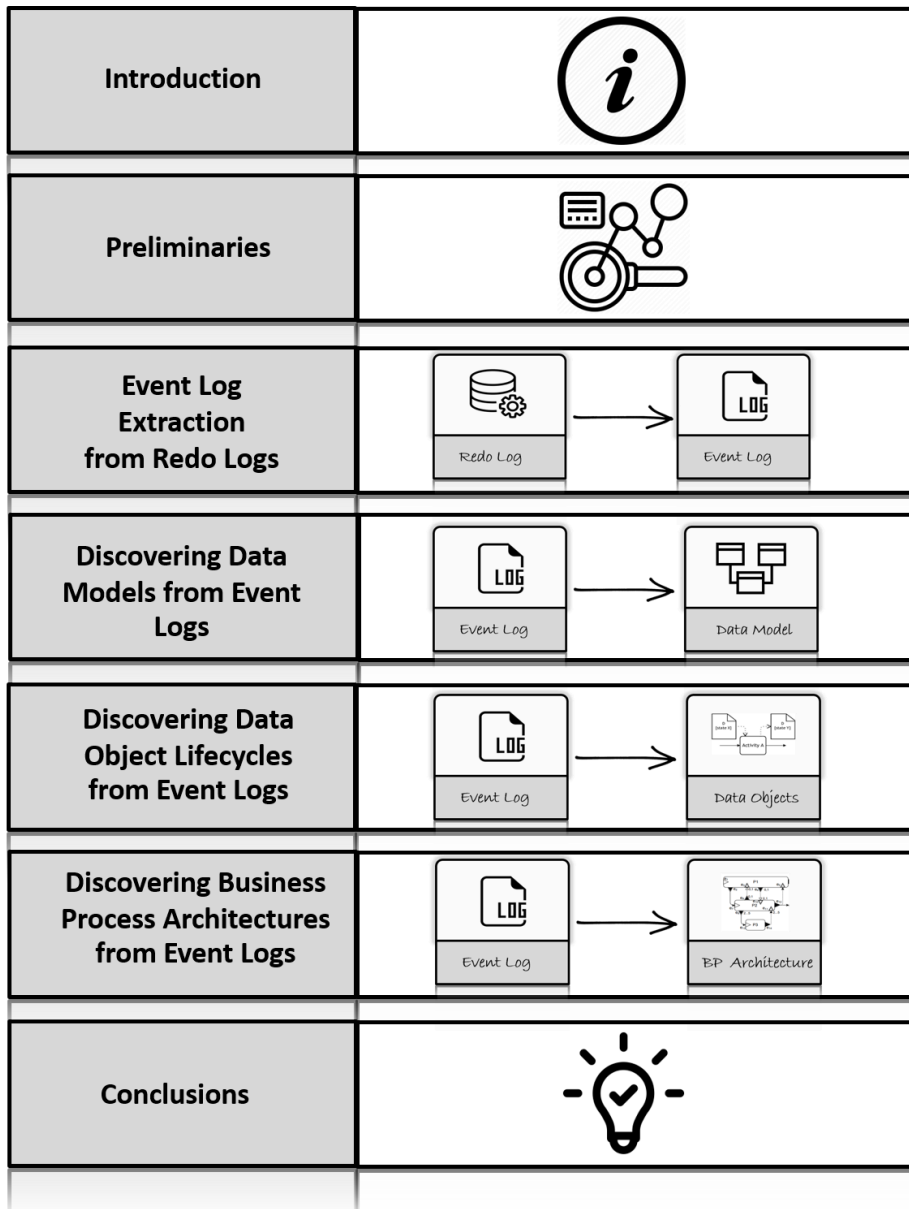


Figure 3: The structure of the thesis

This chapter recalls the basic notions that are necessary to set the stage for introducing the main contributions of this thesis in the consecutive chapters. It elaborates the main concepts through the lens of business process management and process mining, as the fundamental fields of this thesis.

2.1 BUSINESS PROCESS MANAGEMENT AND MODELING

Business Process Management (BPM) has been the focus of research at the intersection of computer science, information systems engineering and information system management since the early 1990 [36, 60]. BPM is defined as a discipline for studying business processes, which are determined as a set of activities performed in a particular order while considering a specific business goal [139]. A business process can be managed based on several aspects of a so-called BPM lifecycle. Many BPM model lifecycles exist in the literature [48, 61, 91, 139, 142] but in this thesis, we are mainly focused on the model defined by Weske in [139]. The lifecycle model covers four fundamental phases: design and analysis, configuration, enactment and evaluation of business processes (see Figure 4). Although there is a logical dependency between these phases, an independent visit is also possible. A detailed explanation of each phase is provided below.

- **Design and Analysis**

As the name implies, the processes are identified within an organization and designed as a process models during this phase. Therefore, business process modeling is considered a core method during this phase. Process stakeholders are equipped with the designed process models and use them to communicate among each other. Analyzing the designed process models helps the stakeholders to shed light on deadlocks' existence, performance analyses (e.g., related to the execution time of the activity) or unreachable activities. Different types of analysis, like structure or behavior analysis, require different efforts to retrieve insights about the process improvements (e.g., considering cost, efficiency, effectiveness). During the re-design phase of the process model, the learned information is taken into account.

- **Configuration**

Once the process model is designed, it needs to be configured before being enacted. Depending on the Business Process Man-



Figure 4: The BPM lifecycle [139]

agement Systems (BPMS), it might be necessary to enrich the process model with technical details that allow its deployment and execution. Setting up the wrong configuration might lead to bad deployment and execution even though a well-designed process model is considered as an input.

- **Enactment**

Once the systems support the designed processes, the enactment phase starts. This phase executes the business processes to achieve the designed business goal. Depending on the BPMS features, different stakeholders might be involved during this phase to monitor the execution of the business process instances as defined in the business process model. The execution data are gathered in so-called log files, which typically track the start and end time of the business process activities.

- **Evaluation**

The business process evaluation mainly considers the information stored in the execution logs, which is output from the previous phase. Business process experts use this information to evaluate the process quality based on predefined metrics. Different process mining techniques (see Figure 4) can be applied to the execution logs and different outcomes related to the process quality can be derived, e.g., process performance, bottle necks and decision paths.

Several behavior aspects of an organization can be captured in a so-called *process model*. These aspects are described and graphically illustrated based on a specific modeling language. Several process modeling languages are defined in the literature and each of them has its

own semantics and syntax. For example, Petri net, as one of the oldest and most well-structured modeling languages, is determined as a state-based language rather than an event-based [121]. The standard modeling language, which is also widely used in both academia and industry (e.g., in e-commerce) is Business Process Model and Notation (BPMN) [27, 139]. In BPMN, processes are treated as a core element and all operations happening within an organization are viewed through the lens of processes. A BPMN process model contains events, activities and gateways. In addition, control-flows are used to define the causal dependencies between the elements. Aside from the control-flow, the process model can represent other aspects such as data-flow and data objects (a detailed explanation of them is provided in the next section). As an imperative modeling language, BPMN is our language of choice in this thesis to model and illustrate the process models. A tiny subset of BPMN notions that are covered in this thesis is illustrated in Figure 5.











	Activity
	Multiple Instance
	Sequence Flow
	Data Flow
	Parallel Gateway
	Exclusive Gateway
	Data Object
	Start Event
	Intermediate Event
	End Event

Figure 5: A subset of the BPM notions covered in this thesis

Formally, the definition of the process model is presented as follows:

Definition 2.1.

(Process Model) A process model is defined as the tuple

$PM = (A, G, E, F)$ where

- A is a finite set of activities
- $G = G_x \cup G_a$ is a finite set of exclusive and parallel gateways respectively, where G_x and G_a are disjoint

- $E = E_s \cup E_i \cup E_e$ is a finite set of start events, intermediate events and end events. E_s , E_i and E_e are pairwise disjoint.
- $F \subseteq (A \cup G \cup E) \times (A \cup G \cup E)$ is a set of sequence flows, which are ordered pairs of activities, gateways and events.

The process models within an organization are stored in a so-called process model repository, which often captures hundreds or thousands of models. These processes require changes in a perpetual repetition in order to accommodate the ever-changing business requirements [139]. The more process models are stored in such repositories, the more challenging it is to manage them over time. The BPM area addressing such a challenge is called *Business Process Architecture (BPA)* [46]. Typically, BPAs are designed using the relationships between process models in a given repository, i.e., BPA design is purely based on process models rather than execution logs of processes. BPAs provide an abstract view of all business process models happening in one organization. Authors in [50] emphasize that different types of patterns are identified to express the relations between two different business processes. However, the following patterns are widely used throughout the literature:

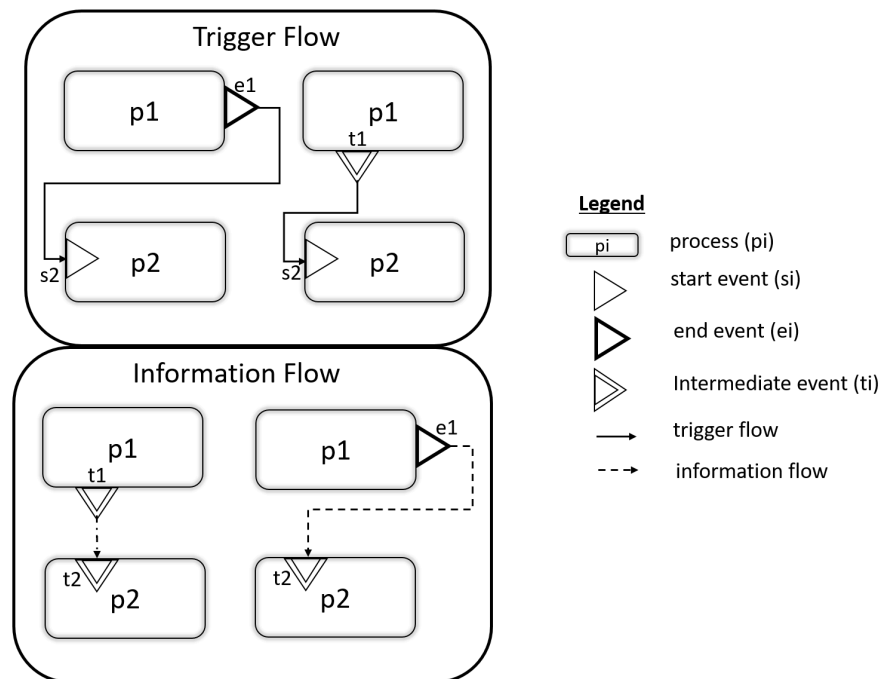


Figure 6: BPA representing trigger and information flow patterns [50]

- **decomposition** - a business process model is decomposed into other business process models each representing a sub process
- **specialization** - a business process model is a specialized version of another business process model

- **trigger flow** - an event of a business process models triggers the instantiation of another business process model
- **information flow** - an event of a processes model passes information to another event of a different process model.

This thesis focuses only on the trigger and information flow between two or more process models. To illustrate these patterns we use the graphical representation presented in [50] and depicted in Figure 6. The upper part of the Figure illustrates the trigger flow pattern, in which the throwing end event e_1 of process p_1 instantiates the process p_2 through the catching start event s_2 . The process can also be instantiated from an intermediate event t_1 . In contrast, the lower part of the Figure 6 illustrates the information flow patterns, which represents the case where process p_2 receives information from an intermediate t_1 or end event e_1 of process p_1 . The trigger flow relations are designed as a normal line connecting two different processes, while the information flow relations are designed with dashed lines.

Since the BPA is defined between at least two process models, it is necessary first to recognize the throwing and catching event in the corresponding models. The *throwing* event model is considered as *active* because it sends a signal to the other process. It can be an *intermediate* or *end* event model (see Figure 7). In contrast, the *catching* event model is considered as *passive* because it waits for the signal to arrive. It can be a *start* or *intermediate* event model (see Figure 7).

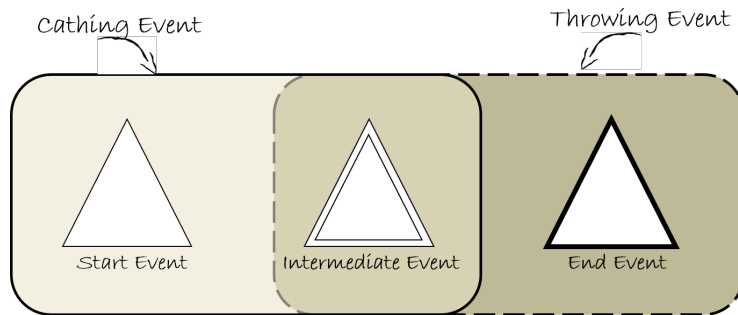


Figure 7: The visual notations of catching (start/intermediate event) and throwing events (intermediate/end event) in a BPA

Below is the formal definition of a BPA as used in this thesis:

Definition 2.2.

(Business Process Architecture) A business process architecture is a tuple $BPA = (P, E, \text{element}, F)$ in which:

- P is a set of processes
- E is a set of events defined as $E = E_e \cup E_s \cup E_i$, where E_e is the set of end events, E_s is the set of start events and E_i is the set of intermediate events and they are pair wise disjoint

- element: $P \rightarrow 2^E$ assigns events to processes
- $F \subseteq (E_i \cup E_e) \times (E_s \cup E_i)$ is a set of flows between two events. In addition, we distinguish between two type of distinct flows: $F = F_i \cup F_t$. $F_t \subseteq (E_i \cup E_e) \times E_s$ is the set of trigger flows (they target only start events); $F_i = F \setminus F_t \subseteq (E_i \cup E_e) \times E_i$ is the set of information flows (they target only intermediate events).

◀

2.2 DATA IN BUSINESS PROCESSES

This section shortly summarize the notions related to data in the business processes.

2.2.1 Database's Redo Logs

Information Systems stores their execution data in a so-called Database. Several Data Base Management Systems (DBMSs), like Oracle DBMSs, provide redo logs to store conducted data operations and use them to bring the database into a consistent state in case of system failure (e.g., power failure). The redo log stores all database changes as they occur and is used to establish the database history for a certain period of time. Each redo log is overflowing with redo entries and each entry in a redo log corresponds to a transaction executed by the database system. For example, if we change a certain value of an attribute in the database (e.g., patient marital status), a redo entry is recorded in the redo log describing precisely the changes made in this table. These systems, like Oracle RDBMS enable redo logs to store the historical view of what has happened in the system.

```

1 insert into "SYSTEM"."PATIENTS" ("ID","GENDER") values ('86','
  M') AAAT; 08-JAN-2021 12:46:15
update "SYSTEM"."DIAGNOSES" set "ID" = '584' where "
  ADMISSION_ID" = '101' and ICD_CODE='P599' and ROWID = '
  BADR'; 08-JAN-2021 12:45:33
delete from "SYSTEM"."ADMISSIONS" where "ID" = '32' and "
  PATIENT_ID" = '34' and ROWID = 'SABD'; 08-JAN-2021
  12:46:27

```

Listing 1: Redo log fragment: each statement corresponds to an transaction made on the database and it is called a redo entry

Each transaction in the redo log is represented as an SQL statement (called redo entry) and consist of:

- the operation made upon a certain database table i.e., insert, delete, and update
- the affected attributes and the corresponding values
- the row id on which the statement must be applied

- the timestamp of the statement occurrence

Some examples of the redo entries corresponding to patients, admissions and diagnoses are illustrated in Listing 1.

2.2.2 Data Model

In conceptual modeling [57, 108] and especially in process modeling [62], the data are considered to belong to a set of objects, which might be linked together through associations. Data models as a language to design conceptual models are defined as fundamental concepts within an organization and widely used to represent their data and used as a common understanding language between the database designers or data modelers. Similar data objects are grouped in so-called data model classes, containing a name and typed attributes. The connection between data model classes is handled via relations characterized by the lower and upper bound values and can be defined as one-to-one, one-to-many, many-to-many, zero-to-one, or zero-to-many relations. Different types of relations can be defined between two data model classes. For example, if there is an instance of a class in order to complete its task requires the information about the other, then an *association* (e.g., graphically represented as a simple line) is defined between these two classes.

Generally, the whole data structure of an organization can be described via a data model. Prominent modeling techniques for data models include Entity-Relationship (ER) models [26] and Unified Modeling Language (UML) class diagram [108]. In this thesis, the standard Unified Modeling Language (UML) is our language of choice. When a data model is instantiated, any instance must comply with the model. In addition, the instances might change over time (i.e., new objects are created, existing ones are updated) and these changes do not require changes in the data model. Below is given a succinct definition of the data model:

Definition 2.3.

(Data Model) A data model D is a tuple

$D = (C, Att, Aso, member, attrmulti, asomulti)$ where:

- C is a non-empty finite set of classes
- Att is a set of attributes
- $Aso \subseteq C \times C$ is the set of associations between classes
- $member : C \rightarrow 2^{Att}$ assigns attributes to classes
- $attrmulti : Att \rightarrow \mathbb{N}_0 \times \{*\}$ defines the multiplicity of any attribute in a class where $*$ stands for arbitrarily many and we assume that $* \in \mathbb{N}$

- $asomulti : Aso \rightarrow \mathbb{N}_0 \times \{*\}$ defines the multiplicity of any association in the data model

◀

An example of the data model illustrated as a UML class diagram is depicted in Figure 8. It consists of two classes: *Class A* and *Class B* and each of them has two attributes (*attribute1* and *attribute2* pertain to *Class A*, while *attribute3* and *attribute4* pertain to *Class B*). There is only one relation defined between these classes with a multiplicity from 1 (*Class A*) to many (*Class B*).

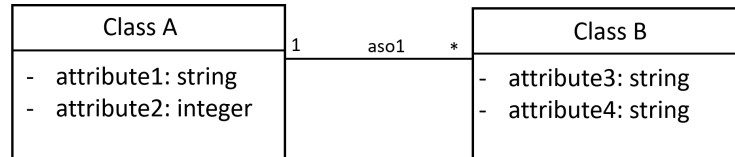


Figure 8: An example of the data model illustrated as a UML class diagram

2.2.3 Data Object

Aside from the control-flow, process model can represent another aspect, that of the data-flow, i.e., the data created and modified by process activities as well as exchanged between them. In BPMN, the data created and used by process activities are represented through data objects, which are defined as entities that are processed during the business process execution. They are connected with activities via directed data associations, represented as directed edges. Based on the data-flow edge direction, read/write access are defined. An edge from the data object to the process model activity represents a read access to the data object. Alternatively, an edge from an activity to the data object represents a write access. Write access defines the creation of a new data object or the modification of an existing one [94].

Data objects in the process can be annotated with their state and each data object can be only in one state at one point in time. During the process execution, the data object state can change. The execution of a certain process activity could require the data object to be available in a particular state. Once the activity is completed and the data object is consumed, it might lead to a data object state update. The combination of different data object states and the possible transitions between them can be captured in a data object lifecycle (OLC), which represent the behavior aspect of the data objects.

A representation of the the data object in BPMN is illustrated in Figure 9. Data Object *X* has state *low* and is written by the first activity (*A*) happening in the process. If activity *C* occurs then it reads the data object written by activity *A*. In contrast, if *B* occurs then a write access to the database (*DB*) takes place. A repository/database access is used in BPMN to persists the manipulation of the data by the process.

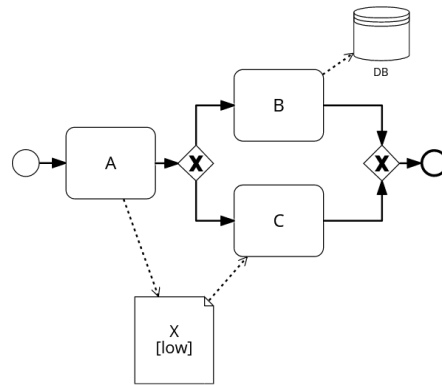


Figure 9: An example of a process model (represented as BPMN) and the data object/repository associated with its activities

2.3 PROCESS MINING

In this section, we provide the basic notions related to the process mining field. First, we describe how this research area is positioned in the real world and afterwards explain the main techniques covered within this area. We conclude the section with a short event log excerpt, which is the main subject of these techniques.

2.3.1 Process Mining Position

The process mining area establishes links between the business process management and the data mining area [124]. The former mainly focuses on the process-based data analysis techniques, where the process is the main artifact. In contrast, the latter considers data-oriented analysis techniques, where the statistical analyzes are the main operations performed upon the data. Process mining links these two areas to answer the questions related not only to the behavior and execution order of a certain business process activities but also provides outcomes related to the performance and conformance analysis of the processes.

Process models serve as a blueprint for process execution, which defines the order under which the process activities have to be executed within an organization. All the process instances conducted during the business process execution are happening in the Process Execution Environment (see Figure 10). However, there are cases within an organization in which the process instances behavior is not the same as it is defined in the model. To capture such behavior, it is necessary to discover the process model from real-life data generated during the business process execution. In addition, the data generated during the business process execution might have a different format, for example, text files, database files, and spreadsheet documents. To discover the process model from this type of data, first, an event log has to be extracted, which is defined as the first requirement for any process mining

technique. The event log is a set of execution cases, where each of them represents an instance of the process execution. Once the event log is extracted, different process mining techniques can be applied: process discovery, conformance checking, and process enhancement [122]. A detailed explanation of each technique is provided below.

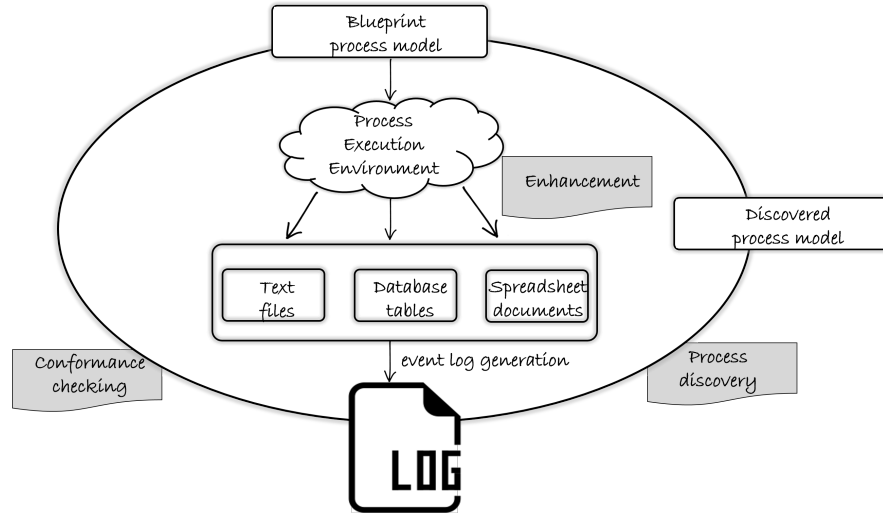


Figure 10: Process Mining position [72]

- **Process discovery**

This technique aims to discover a process model from an event log after applying a process discovery algorithm. Different discovery perspectives are identified in the literature, like process, organization or case perspective [133]. The main goal of the discovery algorithm that focuses on the process perspective is to discover the control-flow, which covers the behavior found in the event log. The majority of the discovery algorithm found in the literature supports this perspective, like Alpha Miner [96], Heuristic Miner [116], Inductive Visual Miner [76]. The organizational perspective covers the classification of people in terms of roles, departments or organization units. The outcomes of the organizational discovery algorithms are plotted in the social network. In contrast, the case perspective is mainly focused on the process instance characteristics. The data elements, roles and persons contributing to the business process instance are some characteristics that can be analyzed within this perspective.

- **Conformance checking**

Conformance checking is another process mining technique that deals with the comparison of an existing process model with an event log of the same process [3, 129]. The main question that needs to be answered from the comparison is whether the real execution cases conform to the model and whether the model can

represent a portion of the cases. There are different approaches towards conformance checking discussed in the literature, like in [66, 107, 110].

- **Process enhancement**

The third process mining technique is called process enhancement, which deals with the enhancement of the process model based on the data captured in the event log. The process model can be repaired to represent the reality better or extended by adding an additional perspective and by cross-correlating the process model with the event log [130].

2.3.2 Event Log

Since all process mining techniques require as input an event log, there is a need to provide the fundamental concepts related to the event log structure [124]. Each event log consists of a set of cases, each representing an execution of a business process instance. Each case consists of a sequence of events, where each event captures the execution of specific business activity within a case. The de facto standard format for storing, exchanging and analyzing an event log is the eXtensible Event Stream (XES)¹ [1]. This format is supported by many process mining tools like Disco², ProM³ and implemented by OpenXES, which is an open-source java library for operating in the XES logs. The event log is structured as a table where rows represent the events and columns represent the event log attributes. The association between events and attributes in the event log is defined via the attribute values. Each event log contains three mandatory attributes [127]:

- the **case identifier**, which correlates several events into a single case i.e., a process instance
- the **activity name**, which identifies well defined execution steps of the process instance
- the **timestamp**, which indicates when the event occurred

Depending on the business goal, events can be associated with additional attributes, such as the resource, describing the person or the device responsible for executing a specific activity, or the department, defining where the activity has been executed.

Within the same case, attributes can be associated with a single event or refer to a case as a whole. The former ones are called *event attributes* and their value can vary based on the process step (event) being executed. The later ones are called *case attributes* and are invariant, i.e., they do not change as the events of the case occur.

¹ www.xesstandard.org

² <https://fluxicon.com/disco/>

³ <https://www.promtools.org/doku.php>

Let us denote a set of mandatory attributes with $M_{att} = \{Case, Act, Time\}$. The following definitions are based on [124].

Definition 2.4.

(Event) An event e over a set of attributes Att is defined as $e = (\#_{att_1}, \#_{att_2}, \#_{att_3}, \dots, \#_{att_n})$ where $\#_{att_i}$ is the value of attribute $att_i \in Att$ for $i = 1 \dots n$. If an event e is not associated with an attribute Att , then $\#_{att}(e) = \perp$ (null value). Otherwise, an event e accesses an attribute Att , if $\#_{att}(e) \neq \perp$ ◀

Definition 2.5.

(Case) A case c is defined as a finite sequence of events σ such that each event appears only once, i.e., $1 \leq i < j \leq |\sigma| : \sigma(i) \neq \sigma(j)$ where $\sigma(i)$ is the i -th element in the sequence σ ◀

Definition 2.6.

(Event Log) An event log El is defined as $El = \{e_1, e_2, e_3, \dots, e_m\}$ where $m \in \mathbb{N}$ is the entire number of events. In this thesis, for each event $e \in El$ it is assumed that all mandatory attributes are present, thus $\forall e \in El: \#_{case}(e) \neq \perp \wedge \#_{act}(e) \neq \perp \wedge \#_{time}(e) \neq \perp$ ◀

An event log example is illustrated in Figure 11 and formally can be defined as: $El = \{e_1, e_2, e_3, \dots, e_6\}$ where:

$$e_1 = \{1, A, 1\},$$

$$e_2 = \{1, B, 2\},$$

$$e_3 = \{1, D, 3\},$$

$$e_4 = \{1, E, 4\},$$

$$e_5 = \{1, F, 5\},$$

$$e_6 = \{2, A, 6\} \dots$$

are defined over the attribute set

$$Att = \{Case\ identifier, Activity\ name, Timestamp\}$$

As you can see, the event log contains a lot of information on top of business processes in the form of attributes where the data model is implicitly and partially contained. In the absence of direct access to the databases, the event log can be a rich source of contextual data surrounding business processes.

In this thesis we are assuming that the event log contains the following information:

- In the process mining area, an event log beside the mandatory attributes (i.e., in this thesis, referred to as meta-attribute) can also contain a set of optional attributes. In order to apply the methods introduced in this thesis to an event log, it is necessary first that the event log contains sufficient non-static optional attributes.
- The association between the event log activities and attributes is handled via the attribute values. We assume that the attribute

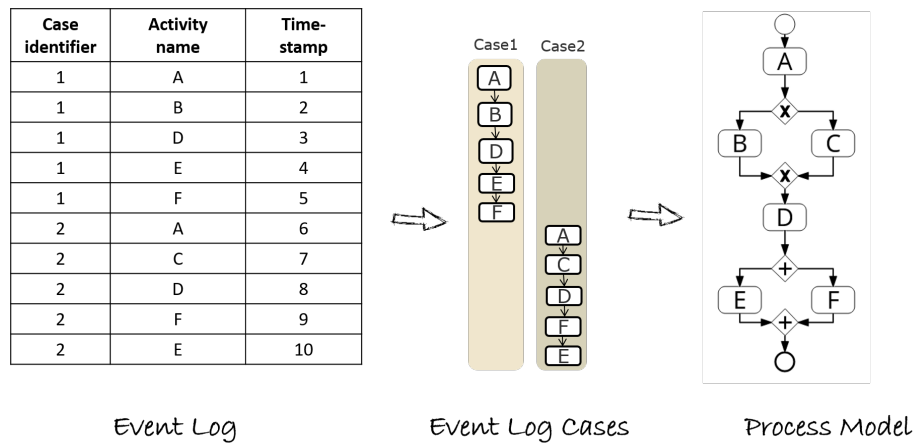


Figure 11: An event log example, its cases and the discovered process model

write access is explicitly represented in the event log, in that for each event it is clear which attributes are written by which activities. For simplicity purpose, we will refer to “write access” as simply “access” for the rest of this thesis. For example, in the Road Traffic Fine Management event log [87] presented in 4.6, the access is represented by concrete values in the accessed attributes and empty values for the rest of attributes that are not accessed.

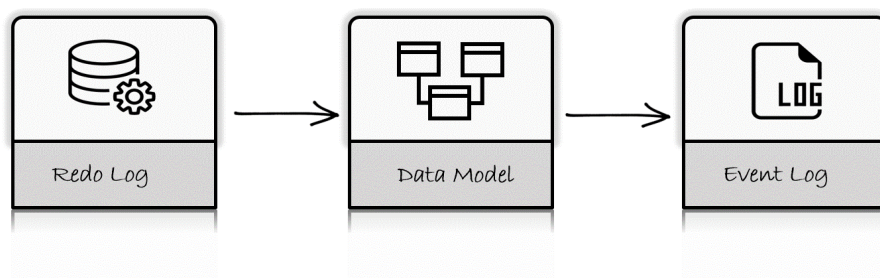


Figure 12: Event log extraction from redo log through the data model

This chapter provides a method to extract event logs from redo logs — components that are used to bring the database into a consistent state in case of a system failure. As depicted in Figure 12, this chapter shows how a data model, which serves as a substitute for the database schema, is discovered directly from a redo log. Such a model is necessary to correlate the redo log entries with the event log cases.

This chapter is mainly based on the previous work presented in [7].

3.1 MOTIVATION

Applying process mining in an organization requires as input an event log. Usually such logs are extracted from the organization's databases [44, 119]. Therefore, access to the database supporting the corresponding information system is necessary to complete the event log extraction process [18, 101]. In a real-life scenario, direct access to the database system is not always possible due to restrictions related to the security and privacy of the data stored under these systems. On top of that, the organization's data might be stored in a distribution setting, which might mean that different parts/modules of the application store their data in different databases. Extracting an event log from these systems implies the need to configure privileges to all organization's databases. However, granting access to the sensitive data stored in these systems (e.g., user's credentials) is not always possible nor preferable by the organization.

The database structures come with different complexity and diversity (e.g., star schema, snowflake schema) making extraction step considerably time-consuming [34, 44] and complicated [59, 65]. Navigating through the right data scattered among different tables and correlating them into the event log cases requires knowledge about the structure under which these data are organized into the database tables [64]. For several reasons, the required knowledge to perform such a task might not always be available. Therefore, several assumptions must be made or some explorative methods must be applied, mostly covering manual-driven tasks. In a real-life scenario, where the databases contain hundreds or thousands of tables, the event log extraction task becomes intractable. Applying explorative methods is not always feasible within certain time constraints and, in particular, some important information might be overlooked during the manual extraction process.

Besides all of the above, the data stored in such systems is not always process-aware [34, 41] in that the data is not always recorded at the application level even though they are available in the data storage. Extracting an event log from such data implies a need to design an intermediate layer, which transforms the data into a specific structure [123]. This comes with the limitation that during the transformation process, some data which might be important for the event log are excluded before the extraction process takes place.

Considering all the aforementioned limitations, the authors emphasize that in many data analysis projects, 80% of the time is spent on data extraction and preparation, taking around 50% of the project's total cost [11, 42].

To overcome these limitations, other sources of information, such as Redo Logs, can be considered for the event log extraction. Several Data Base Management Systems (DBMSs), like Oracle RDBMS¹, use redo

¹ <https://www.oracle.com/database/technologies/>

logs to store the conducted data operations and ensure consistency and fault tolerance in case of system failure (e.g., power failure). Redo logs make it possible to bring the database into a consistent state and restore it to a previous point in time by rolling back the latest operations performed on it. They are an attractive choice for event log extraction since they do not only hold the values stored in the database but also preserve the temporal ordering of the modifications applied to the data [40, 41, 123].

The approaches proposed in the literature that leverage the extraction of an event log from the redo logs (like the work in [123] and [40]) are applicable in both process and non-process aware information systems and do not require direct access to the database. However, the state-of-the-art related work holds the knowledge about the whole database schema as a necessary condition for the event log extraction process, which does not solve all of the abovementioned limitations.

To overcome such limitations, in this chapter, we show that the event log extraction process from a redo log can be performed without requiring direct access to the whole database. Concretely, we propose a method that considers as input only a redo log for extracting the event log without the presence of a schema upfront. We argue that the redo log per se can be sufficient, in terms of information, for deriving the database structure required to extract the event log. Even better, the extraction process can be carried out in a semi-automatic way, requiring domain knowledge, which is ultimately needed to correlate the redo log entries to specific cases in the event log. We prove the feasibility of the proposed method by testing it on redo logs from two different domains, healthcare and traveling.

3.2 EXTRACTION OF EVENT LOGS FROM REDO LOGS

This section describes the method of obtaining reliable event logs by considering as input only the entries performed on the database stored in the form of redo logs.

3.2.1 *Overview of the Approach and Assumptions*

To extract an event log from the redo log we propose a two-step semi-automatic approach. In the first step the database schema is automatically discovered from the redo logs. Afterwards, a domain expert is needed to look at the discovered schema and pick the case notion based on the questions that he/she aims at answering from the process model (discovered from the event log). In the second step, the database schema and the redo log are used as input to correlate the redo log entries with the event log cases. An overview of the approach is provided in Figure 13 while a detailed explanation of each step is provided in the next section.

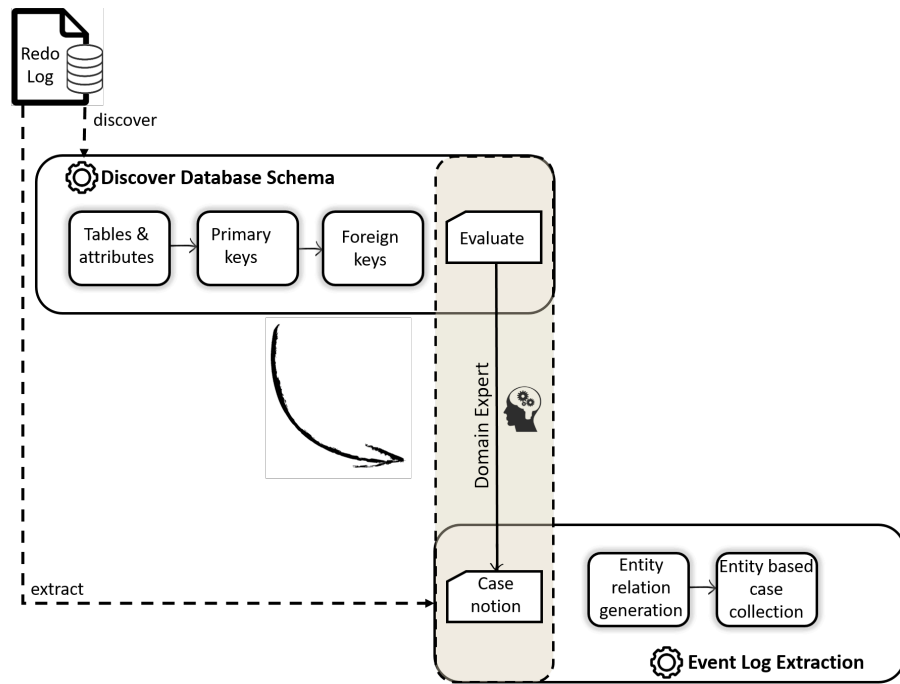


Figure 13: Overview of the event log extraction method

Before explaining the approach in more details, let us first state the assumptions.

- First, to record the redo logs, an explicit configuration² is necessary to be present in the running database management system.
- Second, to be able to read the content of such logs special database privileges are necessary to be activated.
- Third, this method requires sufficient information to be included in the redo logs. The larger the redo log size and the more nuanced (high number of unique tables and attributes), the closer to the reality the extracted event log is.

In this method, we are assuming that the configuration has taken place and the redo logs are available and accessible. In addition, we are assuming that their content is sufficiently rich to represent the running processes in the corresponding organization.

3.2.2 Discovering the Data Model from Redo Logs

We argue that the redo log entries of the running information systems contain sufficient implicit domain information for deducing the database schema, here represented as a data model. In order to discover the data model, we look in the redo log (see Figure 13) for the presence of: (1) the database table and their attributes; (2) the primary key of

² information regarding the Redo Logs configuration: https://docs.oracle.com/cd/B19306_01/server.102/b14231/onlineredo.htm

each table; and (3), the foreign keys (needed to establish the relationship between two or more tables). A detailed explanation of each step is provided below.

- **Database tables and attribute detection**

To identify the database tables, we look at each redo log entry and filter out the table names. For each operation made upon a certain table in the redo log entry, a new database schema table is constructed. If the table name already exists in the database schema then we move forward to the next redo log entry. This step requires only one iteration over all redo log entries because each entry represents only an operation made upon a single table. For example, considering the redo log entries illustrated in Figure 14 three tables can be extracted: PATIENTS, DIAGNOSES and ADMISSIONS, each of them pertaining to a different redo log entry.

Once the database tables are constructed, we have to discover the attributes pertaining to each table. Following the same idea, we iterate through each redo log entry and extract for each table the mentioned attribute names. If a new attribute is detected, it is added to the corresponding table. Considering the example depicted in Figure 14, we can see that the ID and GENDER attributes belong to the PATIENTS tables (i.e., represented with A in Figure 14), while the ID and PATIENT_ID attributes belong to the ADMISSIONS table, if we consider the third redo entry depicted in Figure 14.

After the database tables and the attributes of each table are discovered, we have to identify the relation between these tables in the next step. In each database schema the relation between tables are guided by the use of primary and foreign keys. Since the redo log entries does not contain explicit information about these types of keys there is a need to identify them just by taking in consideration from their attribute values.

- **Primary key detection**

Primary keys are defined as an important constraint indicating the attributes relations that hold on a database [2]. For discovering the primary and foreign keys we focus on the attribute values (represented with Av in Figure 14) that are involved in each redo log entry. By definition a primary key attribute contains unique values [69]. Therefore, we check whether there are attributes whose values are unique. Likewise, if duplicate values appear for the

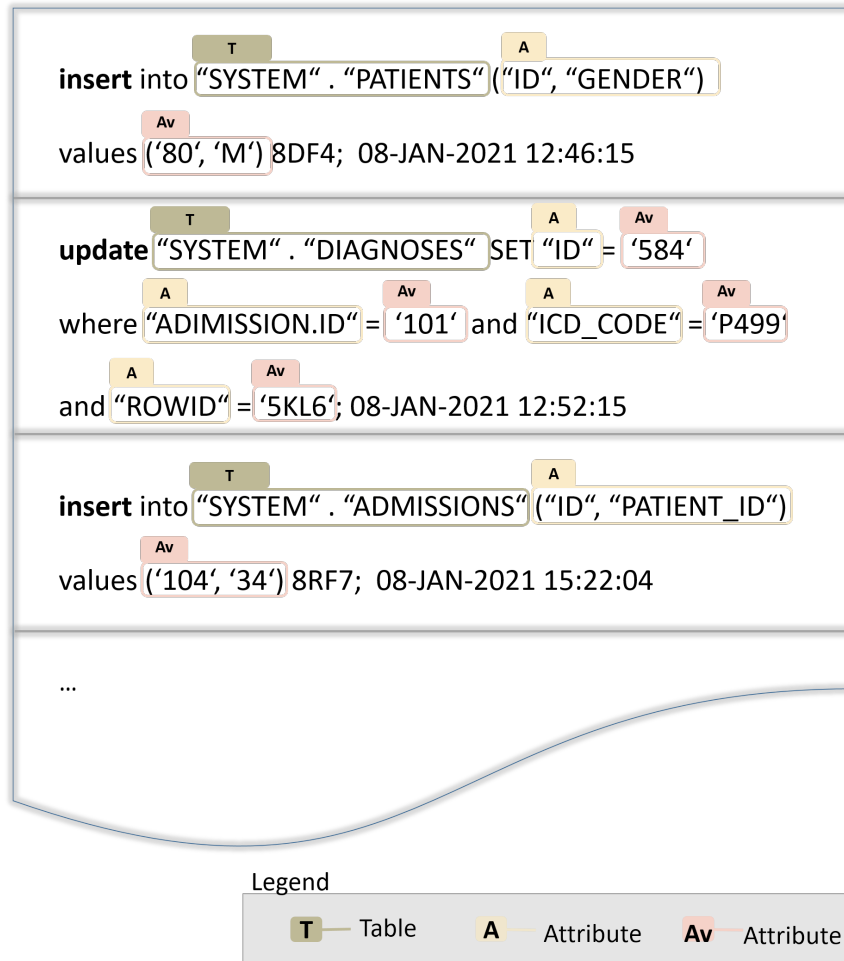


Figure 14: Extraction of tables, attributes and values in each redo log entry

same attribute then this attribute is not any longer a primary key candidate.

Nevertheless, checking for unique value is necessary but not sufficient to detect the primary keys. It might happen that a certain attribute value appears only once in the log (e.g., an attribute which stores the value of an account balance can be easily misinterpreted as a primary key). To solve for this, we check if the attribute values appear in ascending order throughout the redo log entries. The order can be defined by looking at the time of each redo entry. If this not a case, the attributes are not considered as a primary key even if they appear to have unique values.

Finally, to increase the accuracy of the primary key identification step, we are also considering the attribute name's suffix. This is a common practice to understand and maintain the organisation's database, for example, the primary keys contain a suffix like: 'key', 'id', 'nr' [104].

- **Foreign key detection**

The foreign keys are used to set the relations between database tables [141]. In contrast to the primary keys, in a single database table we can have multiple foreign keys, which in turn require the existence of primary keys to reference to [105].

To identify the foreign keys between the previously extracted tables and primary keys we rely on the inclusion dependency notion, which means that all the values of the referencing attribute must be contained in the referenced attribute. For example, if we have two attributes called X and Y respectively, each pertaining in two different database tables, we can say that all attribute values of foreign key Y must be presents in the attribute values of the primary key X. This condition is enough if it is satisfied unidirectionally in that the primary key attribute can contain additional values that do not necessarily reference a foreign key. In the same fashion as for the primary key discovery step we consider also the suffix attribute name to judge if a certain attribute is a good foreign key candidate.

The foreign key detection concludes the database schema discovery step. Before starting the event log extraction the domain expert comes into play to evaluate the discovered schema. She/he has two tasks: evaluate the discovered schema and select the case notion. Once the case notion is selected based on the discovered database schema the redo log entries are correlated to the event log cases. A detailed explanation of this correlation is provided in the next section.

3.2.3 *Event Log Extraction*

The event log extraction step starts with the case notion selection, which is performed by the domain expert based on the desired view of the process. Specifically, the selected case notion denotes which database tables are used to determine the case in the event log. The extraction is achieved based on two steps. The entity relation generation step relates all pairs of tables in a foreign key relation to a so-called entity relation table. The second step, entity-based case collection, collects the event log cases based on the entity relations constructed in the previous step. A detailed explanation of each step is provided below.

- **Entity Relation Generation**

For each pair of tables discovered in the database schema that are involved in a foreign key relation, the following method is applied: for each entity on the left-hand side of the referenced table, it is checked whether one or more entities exist on the right-hand

side of the dependent table (i.e., the one that contains foreign key attribute). If this is the case, then the Row Id of the dependent table, together with the Row Id of the referenced table, are added to an additional table called Entity Relation Table. This is done until all entities on the left-hand side tables are considered. We will repeat the same method for all tables that pertain to a foreign key relation. Therefore, the number of entity relation tables will be the same as the number of relations defined between a pairs of table.

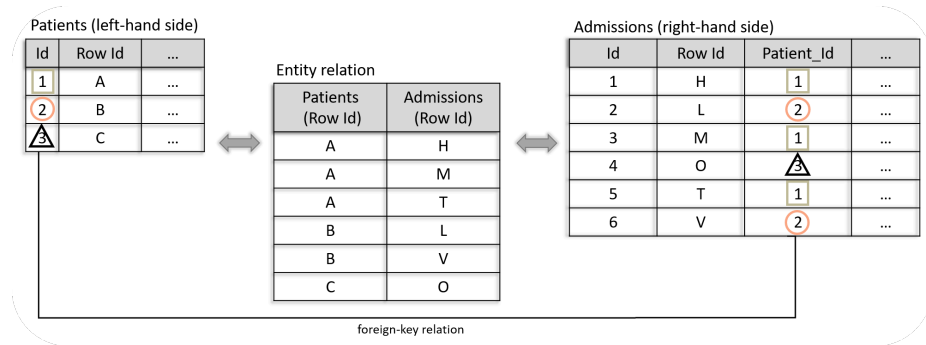


Figure 15: Example of an entity relation generation table extracted from the Patients and Admissions table, which are in a foreign key relation

However, it might happen that several rows in the database might have the same Row Id. This occurs if a delete operation has happened on that row before. If a certain row is deleted from the database, its Row Id can be reassigned to another row via an upcoming insert operation. Therefore, a pre-processing step is needed to ensure that all Row Id are unique for each redo entry. Before constructing the entity relation table, the redo log is parsed and whenever a redo entry that contains a delete operation occurs, its Row Id is tracked. If another insert operation occurs and it occupies one of the tracked Row Id, then a unique suffix is appended to that row. This is repeated until the next delete operation for that Row Id is encountered. In this way, we make sure that each Row Id of the parsed redo log belongs exactly to one redo entry.

Figure 15 provides an example of how an entity relation table can be derived between two tables being in a foreign key relation. Suppose that there are two tables in our schema called Patients and Admissions. Suppose that for each table an attribute called *Id* is discovered. In addition, the *Patient_Id* attribute of the Admissions table (right-hand side) is assigned as a foreign key of the Patients table (left-hand side). Starting from the Patients table, the entity with *Id* equal to 1 is checked to determine whether all entities in the Admissions table have the same foreign key attribute value (i.e., *Patient_Id*). In our example, three entities satisfy this

condition (highlighted with a rectangle). Consequently, the corresponding Row Ids of both tables are added to the entity relation table (i.e., the three first entities in the entity relation table with the Patients (Row Id) equal to A).

The same logic is followed for the next entity of the Patients tables. The entity with Id equal to 2 (highlighted with a circle) is checked if there are some entities in the Admissions table that have the same Patient_Id value. If this holds, the corresponding Row Id values are appended to the entity relation table. Considering our example illustrated in Figure 15, the entity with Id equal to 2, which belongs to the Patients table, is in a foreign key relation (i.e., consider Patient_Id) with the two other entities in the Admissions table. Therefore, the entity relation table is extended with two more entries, where each column stores the corresponding Row Id of both tables in relation. The same logic is followed for all entities that belong to the Patients table.

Since the Row Id in the Patients table are unique values (i.e., after the pre-processing step is sure that each redo entry has a unique Row Id), we argue that the entity relation table can not contain duplicate tuples of Row Id values. Once the entity relation table is constructed for each pair of tables in a foreign key relation, the event log cases are constructed in the next step.

- **Entity-based Case Collection**

This step aims to construct the event log cases by considering as input the entity relation table and redo log entries. Before starting this step, the domain expert has picked the root class, representing the case identifier attribute in the resulting event log. For each Row Id in the root class (from now on referred to as RCID), all corresponding Row Id that are related to it are constructing in an event log case. This implies that the number of event log cases in the final event log will be the same as the number of RCID pertaining to the root class.

To discover all Row Ids in relation to RCID the entity relation table comes into play. Starting from the first RCID, the entity relation table is checked whether some entities are in relation with the selected RCID. Let us call all entities related to RCID target entities relations. Each target entity relation corresponds to one or more Row Ids of another table. Therefore, the same step is followed to find all Row Ids in relation to the second table. In the same way, we iterate through all tables for which a predefined entity relation table exists. For each RCID, an event log case is created as a list of all discovered Row Ids relating to it. Each event in a case has a case identifier equal to the RCID entity value.

Depending on the scope of the to-be-discovered business process model, several optional attributes can be appended to the event log. All the cases defined through this method construct an event log, which can be tailored to the process experts for applying different process mining techniques.

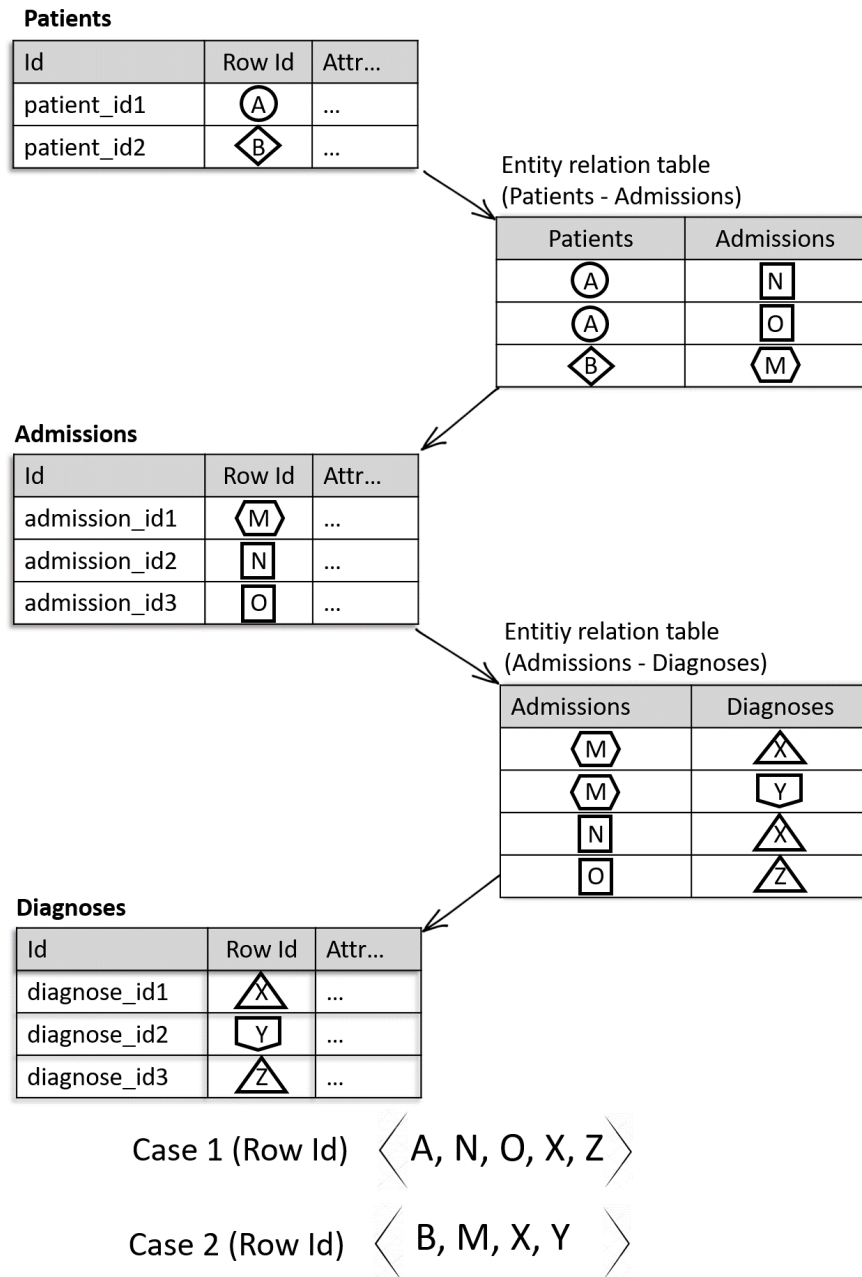


Figure 16: An example of extracting the event log cases after applying the entity-based case collection step on the discovered database schema. The Patients table is picked as a root class by the domain expert

Figure 16 illustrates an example of the entity-based case collection. Suppose that the discovered schema for this scenario has three ta-

bles Patients, Admissions, and Diagnoses. The Admissions table is in a foreign key relation with the Patients table and Diagnoses is in a foreign key relation with Admissions. For each pair of tables in a foreign key relation, the corresponding entity relation table is derived and illustrated in Figure 16. Let us assume that the domain expert has picked as a root class the Patients tables. For the RCID equal to A the entity relation table is checked to identify the target entity relation of the corresponding RCID. Considering our example, there are two target entities, N and O in the entity relation table, which at the same time define the Row Ids of the related table. In the same fashion, we go further and check the related Row Ids of N and O by considering the entity relation table derived between the Admissions and Diagnosis table.

We iterate through all RCID defined in the root class and, in the end, construct a set of Row_Ids that are in relation to RCID. Therefore, the number of cases in the event log is the same as the number of RCID in the root class. Considering the example illustrated in Figure 16, two cases are discovered. The first one contains <A, N, O, X, Z> events, each defined by one Row Id. At the same time, the second event log case contains <B, M, X, Z> events. As the last step, the redo log is queried, and for each Row Id in the discovered cases, the activity names and timestamps are retrieved. The case identifier of the first event log case will have the values of the Row Id equal to A in the Patients table. At the same time, the second event log case will have a case identifier equal to the Row Ids value of B in the Patients table.

3.3 RELATED WORK

Discovering an event log from the redo log is recently subject to research work. Murillas et al. [40] propose three steps approach to extract an event log from the redo entries of the given organization. In the first step, called event extraction, the redo log entries are transformed into the specific structure feasible for manipulation in the upcoming steps of the approach. Once the events are extracted for each redo log entry in the next step, the data model is obtained and used to correlate these events in the event log cases. Data model extraction involves queries on the database tables, columns, primary keys and foreign keys as defined in the corresponding database schema. The last step includes process instance identification, which delineates the case notion selection (i.e., the goal or desired view to obtain the process) and decisions about which events will belong to which event log case. This is made possible through the trace id pattern, which is used to find the common set of attributes between different classes while considering the primary key and foreign key relations.

In [41] Murillas et al. apply the same approach (i.e., like the one introduced in [40]) to the real-life information system, called OTRS process-aware ticked system, responsible for managing the incidents of the IT departments (i.e., customer reporting issue platform). From the same information system, traditional process mining event log approach is applied to extract an event log from the database. Extensive information about the approach used to obtain an event log from the database is outside the scope of this thesis. This study was conducted to provide a comparison on a theoretical and practical level between the traditional process mining approaches for event log extraction (i.e., the event log is extracted from the database) and approaches related to the redo logs (i.e., the event log is extracted from the redo logs). Besides several aspects considered for comparison, the control-flow between the process obtained from the redo log and the process extracted by the traditional approach was one aspect. Authors emphasize that the process model discovered by both approaches mostly represent the same control-flow.

Another approach that deals with the event log extraction from the redo logs is introduced by van der Aalst [123]. The approach assumes that the class model, which is defined as a set of classes connected through relationships (e.g., UML class diagram) is known upfront. The proposed approach is constructed in three steps. The first one scopes the relevant event for the event log extraction. The main question that needs to be answered during this step is: which of the events are relevant to define the case notion and answer the questions one aims to answer? The event selection steps is based on different aspects. One aspect can be focused on a specific time period (e.g., from June 2020 to June 2022). Once the scope is determined in the next step, the correlation of these events to the process instance (event log cases) takes place. During the last step, called classify, the defined process instances from the previous steps are related to processes.

To the best of our knowledge, none of these approaches is able to extract an event log without knowing the class model upfront. This thesis provides a method that considers as input a single source of information - transactions made in the database of the running information system, store in the form of redo logs.

3.4 EVALUATION

The feasibility of the method presented in this chapter is realized based on two synthetic redo logs from two different domain, healthcare and traveling. The method is implemented as a Scala-based CLI tool³ and

³ <https://scala-cli.virtuslab.org/docs/commands/basics/>

can be found on the GitHub repository⁴. To archive and export the redo logs the Oracle LogMiner⁵ is used.

3.4.1 Evaluation Setup

The first redo log is simulated from the MIMIC_III [71] real-life database and it contains redo entries related to the healthcare domain. More specific, the data pertained in this database contain Electronic Healthcare Records (EHRs) related to the patients admitted to the Critical Care Unit (CCU) at the Beth Israel Deaconess Medical center (BIDMC) in Boston, USA. Once the Redo log is obtained, the first step of our method is applied and the database schema is discovered. Afterwards, based on the discovered schema, the redo log entries are correlated with the event log cases (i.e., following the second step of our approach).

The second redo log is extracted from the literature [40] and contains events regarding a Ticket Selling system. It stores information related to concerts, concert halls, available seats, tickets, brands, and booking. In [40] the authors assume that the database schema is known before the event log extraction step takes place (see Figure 17, represented with a gray rectangle). We apply the first step of our method to discover the schema and consequently extract the event log.

In both use cases, the method's feasibility presented in this thesis is proven on the database perspective and not on the process model perspective. For the redo logs related to the healthcare, the discovered database schema is compared with the original database schema illustrated in the MIMIC specification document (the comparison is graphically illustrated with a dashed line in Figure 17). As for the traveling redo log, the discovered database schema is compared with the database schema presented and illustrated in the work [40, p.5].

There are two main reasons why the comparison is performed on the database schema and not at the process model level.

- The event log extraction approach used in the literature is different to our approach. There is not enough details in the literature to replicate the same exact event log extraction. Different event logs would yield different process models, making the comparison futile.
- However, even when the event logs are very similar, the presented process models in [40] are result of several post-processing steps and filtering activities. On top, the process model are discovered for a small subset of activities. Therefore, we compare directly the database schemas.

The main findings of these comparisons are presented below.

⁴ <https://github.com/fyndalf/redo-log-parser> (implemented by Finn Klessascheck and Tom Lichtenstein in the scope of master seminar)

⁵ <https://docs.oracle.com/en/database/oracle/oracle-database/18/sutil/oracle-logminer-utility.html>

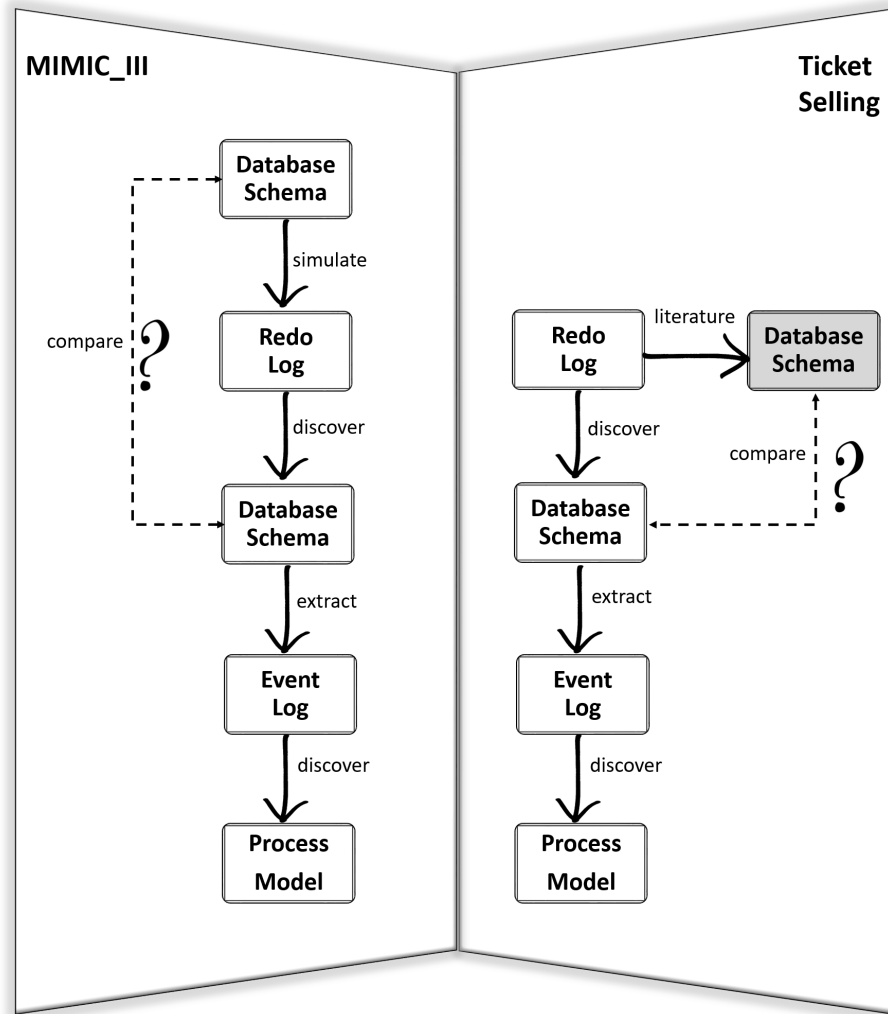


Figure 17: Evaluation setup based on two synthetic redo logs (MIMIC and Ticket Selling). The dashed lines annotate the compared artifacts

3.4.2 Evaluation Results

After applying the first step of our approach on the MIMIC redo log the database schema is discovered and the result is illustrated in Figure 18. The schema contain five classes called: Patients, Admissions, Diagnoses, Pharmacy and Prescription. Patients class contains information about the patient and has two attributes Id and Gender, where Id is assigned as a primary key. Admissions table contain information about the patients admission into the hospital and has also two attribute, Id which serves as primary key of this table and Patient_Id, which stores the patient identification number. It is possible that the same patient is admitted several times into the hospital. Therefore, a relation between these two classes is discovered, which is handled via the primary key Id of the Patients table and Patient_Id attribute of the Admissions table. Besides that, the Admissions table is in relation with two other tables,

called Diagnoses and Pharmacy. The former contains three attributes (Id, Admission_Id and ICD_Code) and the latter contains the same amount of attributes (Id, Admission_Id and Frequency). The relation between these three tables is handled via the Id attribute in the Admissions table and the corresponding Admission_Id of two other classes respectively. In addition, the Pharmacy table is in relation with the Prescription table, which contain five attributes; Id, Pharmacy_Id, Name, Dose_Amount and Dose_Unit. The relation is established through the Id attribute of the Pharmacy table and the Pharmacy_Id attribute of the Prescription table.

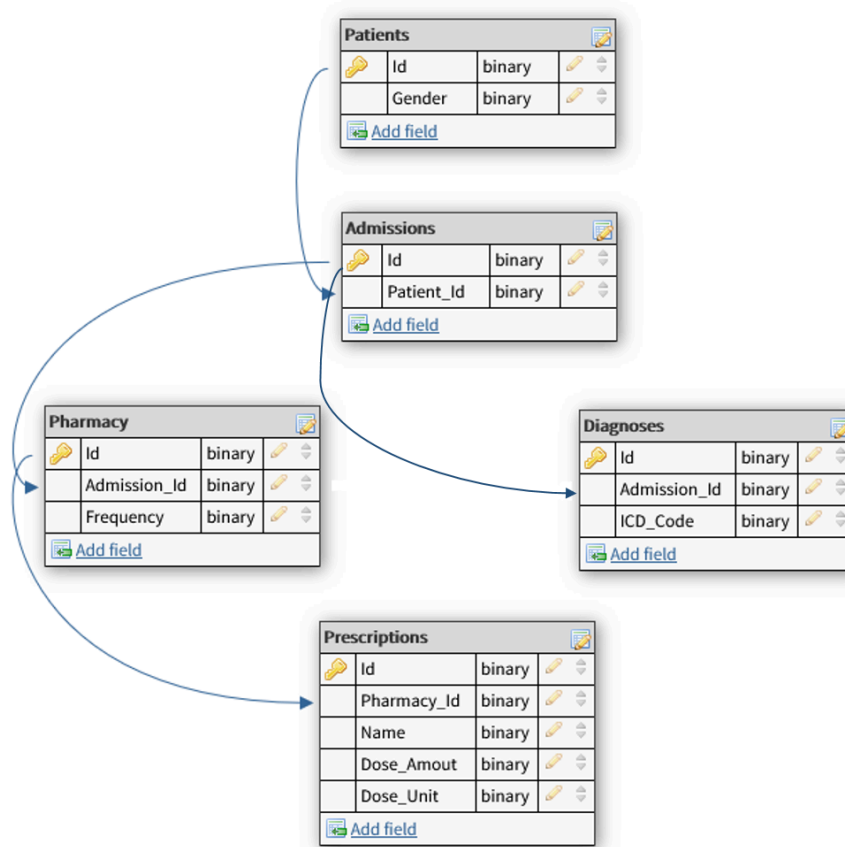


Figure 18: The discovered database schema after applying the first step of our approach. The redo logs are extracted from the MIMIC database

After manually comparing the discovered schema with the original one illustrated in the MIMIC specification document⁶, we come to the conclusion that all the classes and attributes in the discovered schema match all the classes and attributes pertaining to the MIMIC specification document. Note that for our experiment we considered only a subset of classes from the MIMIC database (due to its large size) and the redo log was simulated for the same subset.

⁶ <https://mit-lcp.github.io/mimic-schema-spy/relationships.html>

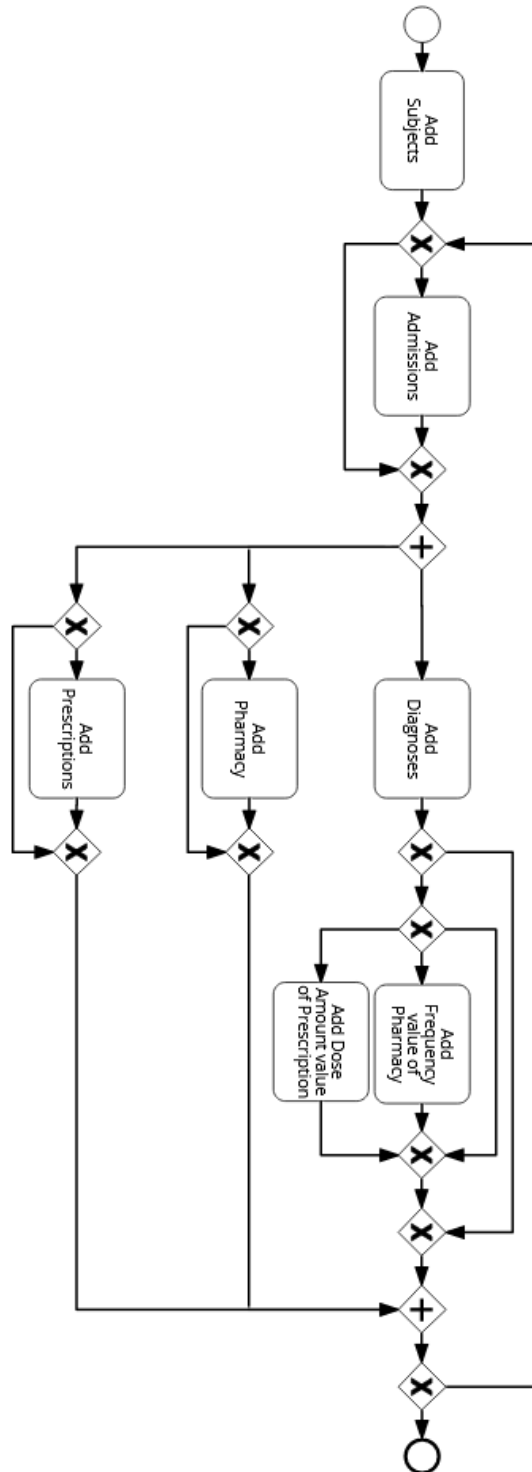


Figure 19: The discovered process model from the extracted event log after applying the method presented in this chapter

After applying the second step of our method (i.e., Event Log Extraction) the event log is extracted based on the database schema discovered in the previous step. Under the assumption that the Patients table is

selected as a root class (i.e., case notion) by the process expert the process model is discovered. We are using the Inductive Visual Miner [76] discovery algorithm to derive the process model from the event log.

From the discovered process model, illustrated in Figure 19 we can observe that the process starts by registering the patient⁷, which is reasonable because we are assuming that the Patients table is selected as a case notion by the domain expert. Afterwards, the patient is admitted into the hospital, which is marked as optional. Next, the patient is diagnosed and the Prescriptions or Pharmacy are added but are not mandatory. The table creation process is followed by either updating the frequency of the pharmacy table or by updating the dose amount of the prescription table, which also marks the end of the illustrated process.

Considering the Ticket Selling redo logs, the database schema is discovered and illustrated in Figure 20, which is self explanatory. We compared the discovered schema with the one presented in [40, p.5] and concluded that we discovered only one extra relation between the *Price* attribute in the *Ticket* table and *No*⁸ attribute in the *Seat* table. To investigate the reason behind this difference, we went one step further and checked the data in the redo log files. We concluded that there might be an inconsistency in the original data because every ticket in the redo logs has a price equal to 35, and no seat has a seat number equal to 35. Therefore, we are assuming that this is a mistake in the data simulation process and does not reflect the reality.

⁷ Subjects table in the MIMIC database represents Patients

⁸ based on the description provided in [40] No attribute refers to seat number

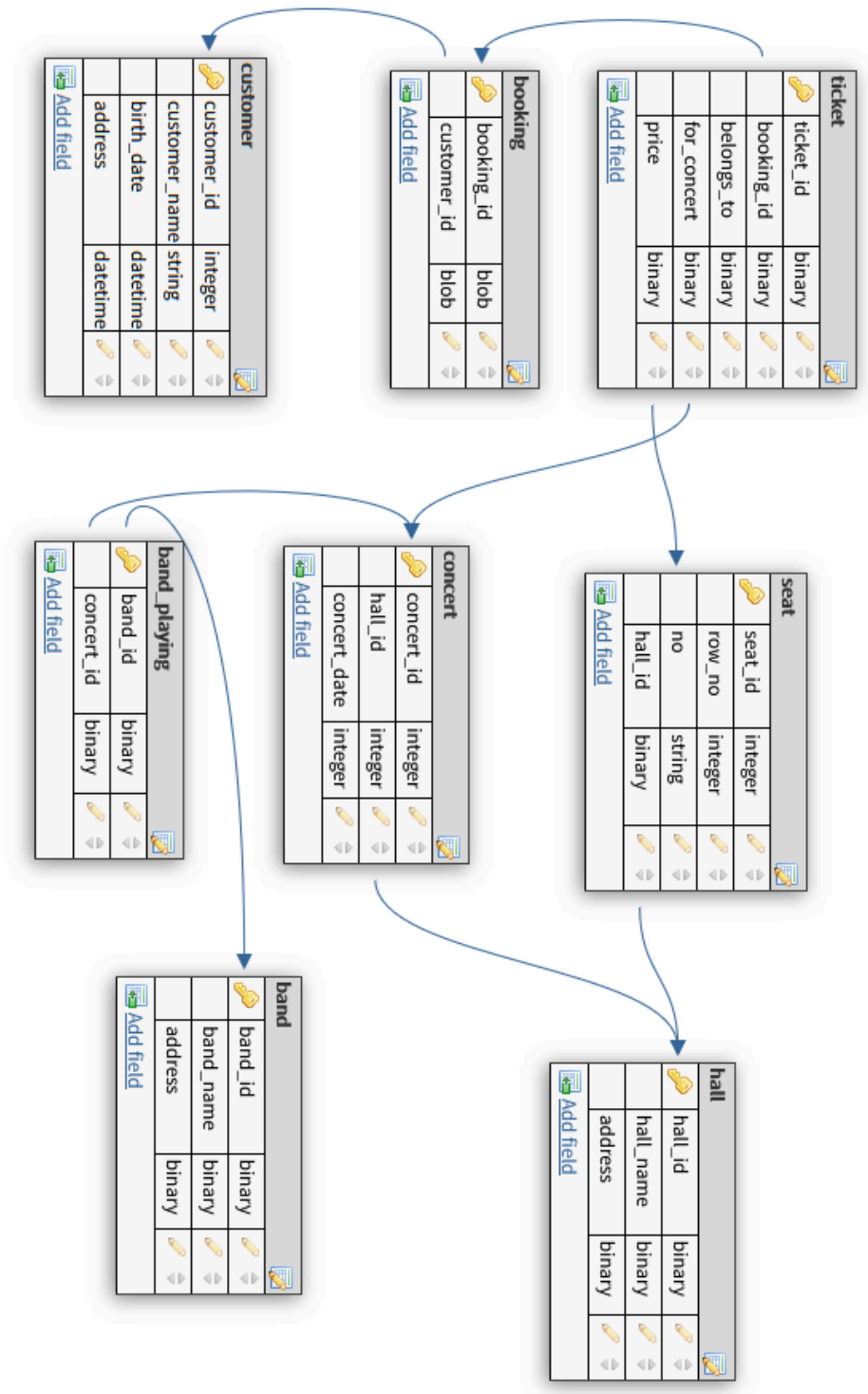


Figure 20: The discovered database schema after applying the first step of our approach. The redo logs are taken from the provided data in the literature

3.5 SUMMARY

This chapter proposes a method to extract an event log from the transactions made by a running information system. Considering our story, we enable John to extract an event log from such transactions without accessing the whole database or having extensive knowledge about its schema. To achieve this, we present a two-step semi-automatic method. First, the database schema is discovered solely from the redo log. Then, the event log extraction is achieved by considering both the discovered schema and redo log as inputs. The assistance of the domain expert, like John, is only required in between the steps to evaluate the discovered schema and select the case notion based on the targeted process model.

The method's feasibility is proven by developing a prototype, which is applied on two synthetic redo logs. The discovered database schemas are shown to be similar to the original one proving the effectiveness of our method. The discovered event log conforms to the XES standard and can be used by process mining experts to apply different process mining techniques based on their needs.

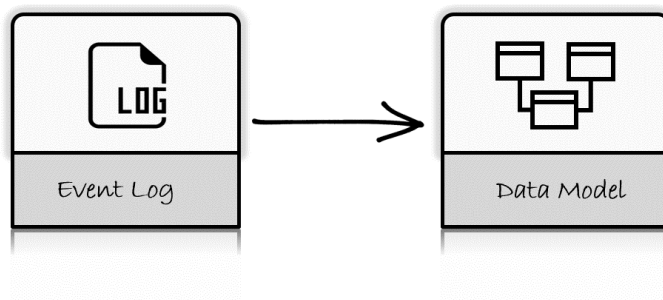


Figure 21: Discovering a data model from event log

This chapter provides a method to discover a data model by just considering as input an event log. Considering our story, we enable John to use the resulting data model as complementary information of the discovered process model (i.e., the process model is discovered from the same event log). This chapter is mainly based on the previous work presented in [6] while an application scenario is achieved in the scope of Tom Lichtenstein's master thesis [79]. In addition, in another [10] we provide another application scenario of the data model, which is upfront discovered from an event log.

4.1 MOTIVATION

In many real-world scenarios, the event logs are extracted from the databases of a given organization [44]. After the extraction process takes place, the event logs are made available to the process mining experts. Being involved in process mining projects these experts are challenged with tasks related to the discovery or improvement of the operational business processes, which always considers an event log as a starting point. Most often, event logs do not explicitly provide information about the context of the data they are extracted from. The data perspective is overlooked and not clearly visible to the process experts [55, 100, 114]. Despite the fact that event logs capture behavioral information, we argue that they are also a rich source of domain-specific information, albeit not explicit. The data perspective is an essential aspect that complements the process mining procedure with useful information. It plays an important role in the understandability of the event log and, in consequence, the process model. To this end, we propose in this chapter a semi-automatic method for discovering a complementary UML data model [108] from an event log. We prove the feasibility of the proposed method by applying it to a real-life event log.

4.2 OVERVIEW OF THE DATA MODEL DISCOVERY METHOD

Deriving a data model from an event log implies systematically deriving the individual UML language [95] (our language of choice as it is widely used as standard for modeling data relations) constructs: classes, class attributes and the associations between classes. To this end, we are using a two-step semi-automatic method as depicted in Figure 22. The first step is automatic and introduces an intermediate representation that captures the access relations between all activities and attributes pertaining to the event log. This representation is called Activity-Attribute relationship diagram (A2A), which is inspired by [125]. In the second step, a set of rules are applied to the A2A diagram to cluster attributes into classes and identify the relationships between these classes, which ultimately constitute the discovered data model. A detailed explanation of each step is given below.

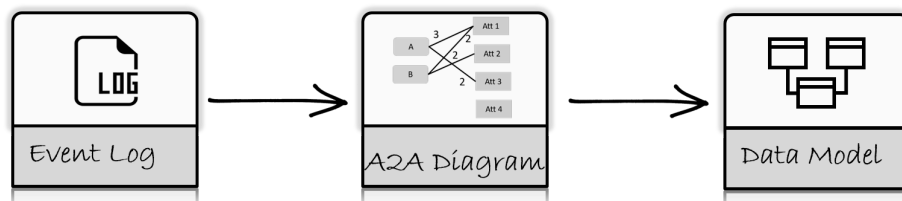


Figure 22: Overview of the data model discovery method

Before explaining in more details the method for discovering a data model from an event log, let us first formal define the notion of an access relation between an activity and attribute pertaining to an event log.

Definition 4.1.

(Activity-attribute access relation) Let El be an event log, A the set of all unique activities, Att the set of all attributes and $M_{att} = \{Case, Act, Time\}$ a set of mandatory attributes. We define $\#_{att}(e)$, where $att \in Att$, as the value of attribute att for event $e \in El$. We say an activity $a \in A$ accesses an optional attribute $att \in Att \setminus M_{att}$ iff $\exists e \in El \mid \#_{act}(e) = a \wedge \#_{att}(e) \neq \perp$. Let us denote $r = (a, att, n)$ the access relation between an activity $a \in A$ and an attribute $att \in Att$ in the event log, where $n \in \mathbb{N}$ is the occurrences of the relation in a log. The set of all access relations in the event log is defined as R . ◀

4.3 DERIVATION OF THE ACTIVITY ATTRIBUTE RELATIONSHIP DIAGRAM

The most prominent information about the attributes that we have from an event log is their relations with the activities. To put this information to the forefront, the A2A diagram is derived. It serves the role of an intermediate representation that acts as input for discovering the data model.

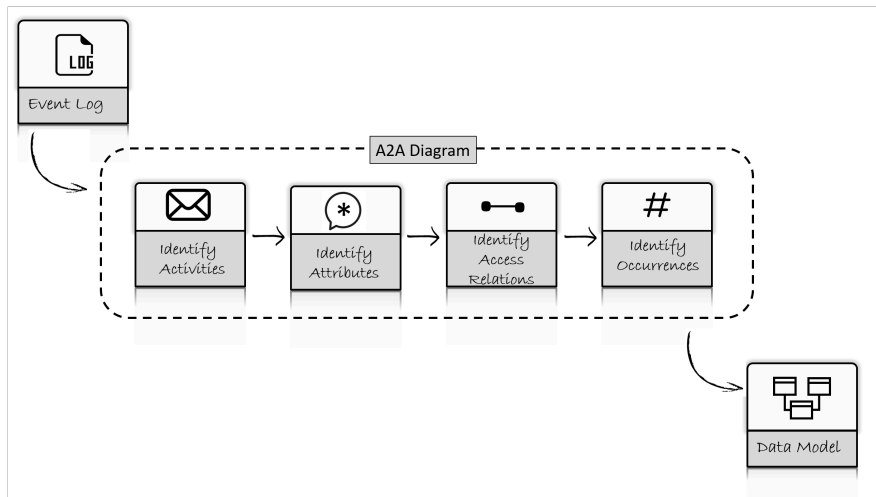


Figure 23: Derivation process of the activity attribute relationship diagram

Figure 23 illustrates the derivation process. The starting point for constructing such a diagram is to identify the activities and attributes pertaining to an event log. Therefore, at the model level we first derive the set of all activities. Secondly, all attributes, except the meta-attributes (i.e., case identifier, activity name and timestamp) are identified. Once the set of activities and the set of attributes are established we have to

derive their relations. To this end, we look at the access relationships between activities and attributes based on the Definition 4.1. In the event log, this relationship is detected inside a single event in that there is a common event where activity *A* and a new value of attribute *Att1* appear. The number of times an access relation between an activity and attribute takes place in the entire event log, independently of the case, is called the access occurrence number and it is graphically represented over the access arrow connecting an activity and attribute in the A2A diagram.

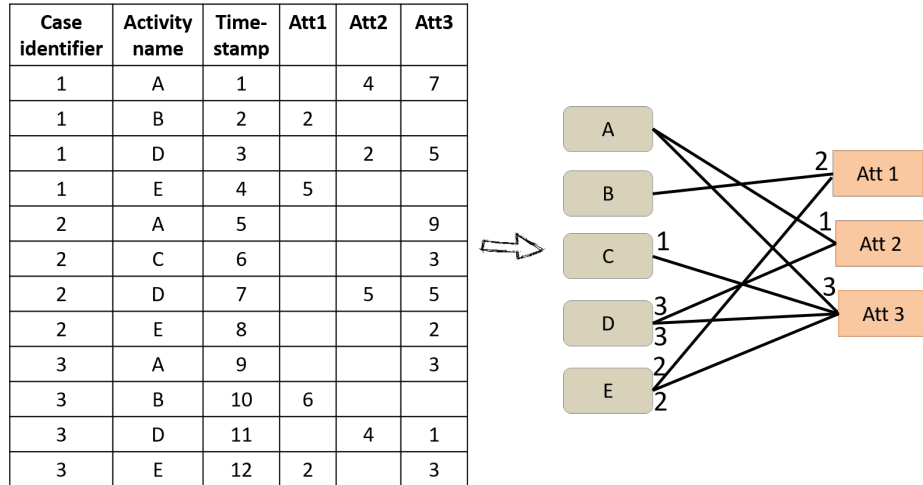


Figure 24: An example of the A2A diagram derived from the event log

To give an example, let us extend the event log illustrated in Section 2.3.2 (cf. Figure 11) with a set of optional attributes *Att1*, *Att2* and *Att3*. The extended event log and the A2A diagram derived from it, following the steps introduced above, are depicted in Figure 24. We see that, at the process model level, there are five activities (activity models) (*A*, *B*, *C*, *D*, *E*) and three optional attributes (*Att1*, *Att2*, *Att3*). The relations between activities and attributes are derived based on the write operations (i.e., access) performed between them, while how often these operations take place is captured by the access occurrence number next to each access arrow. In this example, the activity *A* accesses the *Att3* three times while attribute *Att1* only one time. This diagram clearly shows which activity access which attributes, at what frequency, and more importantly, which attributes are accessed by multiple activities and which share common activities. As we will see next, the latter is very important for discovering the data model.

The algorithm for deriving the A2A diagram from an event log is described in Algorithm 1.

Algorithm 1 A2A diagram derivation from an event log

```

input : Event log El
output : A2A = (A, Att \ Matt, R)
           Activity-Attribute access relation diagram
initialization A: empty set of activities, Att \ Matt: set of attributes
without the meta-attributes, R ⊆ A × Att \ Matt × N: empty set of
access relations
// create a unique set of activities
for e in El do
  if #act(e) ∉ A then
    | add #act(e) to A
  end
end
// populate the set R
for a in A do
  for att in Att \ Matt do
    | int n = 0 // n represents the access relation occurrence
    | for e in El do
    | | if #act(e) = a ∧ #att(e) ≠ ⊥ then
    | | | n = n + 1
    | | end
    | end
    | add r = (a, att, n) in R
  end
end
print(A, Att \ Matt, R)

```

4.4 DATA MODEL DISCOVERY

Discovering the data model implies the need to discover the data model classes with their attributes and the associations between the classes. To achieve this, we look at the relations between two or more attributes in the A2A diagram and consider whether they belong to the same data model class. After exhaustively going through all the attributes and grouping them into the UML classes, we identify the UML associations between those classes. Defining the UML associations entails specifying their multiplicity.

Below we provide a set of rules that are applied to the A2A diagram to group the event log attributes into the data model classes. These rules are organized based on two aspects: non/isolated attributes and non/isolated activities, which are defined as:

- An attribute is called isolated if all activities that access it do not access other attributes.
- Likewise, an activity is called isolated if all the attributes it accesses are not accessed by any other activity.

A detailed explanation of each rule is provided below.

- **Rule 1: isolated attributes, isolated activities**

We identify the isolated access relations in the A2A diagram, in that an attribute is accessed only by a single activity and the activity accesses only the said attribute. In this case, the rule is to assign all isolated attributes to separate independent UML classes. At this stage, there is no other information in the A2A diagram that can give insights about how the generated UML classes could be related.

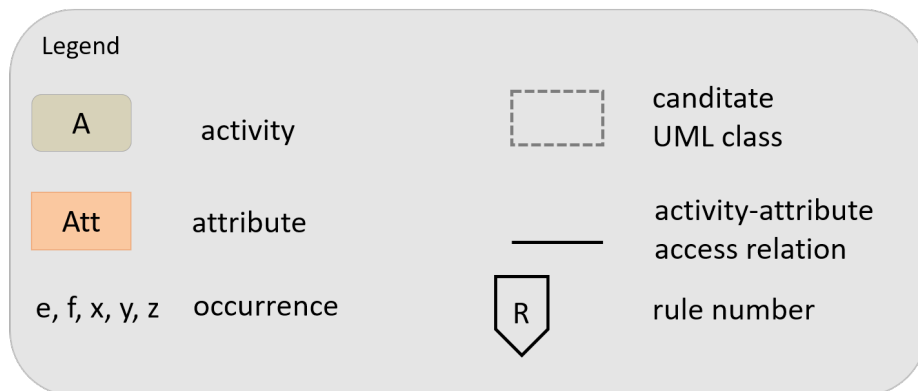
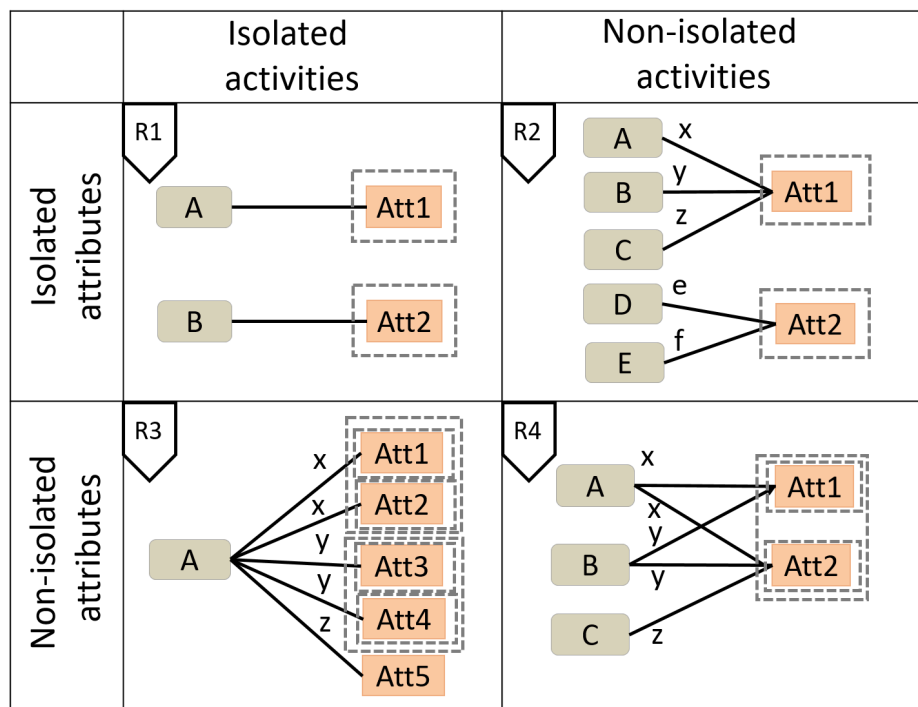


Figure 25: Rules organized based on two aspects (non/isolated attributes and non/isolated classes) for deriving the data model classes

An example is illustrated in Figure 25 (R1) where activity A and attribute $Att1$ are isolated because A accesses only $Att1$ and $Att1$ is accessed by said activity. The same holds for activity B and $Att2$. The $Att1$ and $Att2$ are assigned to a separate UML classes (represented in Figure 25 by a dashed-line rectangle).

Last, the isolated attributes assigned to the UML class together with their accessing activities are removed from the A2A diagram. This action takes place at the end of each rule.

- **Rule 2: isolated attributes, non-isolated activities**

In this case, we search the A2A diagram for isolated attributes that are accessed by non-isolated activities. Similar to rule R1, the attributes are isolated and, thus, there is no additional information on how they can be grouped into classes. Hence, the isolated attributes will be each assigned to a separated class.

As it is illustrated in Figure 25 (R2) activity A , B and C are accessing $Att1$ with the different cardinalities. The same holds for $Att2$, which is accessed by activity D and E . Based on this rule, $Att1$ and $Att2$ are placed in two independent UML classes (represented with dashed lines in Figure 25).

- **Rule 3: non-isolated activities, isolated attributes**

If at least one common activity accesses two or more attributes, then the attributes are said to be related. We are looking specifically for related attributes that may belong to the same class. We argue that if an activity accesses two or more attributes with the same occurrence then these attributes are highly likely to be contained in a single class. Therefore, we group these attributes based on common occurrences. However, we cannot deduce from the A2A diagram alone whether the attributes are accessed simultaneously by the activity. It may happen that in total these attributes are accessed the same amount of time by the activity but never in the same event. This means that the attributes are highly likely to not belong to the same class as they seem to be accessed independently. To counter this problem we offer the following solution.

Let E_{A1} be a set of events from the event log where activity A accesses attribute $Att1$. $|E_{A1}|$ denotes the access occurrence. Similarly, we define E_{A2} as the set of all events where the activity A accesses attribute $Att2$ with occurrence $|E_{A2}|$, where $|E_{A2}| \geq |E_{A1}|$. The decision of whether attribute $Att1$ and $Att2$ belong to the same class is made based on the following function:

$$\text{rel}(E_{A1}, E_{A2}) = \begin{cases} \text{one class} & , \text{ if } E_{A1} \cap E_{A2} = E_{A1} \\ \text{independent classes} & , \text{ if } E_{A1} \cap E_{A2} = \emptyset \\ \text{dependent classes} & , \text{ if } 0 < |E_{A1} \cap E_{A2}| < |E_{A1}| \end{cases} \quad (1)$$

Attribute 1 and 2 belong to the same class if set E_{A1} is a subset of E_{A2} because anytime the activity A accesses attribute 1 it also accesses attribute 2. Both attributes define a new UML class, however, attribute 1 is marked as optional because it is not always accessed when attribute 2 is accessed.

If the two sets are disjoint (i.e., $E_{A1} \cap E_{A2} = \emptyset$), then attribute 1 and 2 are not in the same class and, moreover, these classes have no association between them. This is due to attributes 1 and 2 happening independently of each other.

Finally, there are events in which attribute 1 and attribute 2 are accessed simultaneously, except the first case. This means that there are some events where the attributes 1 and 2 are accessed by the same activity A but this number of events is not the same as $|E_{A1}|$. In this case, the attributes are placed in different classes, but the classes are still related via a bidirectional association. The multiplicity of the association is 0..1 to * from the class containing attribute 1 to the class containing attribute 2.

In a more general case, where the number of attributes that share the same activity with the same occurrence is more than two, we apply the above function to every pair of attributes to determine the resulting classes.

This rule is illustrated in Figure 25 (R3). Activity A accesses $Att1$ and $Att2$ with the occurrence x , $Att3$ and $Att4$ with the occurrence y and $Att5$ with occurrence z . The decision of $Att1$ and $Att2$ belonging to the same UML class or not depends on whether the events where the activity A accesses the $Att1$ are the same events where the same activity accesses $Att2$. The same holds for the $Att3$ and $Att4$ accessed with cardinality y by the same activity.

At last, the attributes assigned to the corresponding classes are removed from the A2A diagram. If their accessing activities do not access other non-isolated attributes, they are removed as well.

- **Rule 4: non-isolated activities, non-isolated activities**

Every relation that cannot be expressed by the previous rules is captured by this rule. In this case, activities and attributes are non-isolated, which means that an attribute is accessed by several activities and each activity accesses several attributes.

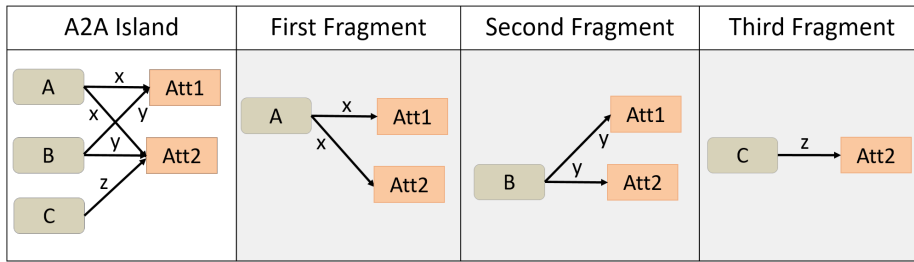


Figure 26: Decomposed A2A diagram into three fragments after applying the fourth rule

After removing the attributes and activities that satisfied the previous three rules we are left with an A2A diagram that contains one or more disconnected subgraphs (i.e., interconnected activities and attributes), which we are referring to as islands. In Figure 25 (R4), there is only one island, but it can happen that another set of non-isolated activities and attributes, which has no relation to the first set, can be left in the A2A diagram. That is why we call these sets islands.

To group the attributes into UML classes, each island is decomposed into smaller A2A diagram fragments for each activity. This means that the number of fragments is the same as the number of activities on an island. The attributes that are accessed by the activity are represented in the respective fragment. Hence, an attribute may appear in one or more fragments (see Figure 26).

The resulting fragments can satisfy either rule 1 or rule 3 but not rule 2 because the fragments contain only isolated activities. The grouping of the attributes, then, follows the rule 1 or 3. However, since attributes may belong to two or more distinct fragments, there is a conflict that needs to be resolved. For example, it might happen that the same attribute is grouped either in a standalone UML class or in a class with some other attributes depending on the grouping results from each fragment. In this case, we leave the choice to the user of the method to make a decision that better fits the overall result.

Figure 26 shows an island fragmentation example. Activity *A* accesses attributes *Att1* and *Att2* with occurrence *x*. The same holds for activity *B* except the occurrence, which is *y*. Last, activity *C* accesses attribute *Att2* with occurrence *z*. Based on this rule the A2A diagram is decomposed in three other A2A fragments, one for each activity.

In the first and the second fragment, we are dealing with non-isolated attributes and isolated activities. Therefore, rule 3 is applied to derive the classes. In the third fragment, rule 1 is applied because activity *C* and attribute *Att2* are both isolated.

After all the classes are created, the classes can be named based on the activity they were generated from. Finally, we have to consider the associations between the remaining independent classes. To this end, we will consider the most frequently accessed class as the root class, which has the highest potential to represent the business process case notion. Then, we introduce an association between the remaining classes and the root class. Their multiplicities are set based on the occurrences from the A2A diagram.

4.5 RELATED WORK

Discovering a data model from the event log is a relatively new development in the process mining area. However, the aggregation of information spanned over a set of business process models into one data model is widely researched in the literature [32, 33].

Cruz et al. [32] present an approach to derive a data model from a private process model, which is then used in the software development process. The authors argue that the data model can be easily used as a language between different roles within the organization (e.g., between business process analysts and software developers), increasing the participants' understandability. The authors use a three-step approach: first, the entities are defined by considering the data stores and the roles played by the participants; second, the relationship between entities is deducted based on the way the participants and activities manipulate the data stores; third, the attributes related to the participants and data stores are determined. Since the input information of the presented approach is a process model, the generation of the data models is manually performed and mainly focused on the elements of the process modeling language, such as BPMN. Therefore, data stores and process participants in the business process model are used as a starting point to identify the data model entities. The relationship between those entities is determined based on the information exchanged between participants and activities connected to the data stores. Nonetheless, if this type of information is not captured in the process model, then the identification of these entity relations will be missing in the data model. One of the most significant limitations of this approach is that it only deals with one business process model. However, a software application can support more than one business process in real organization settings. Therefore, the approach is further extended in [33] to support the aggregation of information spanning over a set of related business process models into one data model. In contrast to [32], the proposed approach is conducted in a semi-automatic fashion.

In this thesis, we propose a semi-automatic method that takes as input an event log rather than a business process model. We argue that discovering a business process model from an event log comes with

losses in valuable attribute and occurrence information that the process model cannot always capture.

Breitmayer et al. [23] discovered a data model as an intermediate step for discovering object-aware processes. Each table in the database belongs to an object in the data model, whereas the database columns represent object attributes. The relation between data model objects is constructed by considering the primary keys defined in the database. The resulting data model is a crucial step for object-aware process model discovery because it is the foundation for both lifecycles and object coordination. In contrast, our method relies only on the event log to discover the data model and use it as a complementary artifact to understand the process model.

Brdjanin et al. [21, 22] present an online platform that automatically generates a data model (represented as the UML class diagram) from the business process models represented as BPMN. The authors introduce a two-phase approach where a simple domain-specific language, called extractor, is introduced as an intermediate representation in the first phase. During this phase, only specific concepts are extracted from the source model. Different source notions require different extractors, making the approach bound to the business process model notions. In the second phase, the data model is generated based on the intermediate representation of the extracted concepts, which is achieved by a single transformation called generator¹. In contrast, the method presented in this thesis is not bound to the process model notion since our input information is an event log and not a process model. Therefore our approach can also support the undesirability of a process model discovered not only as BPMN but also e.g., as Petri Nets [81].

In [78], the authors provide a richer event log, compared to XES, called eXtensible Object-Centric (XOC) by considering multiple objects, which are an abstraction of the data elements like records in the database table of a given information system. This implies that the object-centric systems records the transactions which belong to the same category (e.g., patient) in the same table (e.g., patient table). Each event may refer to any number of objects in contrast to the traditional XES format where a single case notion is considered, and every event belongs to exactly one case. Constructing a data model from an XES event log is more complicated than deriving it from the XOC format because of the single case notion perspective of XES. We argue that XOC holds more information about the data model because the object relations can be derived from the global perspective (rather than the case perspective) of the events.

Comparison based on different aspects of the main approaches presented in this section, which deal with the data model generation, is shown Table 1. Obviously, there is no similar work that tries to infer a data model by considering the historical data extracted in the form of

¹ the generator used in [21] is based on ATLAS Transformation Language (ATL)

an event log. In addition, such generation aims to support the process experts in understanding the discovered process rather than supporting the software products manager during the software development process.

Table 1: An overview of the related work dealing with the data model generation

Ref.	Input	Purpose	Approach	Manner
Cruz et. al [32]	Single BP	During the software development process	BPMN	manual
Cruz et. al [33]	Set of BP	During the software development process	BPMN	semi automatic
Breitmayer et. al [23]	Database	During the discovery of object-aware processes	Database	not specified
Brdjanin et. al [21, 22]	Set of BP	During the software development process and database designers	Domain specific language	automatic
Our method	Event Log	Complement the discovered process model with data model	Based on A2A diagram	semi automatic

4.6 EVALUATION

The feasibility of the method presented in this chapter is evaluated based on the Road Traffic Fine Management (RTFM) event log [87]. The RTFM event log is taken from the information system of the Italian police. The event log contains information regarding the road-traffic fines and includes 150.370 cases (561.470 events) processed by the municipality over three years (January 2010 - June 2013). An explanation of all attributes pertaining to this log is provided in Table 2. To provide a behavioral overview of the event log, we show in Figure 27 the process model (represented as BPMN [139]) that is discovered by applying the Inductive Miner algorithm [76]. Some activities that do not access any attribute are excluded from the process model without undermining its meaning.

As it is depicted in Figure 27, the process starts with *Create Fine* activity. After the fine is created, it can be sent to the offender via *Send Fine*. The offender has the option to pay the fine immediately after it is handed over to him (*Payment*). If this is not the case, the date when the offender receives the fine is registered (*Insert Fine Notification*). If the

Table 2: The description of the RTFM event log attributes

Attribute	Description
Resource	Fine creator
Amount	Fine amount that has to be paid by the offender
Article	The law article based on which the fine is created
Dismissal	The reason why the fine is dismissed
Expense	Additional payment
Last Sent	Fine sent type e.g., by post
Matricola	Offender's matriculation number
Notification Type	The way how the offender is notified for the fine
Payment Amount	Paying amount when the payment of done in installments
Points	The number of points deducted from the driving license
Total Payment Amount	The total payment amount paid by the offender
Vehicle Class	The type of the automotive vehicle

payment will not take place (i.e., within 60 days), then a penalty (*Add Penalty*) is added to the fine. The offender has the option to appeal against the fine through the Judge (*Appeal to Judge*) or Prefecture (*Send Appeal to Prefecture*). If the appeal is successful, then the process ends. Otherwise, the fine is sent for credit collection (*Send for Credit Collection*), marking the process terminations.

Before generating the A2A diagram, the RTFM event log is cleaned from the activity-attribute access relations with the value 0 (i.e., the activity always access the attribute with the value 0). For example, *Create Fine* activity accesses the *Total Payment Amount* always with the value 0. The same holds for *Matricola* and *Resource* attributes accessed by *Appeal to Judge* activity.

Applying the first step of our approach to the event log generates the A2A diagram illustrated in Figure 28. The activities that do not

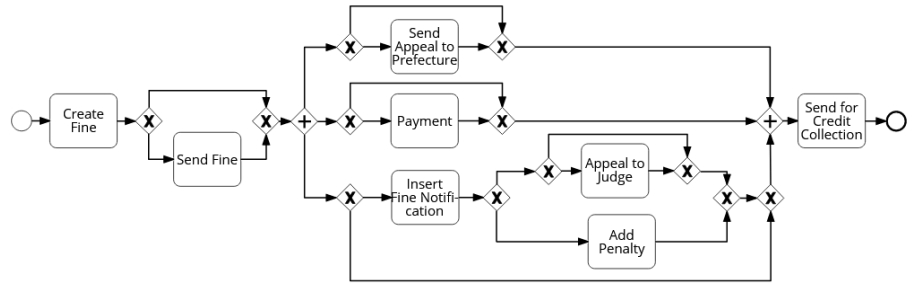


Figure 27: Process model discovered by the Inductive Miner algorithm [76] from the RTFM event log [87]

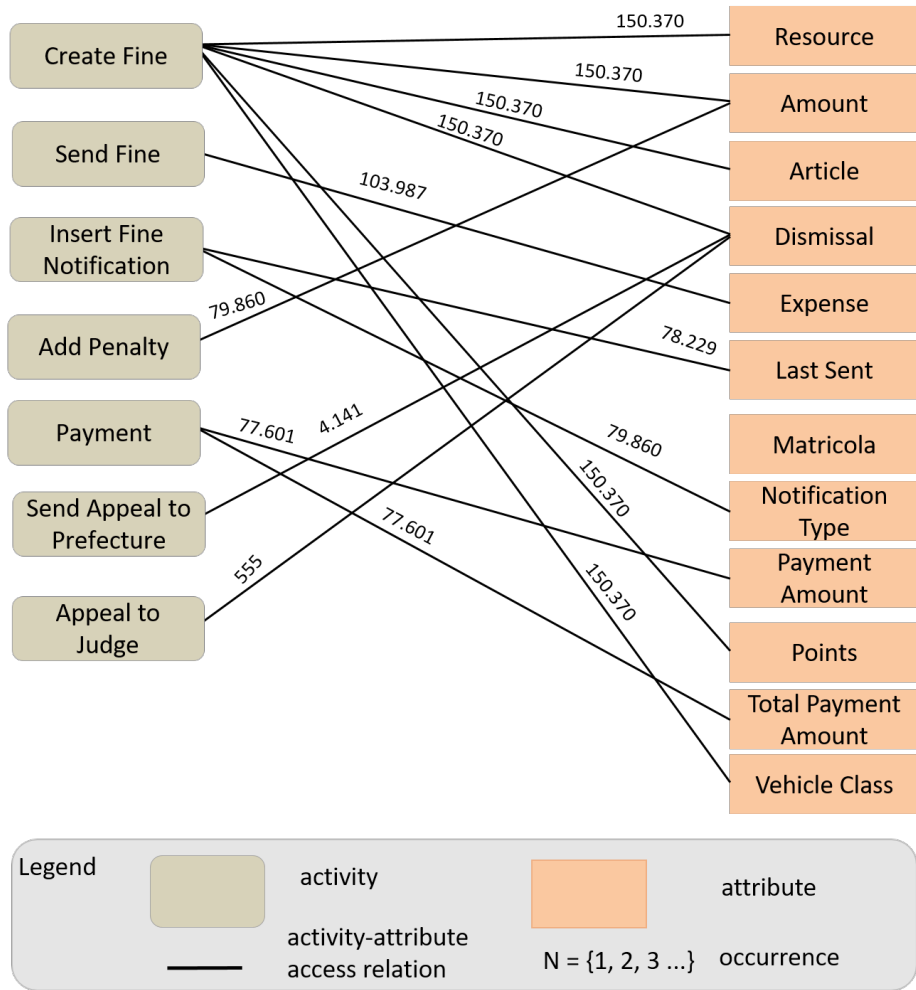


Figure 28: The A2A diagram generated from the RTFM event log after applying Algorithm 1 presented in Section 4.3

access any attribute and all access relations discarded from the clean-up phase are not shown in the A2A diagram. For example, *Matricola* is represented as a stand-alone attribute in Figure 28 as it is always accessed with value 0.

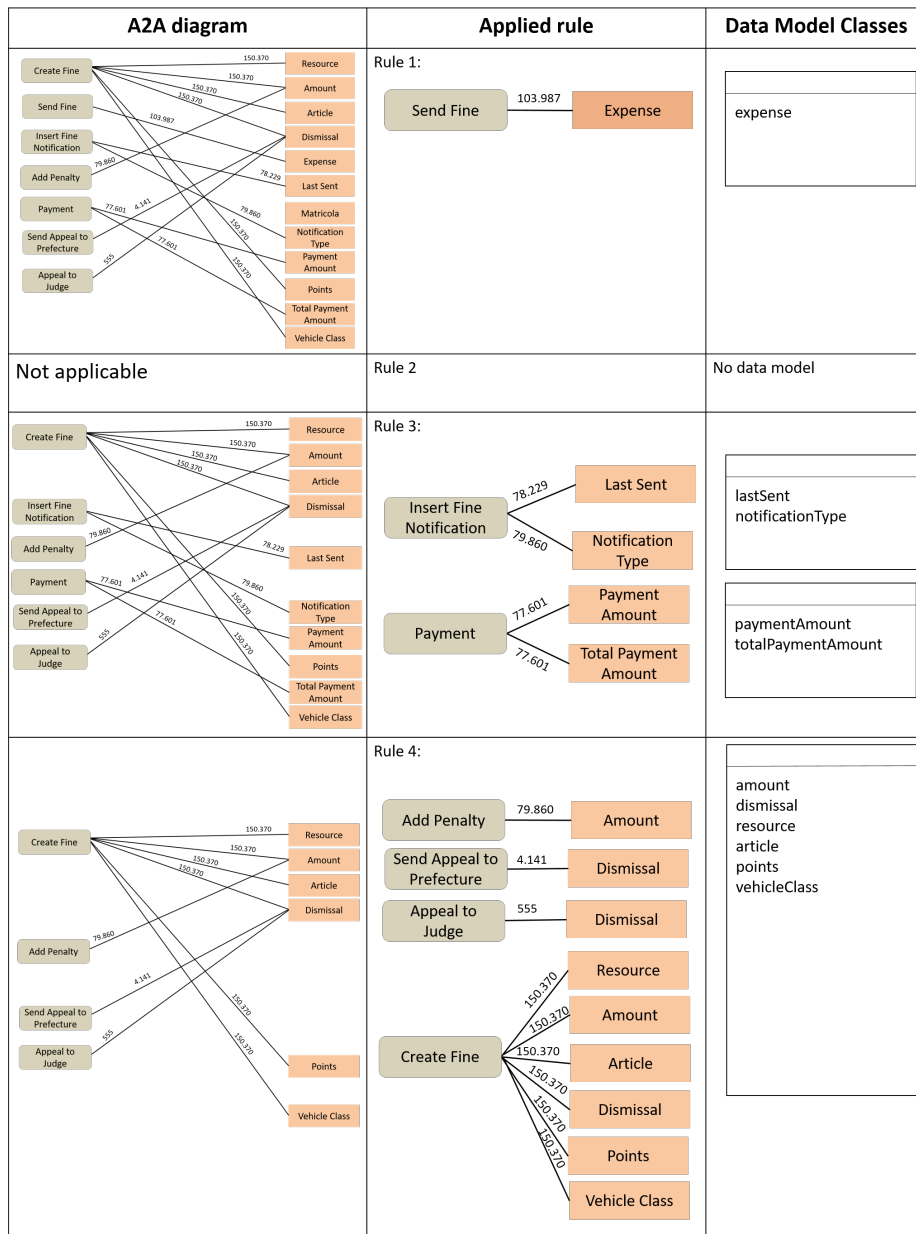


Figure 29: The rules applied to the A2A diagram of the RTFM event log and the resulting data model classes

In the second step, the A2A diagram is used as an input for discovering the data model. Figure 29 depicts the application of the rules from the second step of the approach to the generated A2A diagram. The rules are applied following the defined order (R1 to R4). If a rule is satisfied, all activities and attributes related to that rule are excluded from the A2A diagram and the attributes are assigned to the respective classes. This is repeated until all attributes are grouped into UML classes and there are no attributes left in the A2A diagram.

As it is shown in Figure 29, rule R1 is fulfilled by *Send Fine* activity and *Expense* attribute, where both are represented as isolated in the

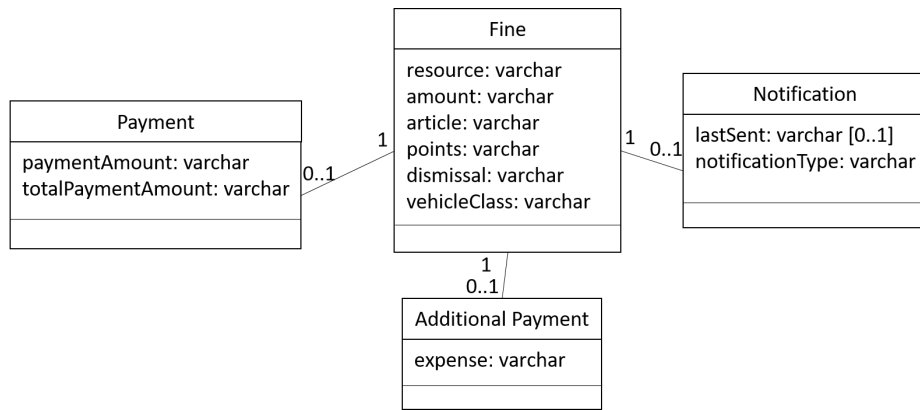


Figure 30: Resulting data model discovered from the RTFM event log

A2A diagram. Therefore, *Expense* attribute is assigned to separate independent UML classes. Since there is no case of isolated attributes and not-isolated activities in the A2A diagram, rule number two does not apply to the remaining A2A diagram. Subsequently, we check for isolated activities and non-isolated attributes (i.e., rule number 3). Two activities satisfy rule R3. First, *Insert Fine Notification* accesses the *Last Sent* and *Notification Type* with different occurrences. In this case, Function 1 is applied to check whether *Insert Fine Notification* activity is accessing both attributes simultaneously. This happens to be the case in the given log, i.e., in all events where *Insert Fine Notification* accesses the *Last Sent* it also accesses *Notification Type*. Therefore, both attributes are stored in one UML class, where *Last Sent* attribute is marked as optional (based on Function 1). The same holds for *paymentAmount* and *totalPaymentAmount* attributes. Both are simultaneously accessed by the *Payment* activity. Therefore, they are grouped in the same data model class.

Lastly, based on rule R4 we check for the non-isolated activities and non-isolated attributes in the derived islands. By applying this rule the A2A diagram is decomposed into four fragments (see Figure 29, rule 4). Considering the relations between the activities and attributes in each island, rule R1 can be applied to the first three fragments, while in the last one rule R3 can be applied. In this case, the user's choice is to group the attributes in a single class. As the most frequent class, *Fine* is assigned as a root class. The discovered classes are named based on the activity they were generated from. After the associations between classes are constructed, the multiplicities are set based on the occurrences from the A2A diagram. The resulting RTFM data model is depicted in Figure 30.

4.7 APPLICATION SCENARIOS

This section briefly summarizes the application of the data model discovery method in other configuration settings. In [79] by using the method presented in this chapter we discovered the data model from an unlabeled event log that lack the notion of the case identifier. The discovered model is used as a structure to correlate those events (i.e., pertaining to the unlabeled event log) into the event log cases (see Figure 31, AS1 annotation). While in [10], we applied the method presented in this chapter to discover the data model from an event log (i.e., beforehand extracted from the sensor data) and used the discovered data model for code generation in order to automate the construction of a process-aware digital twin (see Figure 31, AS2 annotation). An overview of both scenarios is depicted in Figure 31, while a detailed explanation of them is provided below.

4.7.1 Towards Case Notion Identification from Unlabeled Event Logs

The approach presented in this thesis for discovering the data model from an event log is not bound to the event log cases. Therefore, it neglects the case identifier of the event log and it can be easily applied to an event log that misses the case identifier - called unlabeled event log [14]. Within an organization, these types of event logs are generated from the information systems that do not record events in a process-aware manner or by humans involved in manually driven tasks [4]. However, it is interesting to capture the behavior information related to the process hidden in this type of data. To discover the case identifier and use this data as input for any process mining technique, it is necessary first to correlate them with the process instance notion based on domain knowledge or other techniques like payload data [15].

Several approaches in the literature address the challenge of correlating the unlabeled events into the event log cases. Bayomie et al. [13, 14] discuss the case identifier derivation notion based on decision tree learning approach. While Burratin et al. [24] considers available optional attributes pertained to an event log. In contrast, Ferreira et al. [54] propose an iterative expectation-maximization procedure based on Markov chain. However, most of the approaches require additional information (e.g., process models, heuristic data describing the behavior of the process) and most of them fail to discover the cyclic process models.

To overcome these limitations in [79] we propose a method, which discovers a case notion from an unlabeled event log. Indeed, in the first step (see Figure 31) the data model is discovered (i.e., based on the approach introduced in this chapter) from an unlabeled event log, which is used in the next step to correlate the events pertaining to the unlabeled event log into the event log cases. Based on the data model classes, the unlabeled event log is divided into the event log chunks

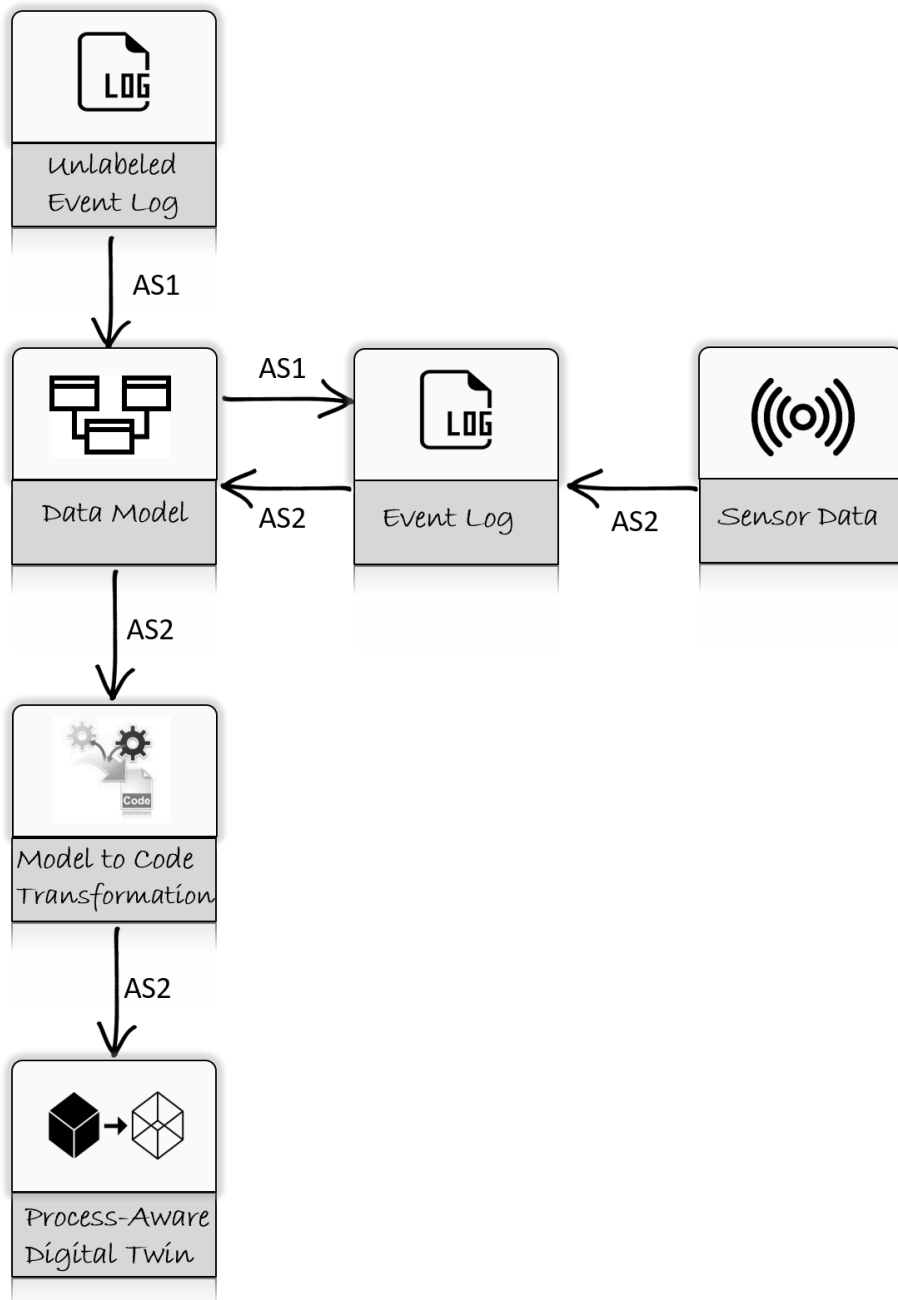


Figure 31: Two data model application scenarios. In the first scenario (annotated with AS1) the data model is discovered from an unlabeled event log and used to correlate such events with the event log cases. In the second scenario (annotated with AS2) the data model discovered from an event log (extracted from the sensor data) is used to automate the process-aware digital twin generation through the model-to-code transformation approach

and the number of chunks is equal to the number of data model classes (one data model class defines one event log chunk). In the next step, the events pertaining to the unlabeled event log are assigned to the data model classes by considering the access relations handled between the unlabeled event log attributes and activities. In consequence, the transition probability [54] is computed, which identifies class instance candidates with the highest transition probability. In addition, the attribute value similarity is computed, which takes as input a set of available class instances and an unlabeled event and computes the degree of affiliation between them. Based on the similarity score and transition probability, the unlabeled events are correlated with the event log cases, resulting in an event log.

The approach is fully implemented² and its feasibility is proven based on two real-life unlabeled event logs (MIMIC [71] and Road Traffic Fine Management [87] event log).

4.7.2 *Towards Process Aware Digital Twin Generation from the Sensor Data*

Another application scenario of the data model discovery method is presented in [10]. The event log is extracted from the sensor data and after applying the method presented in this chapter, the data model is discovered from that log (see Figure 31, UC2 annotation). At the same time, based on a process mining discovery technique, a process model is discovered from the same event log. The discovered data model is used to automate the construction of the digital twins via the model-to-code generation technique. At the same time, the discovered process model is used to enrich the generated digital twin with the behavior information.

Creating a digital twin³ requires the existence of the observable elements in the physical world that can be monitored and controlled. The creation process of a digital twin is challenging [19] due to the system's complexity and the need for collecting information from different disciplines (e.g., domain knowledge (civil engineering [68], healthcare [80]), IT systems and databases, business processes or people knowledgeable of the software engineering of digital twins). Hence, it is important to simplify this process by considering already existing data, called event logs. Using such data during the engineering process of the digital twin enables the application of a low-code development approach, which aims to reduce the hand-written code and shift more responsibilities to the domain experts.

Current research in the engineering process of the digital twin is moving towards the automation process. Therefore, in [10] we propose an approach that aims to automate the engineering of Process-Aware

² the approach is implemented in the scope of Tom Lichtenstein's master thesis and can be found in the GitHub repository: <https://github.com/t-lichtenstein/attribute-driven-case-notion-discovery>

³ The definition of digital twin can be found in [10]

Digital Twin Cockpits (PADTCs). A PADTC is a digital twin's user interaction part and provides a Graphical User Interface (GUI) for visualizing the data. In addition, it can handle the business process of the physical objects and their context. To ensure such processes, one option is to discover them from an event log after applying process mining discovery techniques. While, the extraction of such event logs is usually performed using information systems as a source, which in the case of digital twin represents the software systems accompanying the physical object. Another option to extract such log is sensor data [112, 135], which are defined as a rich source of information related to the physical object [67]. An event log extracted from the sensor data can provide insights regarding the activities of people, machines, and how they behave in a specific environment.

The pipeline of engineering a digital twin in [10] from the sensor data is defined as follows (main steps illustrated in Figure 24): first, the sensor are data gathered and based on this data, the event log is extracted. The method presented in this chapter is applied to discover a data model from this event log. The discover model is used to scratch the data structure of the PADTC, capable of storing and manipulating the data. Based on this data structure, low-code development approaches are applied to automate the generation process of the PADTC and reduce the amount of the hand-written.

4.8 SUMMARY

In this chapter, we have presented a two-step semi-automatic method, which discovers a UML data model from an event log that is purposely designed for process mining. The proposed method is useful for discovering a data model that complements and increases the understandability of the discovered process model. The data model contains classes with their attributes, representing the main entities involved in the process model, and the associations between classes, representing the relationships between those entities. To achieve this, we consider the relations between the activities and attributes pertaining to an event log and represent them via an activity-attribute relationship diagram (A2A diagram), which is an interim artifact of our approach.

We argue that the discovered data model provides additional insights (e.g., to John, if we consider our story) regarding the context and the domain of the event log under consideration. In addition, the data model provides complementary information about the entities that are subject to and can not be captured by the discovered process model.

In addition, two application scenarios are presented to show the applicability of the proposed method. In the first scenario, a data model is discovered from an unlabeled event log and used as the main structure to correlate the unlabeled events with the event log cases. The second scenario aims to use the discovered data model (discovered from an

event log, which is upfront extracted from the sensor data) to automate the generation process of a digital twin via low code development methods.

The proposed method is implemented and the feasibility is proven based on real-life event log.

DISCOVERING DATA OBJECT LIFECYCLES FROM EVENT LOGS

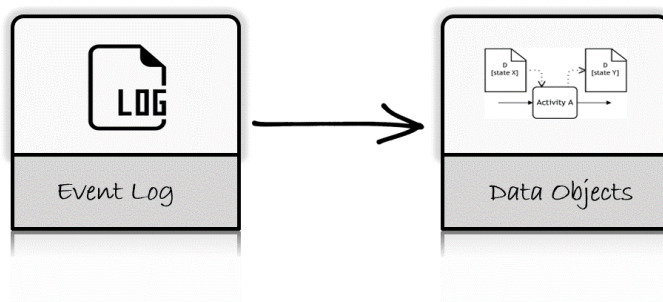


Figure 32: Discovering data objects from event log

This chapter provides a method to discover the data objects and their lifecycle by considering only an event log as input. Considering our story, John can use the discovered information to enhance the process model with a data-flow perspective revealing how the process activities manipulate the data during the execution.

This chapter is mainly based on the previous work presented in [9].

5.1 MOTIVATION

Process discovery is an important area in the field of process mining. It enables the discovering of a process model from an event log [58, 76, 116]. However, most of the discovery algorithms focus on the process control-flow, giving little attention to the data-flow perspective [37, 63], i.e., the data associated with events. As a result, the discovered process models lack information about data dependencies, which data is read or written by process activities, or how many process instances require a certain data object to be manipulated. Manually enriching the discovered process models with such dependencies by the process experts requires domain knowledge, is error-prone and not scalable.

We argue that the data-flow perspective is crucial to understanding the actual execution of processes and, in consequence, the discovered process model. Indeed, a holistic view that combines control and data-flows can support process experts in making decisions, e.g., decisions related to the extracted process execution data [44] or reasoning about data dependencies [28, 94] and their consistency [25, 29, 117]. Besides that, knowing which activity modifies which data objects or which activity requires a certain data object to be in a specific state can help the process experts with post-discovery analyses, for example, focusing on a specific path, pointing out the deviations and reason about root causes.

Research on data-aware process discovery introduces approaches that discover the data-flow perspective from event logs to, for example, identify rules that explain why a certain process path is executed [37, 111] or to discover infrequent paths [90]. However, these approaches mainly focus on deriving data-based conditions that affect the control-flow rather than discovering and enriching the final process model with data-flow information.

In this thesis, we propose a method that leverages existing process discovery algorithms to discover not only the process model but also the data objects and their lifecycles. The approach considers only an event log as a single source of information. To achieve this, we analyze how the attribute values change within each case reflecting how the business process activities modify data on the model level.

We argue that the business process can be understood much better when the process expert is equipped with the discovered process model and the data object lifecycles attached to the process. The feasibility of the proposed approach is evaluated with two real-life event logs: Road Traffic Fine Management and Hospital Billing.

5.2 SCENARIO

This section introduces a small scenario inspired by a real-world running process in a healthcare system. We further use this example to

help explain the discovery of the data objects and the process model enhancement.

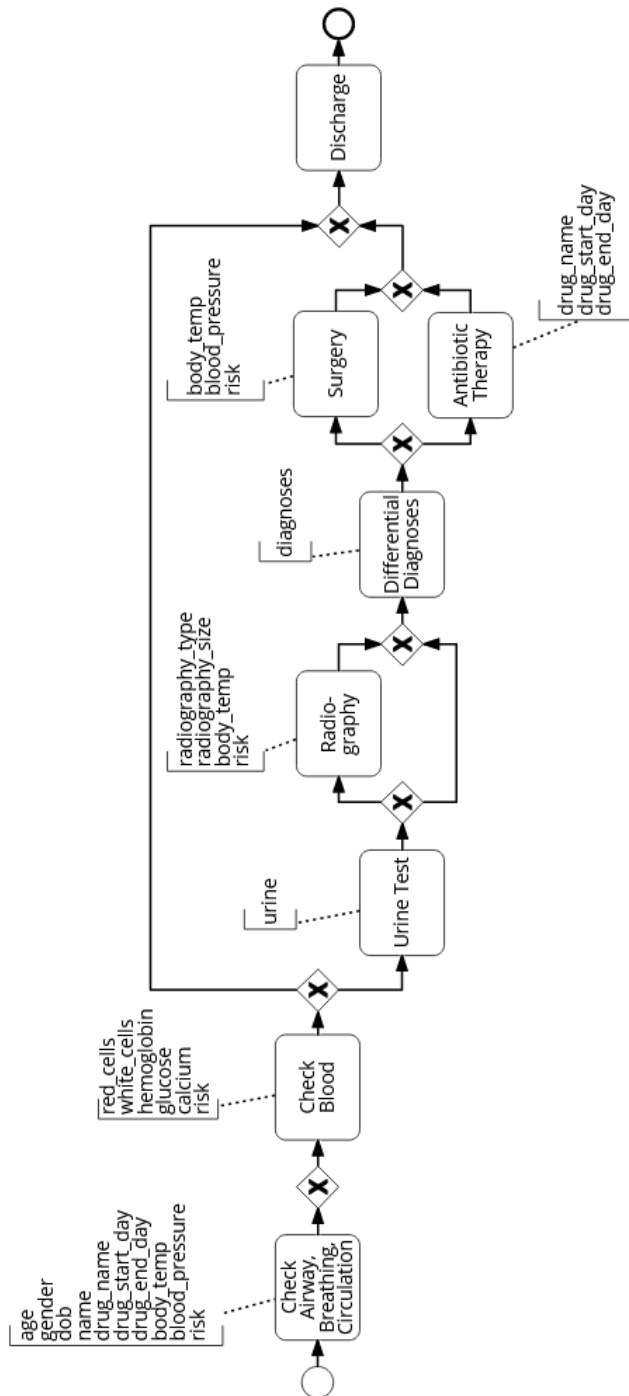


Figure 33: Process model excerpt represented as BPMN inspired by the process management system within a hospital

A simplified process model represented as a BPMN model is illustrated in Figure 33. For this example, the data attributes modified by the process activities are listed as text annotation. The primary care

provider starts the physical examination with the basic check-up called Airway, Breathing, Circulation (ABC). Examples of the collected information within this step are the patient's name, age, gender, day of birth (dob) and information related to the current treatment in case the patient is under another treatment process. Afterward, the examination related to the blood test (Check Blood activity) takes place, which collects information regarding the red and white cells, hemoglobin, glucose, calcium and risk. Based on the blood test findings, the primary care provider determines whether it is necessary to diagnose the patient further or discharge him. If the blood results turn out to be not good, then further laboratory (Urine Test) or imaging diagnoses (Radiography) take place. Then, based on blood analysis results, urine tests or imaging diagnoses, the primary care provider needs to determine the diagnosis (Differential Diagnoses). In the case of uncomplicated disease, the patient can be treated with an antibiotic (Antibiotic Therapy) and information related to the treatment is gathered. Otherwise, the surgery activity (Surgery) takes place. The process ends with the discharge (Discharge), which means that the patient no longer needs to receive inpatient care.

During the execution, the process creates and modifies data attributes. For example, the event attribute `body_temp` is modified by the first activity (Airway, Breathing, Circulation) and the Surgery activity. If we reflect this information in the process model, the body temperature would be part of the data object, connected to the activities mentioned above through data associations and showing a particular state based on the process activity modifying it. However, on the one hand, by just looking at the process model it is not easy to understand how many process instances require access to a particular data object, for example, in how many instances the Surgery activity writes the data object containing `body_temp` attribute. On the other hand, by just looking at the corresponding input event log it is not trivial to understand how the event log attributes can be grouped into data objects and how these data objects evolve during the business process execution.

We reason that the information provided by an event log is valuable to derive the data object behavior. To this end, we present a method that aims to close this gap by discovering the data objects and their lifecycles directly from an event log. Subsequently, the discovered data objects are used to enrich and provide a holistic view of the process model, which is upfront discovered from the same event log. In addition, the process model is complemented with historical information about the number of activity/process instances accessing a particular data object.

5.3 ENHANCING THE PROCESS MODEL WITH DATA OBJECT LIFECYCLES

The discovered process model from an event log does not contain explicit information on how the process activities manipulate the data during the process execution. This chapter presents a method that captures such information from an event log to later enhance the discovered process model. More specifically, we track how the event log attribute values change within a case. The main steps of the proposed method are illustrated in Figure 34. In the first step, we track the behavior of each event log attribute by extracting first an event log called the Attribute-Access event log. For each Attribute-Access event log, the behavior is discovered based on the traditional process mining discovery algorithm (i.e., Alpha Miner [96]). The Attribute-Change matrix is created in the next step, which stores the information regarding the number of times a certain activity accesses an event log attribute. The Attribute-Access behavior, together with the Attribute-Change matrix, are used as input to discover the data objects' lifecycle. In the last step, the discovered data objects are mapped to the business process activities revealing information on the data-flow. A detailed explanation of each step is provided below.

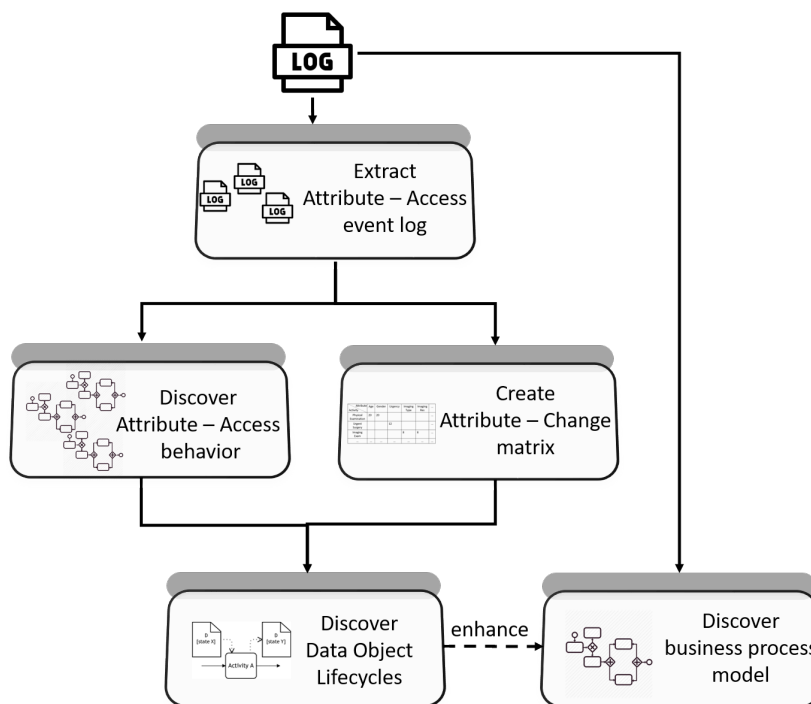


Figure 34: An overview of the approach used to discover the data object lifecycles and enhance the discovered process model with such information

5.3.1 *Attribute-Access Event-Log Creation*

In this section we present how the event log data is transformed into another representation as an intermediate step to discover the data objects and their lifecycles.

- **Attribute-Access Event Log Extraction** We consider an event log as a single source of information for our approach. The association between the activities and the event log attributes is explicitly defined based on the attribute values. Within the same event log case, the attribute value can be invariant or modified by another upcoming activity. Therefore, to understand the behavior of data attributes within an event log, we first split the event log into event log chunks, each pertaining data of one event log attribute. Activities (i.e., events) that do not write or modify the values of any attributes are left out as they do not play a role in determining the behavior of the chosen attribute. Hence, the number of event log chunks will be the same as the number of event log attributes that are modified by at least one activity. All other activities, together with the case identifier, timestamp, and the corresponding attribute values, are kept in such an event log hereinafter called Attribute-Access event log (AAEL).
- **Attribute-Access Behavior Discovery** After having obtained the AAEL for each attribute pertaining to the original event log, we apply a traditional process mining discovery algorithm, such as Inductive Visual Miner [76], to discover the Attribute-Access behavior. Our language of choice to represent the discovered process model is BPMN, as it allows us to discover the complex behavior (i.e., exclusive and parallel gateways) - a feature that will turn useful when representing the data objects in the process model (see Section 5.3.4). A design alternative would be to use the Directly-Follows Graph (DFG) [126] (since it captures how many events or cases have write access to a specific attribute) instead of BPMN, but our goal is to capture the generalized behavior of the data attributes, including exclusive and parallel behavior.

If the discovered process models contain activities with very low frequency (i.e., activities that are executed a low number of times across cases) compared with other activities, a threshold might be introduced to filter them out.

5.3.2 *Attribute Change Matrix Creation*

The BPMN process models discovered in the previous step capture the Attribute-Access behavior showing *how* process activities are writing or modifying a specific event log attribute (i.e., (over)write a particular attribute). However, they do not provide any information regarding the number of cases in which a certain activity accesses the attribute.

Indeed, considering only the Attribute-Access behavior to group the event log attributes into data objects is necessary but not sufficient as two attributes might have the same access behavior, but the number of discovered cases for each attribute can be different. To additionally capture this type of information, we construct an *Attribute-Change matrix*. This matrix is used together with the Attribute-Access behavior to group the event log attributes into data objects (see Figure 35).

The method grouping the event log attributes into the data model classes presented in the previous chapter (Chapter 4) relies only on the access relation between the event log attributes and activities, which is not bound to the event log cases. In addition, the order of activities within the case is not considered. In contrast, the method presented in this chapter strongly relies on the order of activities. It is worth noting that the data objects and data model classes do not necessarily have a 1-to-1 correspondence.

Attribute Activity	age	gender	body_temp	radiography_type	radiography_size	risk	body_temp X risk	...
Check Airway, Breathing, Circulation	20	20	20					...
Surgery			15					...
Urine Test								...
Radiography			12	8	8	12	X	...
Differential Diagnoses			19			19	X	...
...

Figure 35: An example of Attribute-Change matrix based on the running example. The numbers in the matrix indicates how many times each activity (over)writes a specific event log attribute

The matrix is structured as a two-dimensional table where rows represent all activities and columns represent all attributes pertaining to an event log. The number of columns in the matrix is the same as the number of AAELs. Each row in the matrix is populated with the frequencies that represent how often the respective activity accesses the attribute (i.e., the number of times a certain activity writes/modifies an attribute in each AAEL).

Considering the scenario illustrated in Figure 33, Section 5.2, an excerpt of the corresponding Attribute-Change matrix is illustrated in Figure 35. The rows represent some of the activities, while the columns represent some of the data attributes modified by the corresponding process model activities. As shown in Figure 35, the *Check Airway, Breathing, Circulation* activity is (over)writing the value of the *age* attribute 20 times in the AAEL. The same activity, for the same number of times, is (over)writing the *gender* attribute. This indicates that these two attributes might belong to the same data object. However, we cannot guarantee that this is true without looking at their individual behavior.

This is why the matrix complements the behavior with the ultimate goal of grouping the attributes into data objects.

5.3.3 Discover Data Object Lifecycles

This step aims to aggregate the event log attributes into the data objects by considering the Attribute-Access behavior and Attribute-Change matrix obtained in the previous steps (see Figure 34). The discovery of data objects starts by taking into consideration the case attributes (i.e., their value does not change as the events of the case occur). To identify them in the matrix, we search for all attributes that are written only once per case. It might happen that the attribute value is overwritten by the same activity within the same case and this information is not explicitly captured by the Attribute-Change matrix. Therefore, the Attribute-Access behavior has to be taken into account and checked if the discovered process model activity contains a self-loop. If this is the case, the attribute is not considered as a case attribute because within a case its value is overwritten by the same activity name.

For the identified case attributes, we consider grouping them into one data object if the frequency of the activities that are accessing these attributes is the same. In some cases, it might happen that the frequency is not exactly the same but varies by a few units. For example, one activity is accessing one attribute with frequency 2.743 and another attribute with frequency 2.741. This might happen due to incomplete cases being extracted in the original event log (e.g., if some cases were still open when the database was dumped [39]) or it can be the result of data quality issues [20]. Therefore, whenever we compare the activity frequencies, we suggest considering a reasonable threshold, which can be set by the process expert.

As the last step, we check the event timestamps to ensure that the attribute values are written roughly simultaneously within one process instance. If this is true, then the case attributes under consideration are assigned to the same data object.

For example, if we consider the Attribute-Change matrix illustrated in Figure 35, we can see that the case attributes *age* and *gender* are both accessed by the same activity (i.e., Check Airway, Breathing, Circulation). If we are sure that the Attribute-Access behavior is the same and after checking the timestamps we see that they are also accessed simultaneously by the same activity, then we can group them into one data object.

Once all case attributes are grouped into the respective data objects, the next step is to look at the event attributes (i.e., their value can vary based on the process step (event) being executed). Unlike case attributes, event attributes are accessed by more than one activity or multiple times by the same activity (within the same case). Therefore, besides the absolute frequency, the order under which these attributes

are modified is essential for determining the data objects. To capture such information, we compare the discovered Attribute-Access behaviors with each other and distinguish the following two cases.

- *Same activities, same behavior.*

From the Attribute-Change matrix, we can group the attributes that are accessed by exactly the same activities with the same frequency. Afterward, their access behavior is compared to determine their similarity in terms of activities and control-flow. Based on the similarity result, the Attribute-Change matrix is updated to store such information as follows: an additional column representing the group of attributes with similar behavior is added. Considering the matrix illustrated in Figure 35 (gray column), we can understand that the *bodyTemp* and *risk* attributes have similar behavior. The total number of added columns equals the number of attribute groups with similar behavior.

As the last step, for each group, we check if the same activities are accessing the group attributes simultaneously (i.e., within the same case). If that holds, then most likely, these attributes will belong to the same data object. Otherwise, the attribute groups will be further divided until the above holds. Finally, each group defines one data object, even when the group size is equal to one (the data object is created due to one event attribute).

An example of two Attribute-Access behaviors which involve the same activities but different behavior is illustrated in the first column of Figure 36. The discovered behavior for both attributes contains the same activities (i.e., named *A*, *B*, *C*, *D*) but different behavior. For the first attribute, *A* occurs, and after that, a parallel gateway is discovered, which implies that *B* and *C* will be executed without a particular order, and the process will end with the execution of activity *D*. In contrast, the discovered Attribute-Access behavior for *Attribute 2* implies that after activity *A* is executed, either *B* or *C* will be executed and the process ends with the execution of activity *D*. In this example, *Attribute 1* and *Attribute 2* are assigned to different data objects. If the Attribute-Access behavior for these two attributes would have been the same (i.e., they would have both included either an exclusive or parallel gateway), then these two attributes would have been assigned to the same data object.

- *Similar activities, one common behavior subgraph.*

In this step, we look for attributes accessed by a similar set of activities; they share some common activities but do not have exactly the same Attribute-Access behavior. For example, the first attribute is accessed by five activities, and the second attribute is accessed by three of those five activities and three additional

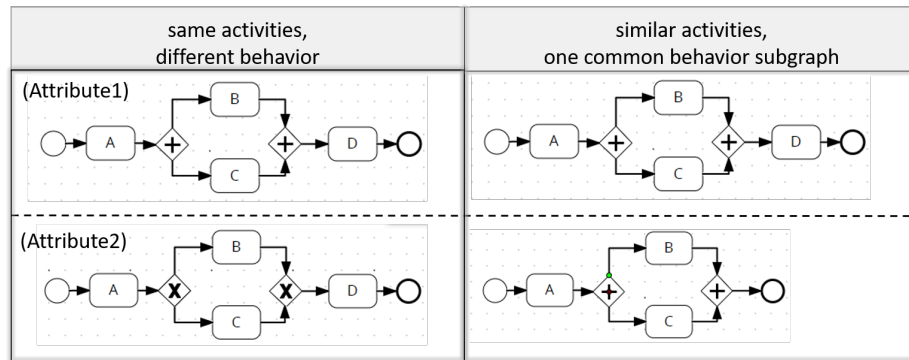


Figure 36: Two cases of discovered Attribute-Access behavior

ones that are not accessing the first attribute. This implies that the behavior of these two attributes cannot be identical.

However, we can still look for a common sub-behavior and argue whether these attributes can be part of the same data object. It is essential to check if we can identify a common subgraph between these two Attribute-Access behaviors. More specifically, we are looking for a subgraph that contains at least two activities and at the most the same number of activities of the smallest behavior. For simple graphs representing Attribute-Access behavior, the comparison can be visually performed. However, if the graphs are complex, approaches based on causal footprints [134] or refined process structure trees [118] may be used.

Once the subgraph is computed, the matrix is updated to capture such a group of attributes to ensure that the subgraph also represents the concurrent access of these attributes in the event logs. This means that all activities of the subgraph are accessing the attributes simultaneously within the same case. The process expert, who is equipped with the common subgraph and the matrix, can decide whether to create a new data object from these groups of attributes. If so, the attribute that is not always accessed from an activity of the common subgraph is marked with * sign in the corresponding data object. That is because these attributes are not always accessed every time the data object is accessed. Otherwise, the process expert can decide not to put these attributes as part of the same data object.

Since the data object might be a composition of several event/case attributes, coming up with a representative data object name is left to the process expert. However, an NLP approach can be used to support the process expert, for example, the one in [77]. In the simplest case, the data object with a single attribute can share the same name with the attribute.

Figure 36 depicts an example of two Attribute-Access behaviors which have similar activities and share one common subgraph

(second column in Figure 36). Activities *A*, *B* and *C* are discovered in both Attribute-Access behaviors. First, the activity *A* occurs. Afterwards, a parallel gateway is discovered, which implies that *B* and *C* will be executed. For the first attribute, the behavior has another activity called *D*. It is important to point out that both the discovered behaviors share a common subgraph, which contain activities *A*, *B*, and *C*. Since activities *A*, *B* and *C* are accessing *Attribute1* and *Attribute2* simultaneously within the same case (i.e., our assumption for this example), then these attributes can be assigned to one data object. Since *Attribute1* is not always accessed from an activity of the common subgraph then it is marked with * sign in the corresponding data object.

Once the data objects are discovered, their lifecycles are identified. The data object lifecycle captures the data object states and the state changes, representing how the data object is being transformed during process execution. We rely on the Attribute-Access behavior of each attribute involved in a single data object to derive its lifecycle.

The data object state constitutes the collective values of its composing attributes after each activity that manipulates any of these values. Specifically, for each activity discovered in the Attribute-Access behavior, a new data object state is created and named after the activity label. This implies that if the data object is created as a result of grouping only case attributes, then its lifecycle consists of a single state.

Otherwise, when the data object is defined as a set of event attributes, excluding the event attributes modified by the same activity within the same case, then any data object lifecycle has at least two states. For each activity involved in the Attribute-Access behavior, a new state is created. For the data objects that involve optional attributes, the number of states is the same as the number of activities of the largest (i.e., in terms of the number of activities) Attribute-Access behavior.

The name of the data object state is based on the activity accessing the data objects (i.e., discovered in the Attribute-Access behavior). If the activity name follows an action-business object pattern [93], we recommend using the past tense form of the action as a data object state name. Once the data object lifecycles are discovered in the next step, we explain how they can be mapped with the business process activities revealing information on the data-flow.

5.3.4 Data Object Assignments in the Process Model

To represent the discovered data objects, first on the process model, it is necessary to discover the latter from the original event log. To achieve this, we are applying a traditional process mining discovery algorithm to the event log.

Once the data objects and the process model are discovered, the next step is to combine these two models together, which implies assigning

each data object to the corresponding process model activity. To achieve this, the data object lifecycle is used as a basic input to derive the data-flow behavior on top of the control-flow. Two cases are considered and explained in more detail below:

1. *Mapping the data objects derived from case attributes*

To establish the connection between the single activity that accesses the data object and the data object itself, a data output edge is used with the activity as its source and the data object as its target. This is because the data object is only written once by this activity and never updated for the rest of the process execution.

2. *Mapping the data objects derived from event attributes*

To map these types of data objects to the process model, we distinguish three behavioral patterns:

- *Sequential access*: If the data object lifecycle manifests a sequential behavioral pattern between pairwise of two states (i.e., one state follows another state but not vice versa), then the data object is output from the first respective activity (the activity from which the state is derived) and it is read from the second respective activity before it is output by it in a new state. That is because the second activity has to read the state of the data object before updating it. For the read access, the data input edge is used in the model, which has the data object as a source and the second activity as a target.
- *Exclusive access*: If the data object lifecycle manifests an exclusive behavioral pattern between pairwise of two states (i.e., if a state is reached, then the other state is never reached and vice-versa), then each activity outputs the data object with the corresponding state. The activities do not read the state of the data object that is output from each of them. It may happen that these activities are manifested in a conflicting behavioral pattern in the process model, e.g., they are in parallel. It is exactly in this case where our approach helps the process experts to understand the process's behavior beyond what is initially captured in the process model and perhaps even update the process model in order to be consistent with the data-flow.
- *Parallel access*: If the data object lifecycle manifests a parallel behavioral pattern between pairwise two states (i.e., one state is reached from another state and vice versa), then we distinguish two cases:
 - *Parallel activities*: If two activities are in parallel in the process model, then this is not desirable because whichever activity updates the data object last overwrites the previous state that is output by the other activity. Therefore,

the first state of the data object is permanently lost. This is an anti-pattern, which is desirable to be avoided.

- *Exclusive to sequential activities*: If, however, there is an exclusive gateway in the process model that leads to two different paths, where one includes a sequence of the activities and the other one contains the opposite sequence, then each path is considered as a sequential behavior pattern, and it is treated like shown above (i.e., parallel activities). For example, a decision is made upfront based on the customer to a) retrieve the payment and then send the shipment or b) send the shipment and then retrieve the payment.
- *Self loops*: If the discovered Attribute-Access behavior contains self-loops associated with activities, then, from a data-flow perspective, they can represent two different situations:
 - The activity is executed multiple times, modifying the data attribute at every execution. This corresponds to a loop in the control-flow of the original process model.
 - The activity is executed once but modifies several instances of the same data attribute, e.g., writing different values in a batch. This corresponds to several rows in the original event log that refers to the same activity, have consequent yet close timestamps and write different values. To represent this behavior in the original discovered process model, a multi-instance data object can be used.

5.3.5 Holistic View Generation

In the discovered process model from the original event log, besides discovering the data objects and their lifecycle, we aim to provide some statistics. More specifically, for each data object state, we capture the *access ratio* in percentage — the number of times a business process activity accesses the output data object over the total number of times that this activity occurs in the original event log. If the data object contains an optional attribute, then this information is provided for the attribute with the highest frequency. This information can be taken from the Attribute-Change matrix. The access percentage is appended to the data output edge connecting the activity with the data object. This information is helpful for the process expert to understand the involvement of activities in the data object manipulation during business process execution and the importance of the data in the overall process execution.

5.4 RELATED WORK

Much research in business process management is conducted considering both perspectives of the process model, the data-flow and control-flow perspective [115, 120]. The relation between these two perspectives is defined as bidirectional. There already exist work that tries to discover the process model based on the predefined object lifecycles [75, 99] and there is also work which discovers the data object lifecycles from the process models [52]. Besides that, Meyer and Weske [94] also discuss on the conceptual level the relation between the process model and object lifecycle. In contrast to our method, none of these these related work address the discovery of the data object lifecycles from the event log.

Rozinat et al. [106] consider the process model, which is upfront discovered by a conventional process mining discovery algorithm and analyze how the data attribute impacts the choices performed in the process model. The main goal of this information is to understand how the data dependency affects the process instance execution. To achieve this, first, the decision points in the process model are identified, then, it is determined whether this decision point is influenced by the case data. Therefore, every decision point is categorized into a classification problem. In order to solve such a classification problem, decision trees are used together with all case attributes pertaining to an event log. The implemented algorithm is called the Decision Miner [106] and is part of a ProM plug-in [133]. The proposed approach presented in [106] has several limitations, as it can not deal with process models that do not fully conform to the event log. In real-life scenarios, most of the discovered process models discard the less frequent behavior and consider them as noise. Hence, the discovered process model does not fully conform to the event log under consideration. Even if the process model is manually designed, it rarely covers all observed behavior. Therefore, to overcome such limitations, Leoni et al. [38] discover a process model from an event log (after applying any traditional process mining discovery algorithm) and align them (i.e, process model and the event log) both together to mitigate the non-conformance effect between them. Once the alignment takes place, the data-flow perspective is discovered, which also covers read and write operations. In the same way as in [106] by looking at each decision point as a classification model, machine learning techniques are used to derive guards. The proposed approach is also implemented as ProM plug-in [136]. Discovering the process model first and then identifying the decision points makes these approaches strongly depend on the process model. In contrast, our approach initially is independent from the process model as it aims to discover the data objects by just considering as input the event log. In addition, rather than providing a meaning to the decision points in the discovered process model, we aim to discover the data object lifecycles,

enhance the overall process model with such information and provide a clearer meaning to the overall process model.

In contrast to the approach presented in [38], in which the control-flow is the most important perspective for identifying deviations, Mannhardt et al. [88] emphasize that balancing the perspectives in a customizable manner (i.e., taking into account the cost function) delivers more meaningful results. Therefore, they do not treat the data-flow as second-class citizen, meaning that the control-flow perspective is discovered first and based on that, other perspectives take place, for example, the data-flow perspective. The authors argue that treating the data-flow perspective as second-class citizen might lead to wrong conformance checking results because, if a data attribute confirms that a certain activity was executed wrong, these approaches will diagnose a deviation in the data-flow instead of control-flow. Therefore, they propose in [88] an algorithm that balances these deviations by considering all perspectives and not prioritizing the control-flow perspective over the others. Similar to our approach, the data object discovery method does not rely on the discovered process model but it is used to enhance such a model. In addition, our approach is independent from the language (e.g., BPMN, Petri nets) used to model the corresponding process. Another similarity compared to our approach is that the number of event log attributes plays an important role in the accuracy of the findings. Different from our approach, the authors consider the attribute values to generate rules on how attributes affect the control-flow (in the same vein as in [16]). In our approach, we focus on how the control-flow affects the collective behavior of the attributes (i.e., the data objects' behavior).

The same authors as in [88] propose the Data-Aware Heuristic Miner algorithm in [89], which uses the data attributes to distinguish process instances with very low frequency from the so called noise (i.e., recording errors) by using classification techniques. The authors argue that some process instance might have low frequency because specified conditions are not often satisfied and if only the control-flow is considered, such instances might be filtered as noise and excluded from the process model. However, they might contain useful information for the process analysts and should not always be set aside as noise. Similar to our approach, the data is leveraged to full potential, i.e., the control and the data-flow are discovered together. Another similarity compared to our approach is that the discovered process model reveals information about the control and data-flows but this information is discovered and presented to the process expert in a different way. In contrast to our approach, the authors are more focused on the actual attribute values in the event log rather than considering the relation between activities and attributes regarding the write/update access.

Fahland [53] focuses on artifact-centric process mining, which is defined as an extension of the traditional process mining [124] because

it is able to analyze event logs with more than one case identifier (i.e., case notion). The approach takes as input an event log or a relational database and outputs a data model of objects together with their relations and a process model called artifact-centric process model. The process model describes the behavior of each data object and the dependencies between them. The approach assumes the association of each event to one data object and from the behavior relations between all events pertaining to one data object, the artifact lifecycle is discovered. Afterwards, the behavior dependencies between different artifacts are defined. In this way, the complete artifact-centric process model provides not only the lifecycle model of each artifact but also the dependencies and their cardinalities between different artifacts. Different from our approach, which requires as input an event log, artifact-centric process discovery requires an event log with multiple case identifiers (multiple case notion event log). In addition, the data model is required as input and assumes that each event is assigned to one data object. This is a strong assumption because event logs, in the classical process mining sense, are unaware of data objects. They, rather, contain information about certain attributes. It is the focus of this chapter to categorize event log attributes into data objects, not just based on static information (see [6]), but on the behavior aspects of the attributes.

5.5 EVALUATION

To evaluate our approach, we are using two real-life event logs, namely the Hospital Billing event log [85] and the Road Traffic Fine Management event log [87].

The Hospital Billing event log contains events regarding the billing of hospital services and includes 100.000 cases (451.359 events) that are processed over three years. It contains 17 attributes and a description of them is provided in Table 3. Following the first step of our approach, the Attribute-Access event logs (AAELs) are generated for all optional attributes except resource and activity lifecycle attributes because they describe respectively who executes the activity and the activity states. This step is implemented in the PM4Py framework [17] and is available in the GitHub¹ repository. In total, there are 17 AAELs extracted (equal to the number of event log attributes), which overall involve 6 activities out of the 18 happening in the original event log.

For each AAEL the Inductive Visual Miner [76] algorithm is applied and the Attribute-Access behavior is discovered. For understandability reasons, the activities that do not access any attribute are removed from the process model without affecting its behavior. Following the next step of our approach (explained in Section 5.3.2), the Attribute-Change matrix is created and depicted in Figure 37.

¹ <https://github.com/DorinaBano/dataObjectDiscovery.git>

Table 3: The description of the Hospital Billing event log attributes [86]

Attribute	Description
actOrg	a flag indicating that the service might be covered by a standard health insurance
actRed	a flag indicating that the service might not be covered by a standard health insurance
blocked	a flag indicating that the billing process is blocked
caseType	the type of the billing package
closeCode	the code used to close the billing package
diagnosis	diagnosis code used in the billing package
flag A, B, C, D	anonymized flags
isCancelled	indicated the cancellation of billing package
isClosed	indicated the the closing of billing package
msgCode	output of the Code Nok activity
msgCount	the number of messages outputs by the Code Nok activity
state	the state of the billing package
version	rule code
speciality	medical code

Then, the data object discovery step takes place (see Figure 38) and there are six data objects discovered in total for the Hospital Billing event log. One data object pertains only case attributes and the rest are created based on the event attributes. There is only one data object which contains a single optional attribute. Specifically, *diagnoses* and *caseType* attributes are assigned to the same data object (see Figure 38) where the *diagnoses* attribute is marked as optional because it is not accessed the same number of time as the *caseType* attribute. In addition, there is a common subgraph between the discovered Attribute-Access behavior of these attributes and the discovered process activities are accessing these two attributes subsequently. As shown in Figure 38, there is also a common subgraph discovered between the data objects containing the *msgType* and *msgCode* attributes and the data objects created as a result of *actRed*, *actOrange* and *flagC* attributes but in the common

Activity \ Attribute	isCancelled	diagnosis	caseType	speciality	blocked	isClosed	flagD	flagB	flagA	closeCode	actRed	actOrange	flagC	msgCount	version	msgType	msgCode	diagnosis x caseType
New	77627	35528	77664	77627	77627	77661	77627	77627	77627									x
Fin		341	71							71619								x
Change Diagn		45451	1467															x
Code ERROR											60	60	60	60	62	60	60	
Code OK											63169	63124	63208	63746	64867			
Code NOK											2992	2988	2994	3058	3330	1546	1548	

Figure 37: Attribute-change matrix created for the Hospital Billing event log (the color figure is available online)

subgraphs activities are not accessing these attributes simultaneously. Therefore, they are stored in two different data objects.

Subsequently, the process model is discovered from the original event log, which is illustrated in Figure 39. The discovered process model reveals that many of the activities can be skipped, i.e., they have an exclusive gateway upfront. The lifecycles of the data objects are discovered and represented in the original process model, as depicted in Figure 39.

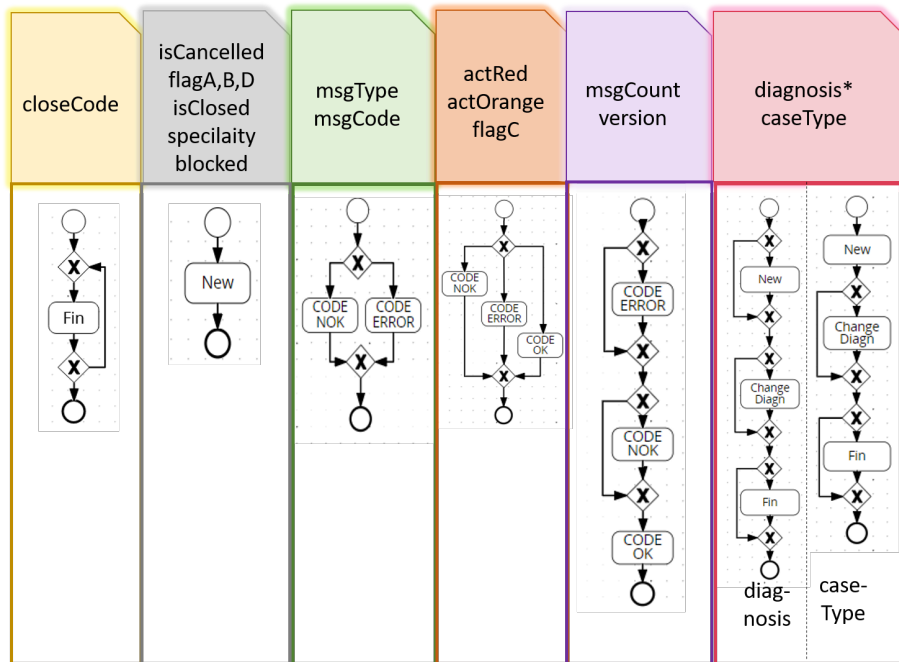


Figure 38: Discovered data objects and their Attribute-Access behavior for the Hospital Billing event log (the color figure is available online)

Each data output edge shows the number of instances that the activity has written the data object over the number of instances the activity occurs, represented in percentage. For example, the data objects discovered from the case attributes are written in 100% of the times when the respective activity (*New*) is executed.

We argue that enriching the process model with data objects and their lifecycles helps the process experts to understand the process execution better. For example, the Hospital Billing process model illustrated in Figure 39 contains a parallel gateway. Focusing on the data object that contains the *msgCount* and *version* attribute, it can be observed that the *Code Error* and the *Code Ok* activities access the data object in parallel to the *Code Nok* activity. However, judging from the data object lifecycle (see Figure 39), these activities should all be in a sequential relation. In this specific case, we investigate this discrepancy between the process model, the data object lifecycle and the original event log. The following is observed:

- there are only 8 cases out of 100.000 in the event log where the *Code Ok* activity is eventually followed by the *Code Error* activity. This is due to the presence of a loop in the discovered process model (not shown in Figure 39 due to the low frequency), where the *Code Error* activity happens a second time. However, the second time the *Code Error* activity does not access the data object again. Hence, the data object lifecycle manifests only a sequential state change.

- there are 13 cases where the *Code Ok* activity is eventually followed by the *Code Nok* activity, but in none of these cases, the data object is overwritten by the latter.

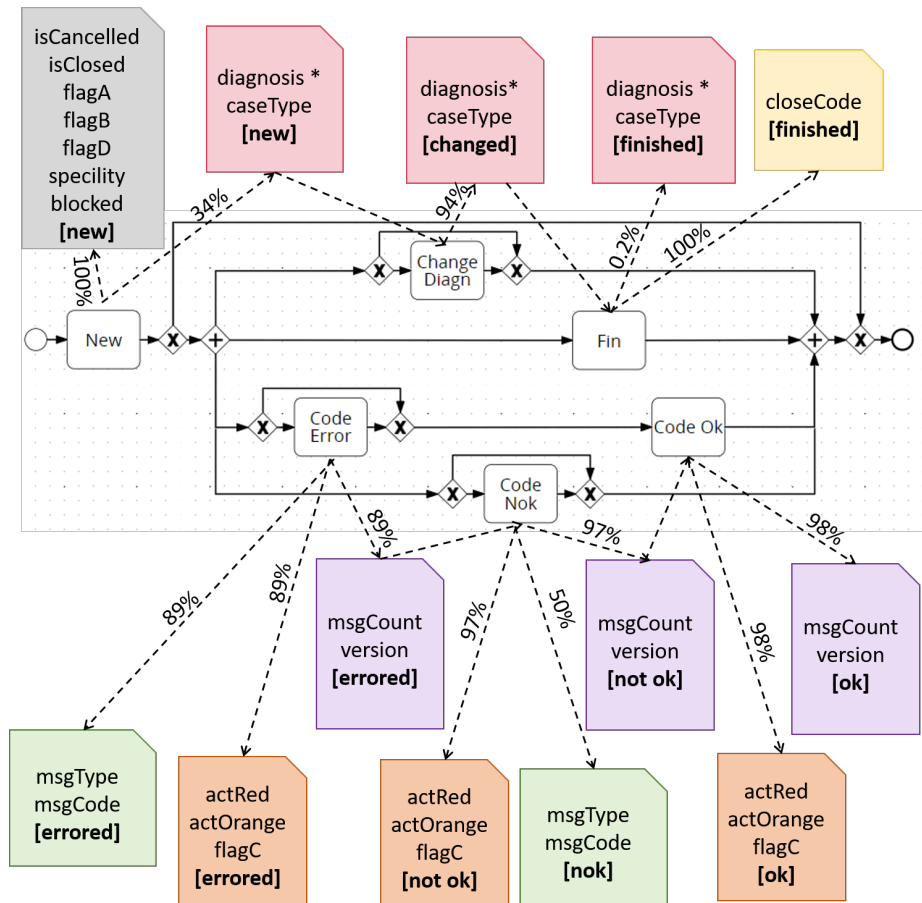


Figure 39: Process model (discovered by the Inductive Visual Miner), data objects and their lifecycles for the Hospital Billing event log (the color figure is available online)

While the RTFM event log involves events regarding the road traffic fines issued by the Italian police, it includes 150.370 cases (561.470 events) that the municipality has processed over thirteen years. The event log contains 11 activities and only 7 of them are accessing (writing or modifying) an event log attribute. The attributes of RTFM event log are described in details in Section 5.5, Table 2. The discovered data objects and their Attribute-Access behavior are illustrated in Figure 40, which are used to enhance the process model discovered from the original event log (illustrated in Figure 41).

Compared to the Hospital Billing event log, the RTFM log contains more static data objects (i.e., their states do not change during process execution). That is why there are no activities that read something from the previous data objects and later change their state. In addition, 5 out of 8 discovered data objects are created as a result of case attributes. An-

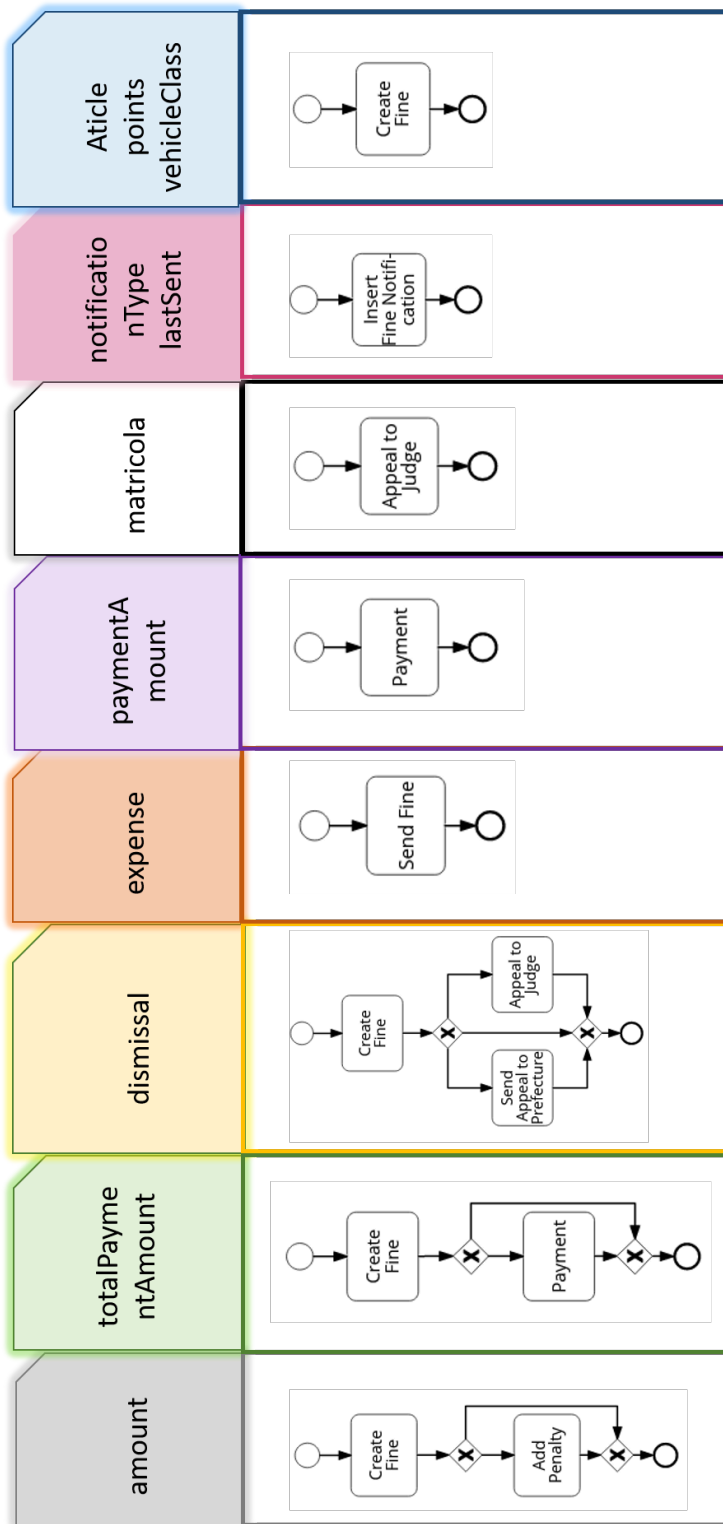


Figure 40: Discovered data objects and their Attribute-Access behavior for the Road Traffic Fine Management event log (the color figure is available online)

other difference compared to the Hospital billing event log is that there are more activities that manifest a 100% writing ratio, which means that always when an event occurs, it writes a value to the event log attribute.

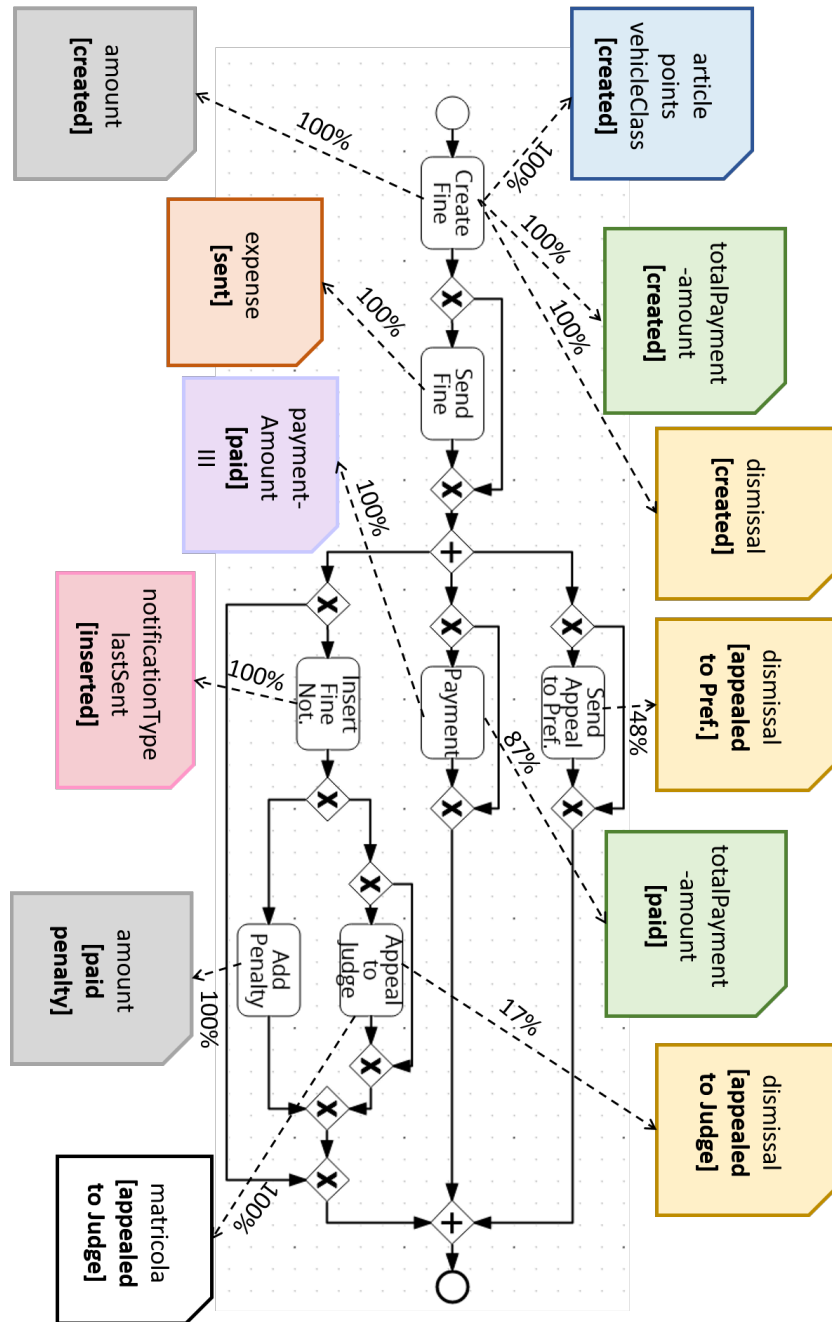


Figure 41: Process model (discovered by the Inductive Visual Miner), data objects and their lifecycles for the Road Traffic Management event log (the color figure is available online)

5.6 SUMMARY

This chapter presents a method to discover the data objects and their lifecycles from an event log that is tailored for process mining. To achieve this, we analyze how the event log attribute values change for each case and group these attributes into data objects, which reflect how the business process activities consume the data during the process execution. The process model is enhanced with data object lifecycles and execution heuristics stemming from the event log.

Enriching the discovered process model with data objects helps the process experts (e.g., to John, considering our story) to understand how data is manipulated through the business process execution. The discrepancies between the process model and the data object lifecycles, together with the access rates, can shed light on how the data-flow matches the control-flow. This is due to the fact that the relation between the business process activities and data objects are clearly defined.

The feasibility of our method is proven based on two real-life event logs, called Hospital Billing and Road Traffic Management event log. Our evaluation shows that the discovered process model can reveal more behavioral information to the process expert when it is enhanced with data objects and their lifecycles. Thus, the enhanced process model is closer to the real-world execution of the process in terms of semantics.

DISCOVERING BUSINESS PROCESS ARCHITECTURES FROM EVENT LOGS

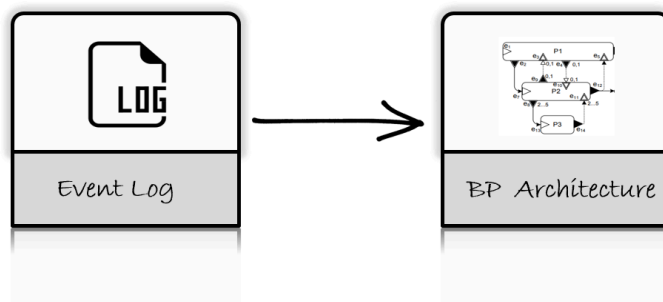


Figure 42: Discovering business process architectures from a set of event logs

This chapter provides a method (see Figure 42) to discover the relations between a set of business process models, which are upfront discovered from their respective event logs. Considering our story, John has applied the method to the event logs that are all extracted from the same organization's database considering a specific business goal.

This chapter is mainly based on the previous work presented in [8].

6.1 MOTIVATION

The business process management lifecycle [139] deals with the design, configuration, enactment and evaluation of business processes in a perpetual repetition in order to accommodate the ever-changing business requirements. An important artifact in this context is a process model repository, which often captures hundreds or even thousands of process models [103]. For example, SAP contains around 600 reference models (i.e., models used as a core to design additional models) [102]. While Suncorp, which is an Australian finance, insurance, and banking corporation holds around 6,000 process models in their process repositories [137]. With the increase in size and complexity of process model repositories, it gets harder to manage them. The area of BPM addressing such a challenge is called Business Process Architecture (BPA) [46]. BPA is an important vehicle for organizing business process models within an organization since they provide a holistic view of the interrelationships between business process models [56].

Typically, BPAs are designed by process experts by identifying the relationships between process models in a given repository (i.e., BPA design is purely based on process models). This step involves manual-driven tasks [74], and in most cases, explorative methods are applied based on the activity labels and domain knowledge [56]. To overcome such limitations and use additional information besides the activity labels and domain knowledge, this thesis provides a method that considers the historical execution data such as an event log.

In the process mining area, event logs are usually extracted from the given organization's database based on a specific case notion that determines one's perspective from which an event log can be extracted or the questions one aims to answer (see Figure 43). It is possible to extract more than one event log from a single database, each pertaining to a different case notion. For each extracted event log, the process model is discovered by applying any process mining discovery algorithm. To define the relations between these processes, we reason that the information stored in the event logs is rich enough to indicate relations between two or more processes. The relations defined between processes purely on the model level might not always reflect the real-world ones that actually occur during business process executions which are captured, in part or fully, in an event log.

Therefore, in this thesis, we propose a method for automatically discovering a business process architecture from an organization's historical data that is captured in several event logs. The method focuses on defining two types of process relations — trigger flow and information flow. Trigger flow represents situations when a business process triggers the instantiation of another business process. In contrast, information flow captures data exchange between business processes. The resulting method can discover such complex process interdependencies

that we needed to design, in addition, an extension of an existing BPA graphical representation [50] to accommodate these relationships properly.

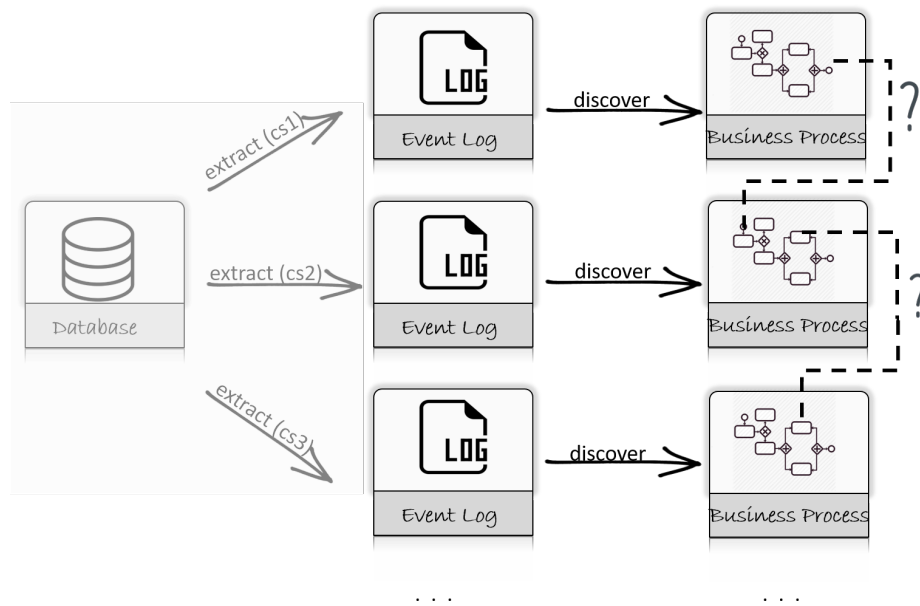


Figure 43: The overall approach that discovers the business process relations (represented by dashed lines) in the form of a BPA by just considering a set of event logs as input. Each event log is beforehand extracted from the organization's database by having in mind a different business goal

6.2 BUSINESS PROCESS ARCHITECTURE AND EVENT LOG META-MODELS

Before describing the approach of deriving a business process architecture from a set of event logs, let us first define the relationships between these artifacts on the meta-model level (see Figure 44). At a conceptual level, *BPA* is defined as a composition of the *Process Models* and *Information/Trigger Flow* models. In its simplest form, a *Process Model* is a composition of *Sequence Flow* models and *Flow Node* models, which can be *Activity Models* or *Event Models* that correspond to some *Events* from the *Event Log* (illustrated on the right-hand side). An *Event Log*, one of the many in an organization's *Event Log Repository*, is a collection of events that are grouped into cases. One *Event Log* corresponds exactly to one *Process Model*. This chapter aims to discover the relationship between a set of event logs at a BPA level of abstraction, defined by the *Trigger* or *Information Flow* relation (illustrated by the *discover* arrow from the *Event Log* self relation to the *Trigger/Information Flow*).

Given a set of event logs derived from a single organization's database, we show how to discover the business process architecture of that or-

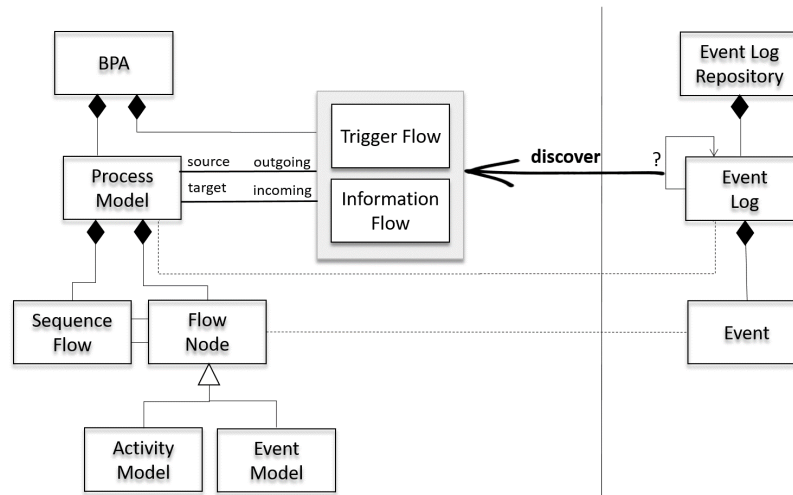


Figure 44: The meta-model for describing the relation between the business process architecture and the event log

ganization. The focus here is on the discovery of the trigger and information flow between processes¹, which implies the need to identify the *throwing* and *catching* event models inside a process model. The throwing event model is considered as *active* because it sends a signal, and it can be an *intermediate* or an *end* event model. In contrast, the *catching* event model is considered as *passive* because it waits for the signal to arrive, and it can be either a *start* or *intermediate* event model (the concepts are illustrated and explained in more details in Section 2.1). The method of how to discover these dependencies between processes based on the data generated during the business process execution in the form of event logs is provided in the upcoming sections.

6.3 EVENT LOG AWARENESS

To discover the trigger and information flow relations between a set of event logs, we exhaustively investigate all unique pairs of event logs in the repository and draw the final BPA. As a prerequisite, for a trigger or information flow to exist between two event logs, at least one event log has to be *aware* of the other. With awareness we mean here that a significant number (decided by the process mining expert) of cases of a given event log contain explicit information about the existence of cases pertaining to the other event log. In short, an event log is aware of another if a significant number of its cases are aware of the other. Formally we have:

¹ Other types of relations can exist between business processes (see Section 2.1), but in this thesis, we are considering only trigger and information flow relations

Definition 6.1.

(Aware Cases) A case c from event log El (i.e., $\exists e \in El \mid \#_{case}(e) = c$) is aware of a case c' from event log El' (i.e., $\exists e \in El' \mid \#_{case}(e) = c'$) if at least one event in case c contains information that refers to the case id of case c' . Let us denote $Caw_{El,El'}$ as the set of all ordered pair of cases (c, c') where c is aware of c' . ◀

Subsequently, we define the event log awareness:

Definition 6.2.

(Aware Event Logs) An event log El is aware of another event log El' if its awareness is not less than a threshold τ , where $0\% \leq \tau \leq 100\%$ is specified by the process mining expert. The awareness of event log El for the event log El' is defined as $\mathcal{Aw}_{El,El'} = \frac{|Caw_{El,El'}|}{|El|}$. ◀

As one may notice, the awareness is not symmetric, in that $Caw_{El,El'}$ may be different from $Caw_{El',El}$. Moreover, a case c from El can be aware of more than a single case from event log El' as shown later in Section 6.7, where a case triggers two cases of the opposite log.

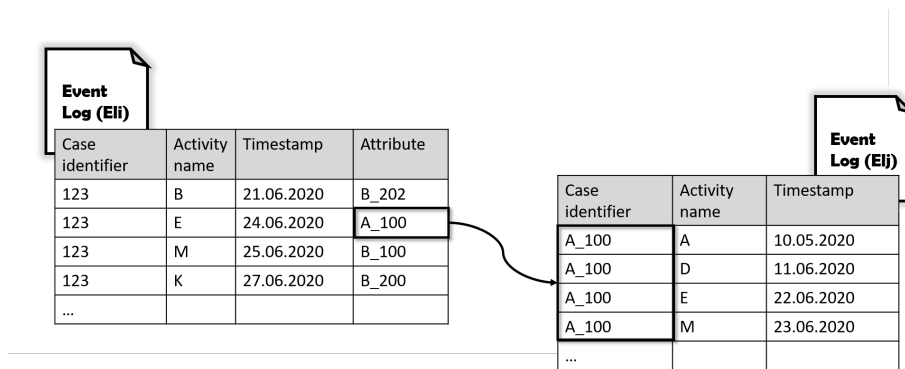


Figure 45: Example of a case from event log El_i being aware of another case pertaining to the event log El_j

An example of case awareness between two event logs El_i and El_j is depicted in Figure 45. Specifically, the case with identifier number 123 in the event log El_i is aware of the opposite case pertained to the event log El_j because its attribute value (i.e., A_{100}) is equal to the case identifier of the event log El_j .

Given two event logs, we have to first check whether one of the event logs is aware of the other. If none of the event logs are aware then there cannot be a trigger or information flow to speak of. Otherwise, we have to analyze all the aware cases of the respective event logs. Specifically, we need to decide on a case basis whether there is a trigger or information flow. Only then we can discover the generalized trigger or information flow at the BPA model abstraction level. The discovery part is very crucial because the trigger or information flow might depend

on a case to case basis. For example, in half of the cases the trigger or information flow source is a throwing *activity A* and in the other half is a throwing *activity B*. To capture this kind of behavior, we provide an extension of the BPA modeling language (see Section 6.5).

Event logs usually do not contain explicit information whether certain events in them are catching or throwing. Therefore, we have to deduce this information via other means. Specifically, in order to detect whether there is an information or trigger flow between two events where at least one case is aware of the other (see Definition 6.1) we have to analyze:

1. the causality relationships between these events
2. and, in case of information flow, the information contained in these events, i.e, the event attributes.

6.4 DISCOVERING THE BUSINESS PROCESS ARCHITECTURE

This section provides a detailed method for deriving the trigger and information flow from a pair of event logs. In addition, the algorithm used to derive such information is presented for each method.

6.4.1 *Trigger Flow*

To discover the trigger flow behavior between two event logs, we have to detect the trigger flow instance for each case, then use process mining discovery techniques to arrive at a general representation at the BPA level. The overall approach for discovering the trigger flow between two event logs is depicted in Figure 46.

For simplicity purposes, every pair of aware cases from the opposite event logs (from hereon referring to cases belonging to two different event logs respectively) are merged into one joined case where the new case identifier is the cross product of the original case ids. Considering the example illustrated in Figure 45, the case identifier for the joined case is equal to $123 \times A_100$. Each event is annotated accordingly in order to specify which event log it originates from. Considering the same example, all events pertaining to the event log El_i are suffixed with a 123 token. The same applies to the event log El_j , in which all events are suffixed with A_100. All the merged cases are put together in a joined event log El_{ij} .

For each pair of opposite cases, where one is aware of the other, we have to detect the source and the target of the trigger flow instance. The target of a trigger flow is always a *start* event. While the source it can be an *end* or *intermediate* event. On a case level, the target of the trigger flow instance must always be the first event of the case. Therefore, the only candidate for the target of the trigger flow instance between the two start events is the one that occurs second.

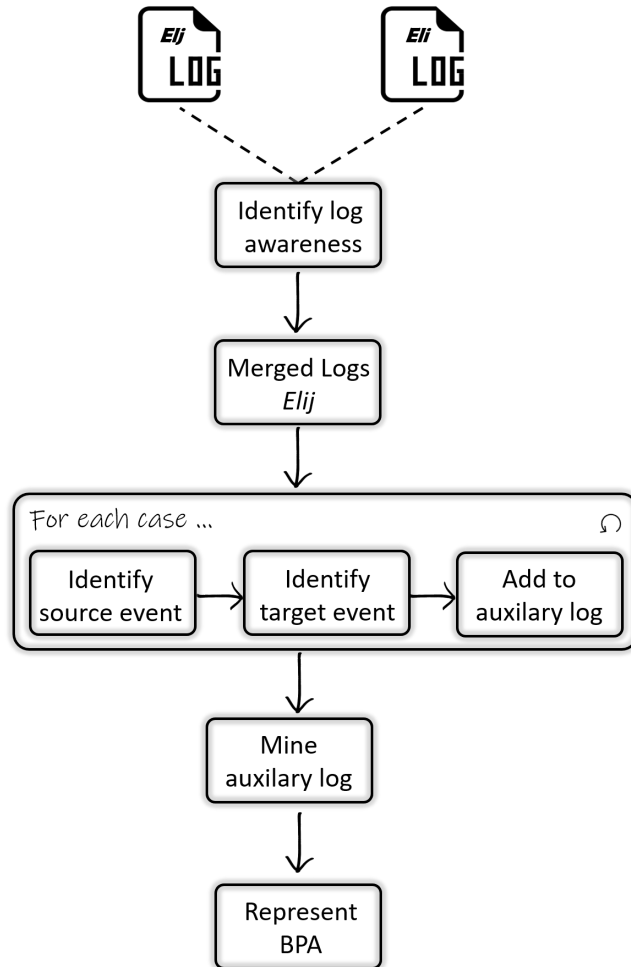


Figure 46: The overall method for discovering the trigger flow patterns between two pairs of event logs

Once the candidate target of the trigger flow instance is fixed, we have to detect its source, which is a throwing event (i.e., it can be an *end* or *intermediate* event) from the opposite event log that happens before the target event. From all the possible events, we choose the latest event as the source of the trigger flow instance among those that refer to the case identifier of the target case. We argue that this is very important because the trigger flow source event is responsible for passing all the important information (at least the *case identifier*) required to start a new instance on the target process model. The exact algorithm for detecting the trigger flow instances for each pair of aware cases is provided in Listing 2.

If a trigger flow is detected between two cases, then its source and target events are added to an auxiliary event log. This event log has as a case identifier the combination of the source and target case id (the same as in event log El_{ij}). Once it is completed with all possible source and target events, it is mined to discover the final behavior of the trigger flow. We argue that this information is very important because

the trigger flow between two business process models in a BPA might not always be static, in that the trigger flow might have more than one source and as well as more than one target. In addition, not always the same source might be related (i.e., send information) to the same target. We show in Section 6.7 that it is very important to capture this behavior on the BPA level to enable comprehensive analyses.

```

input: merged event log Elij
output: auxiliary event log (Elaux)
#i_event and j_event refer to events that originate
  respectively from Eli and Elj
for case in Elij
5  {
  start_event_i = first_i_event(case)
  start_event_j = first_j_event(case)
  if start_event_i.index < start_event_j.index
  target_event = start_event_j
10 else
  continue #jump to the next case

  for event_index = target_event.index - 1 to 0 step -1
  {
15  candidate_event = case.get_event_with_index(event_index)
  if candidate_event is i_event
  {
  source_event = candidate_event
  add source_event to Elaux
  add target_event to Elaux
20  break
  }
  }
}

```

Listing 2: The trigger-flow instance detection algorithm

6.4.2 Information Flow

Similar to the way we discover trigger flows between two event logs, we have to consider on a case basis about the existence of data flowing between events of opposite event logs. The method overview is presented in Figure 47. An information flow instance between two aware cases assumes the existence of a throwing and a catching event from opposite cases that manifest a data-flow, i.e., the information contained in the throwing event is passed to the catching event. In other words, we look for new information appearing in one event that stems from an event of the opposite event log.

Differently from trigger flows where we consider only the *start events* of the processes, here we have to consider all the possible events with few exceptions, e.g., *start events* cannot be throwing and *end events* cannot be catching.

A joint event log $E_{i,j}$ is created in the same manner as described above for the discovering of the trigger flows. What is slightly different is that all the common attributes between two event logs have to be identified first. If two event logs share no common attributes other than the case identifier (which is given since at least one event log is aware of the other), then there is no information flow to consider. It is worth pointing out that the attribute name may not be sufficient to check whether two attributes from different logs have the same domain and value ranges or, in short, refer to a common entity. That is why this step might require a process expert to create a mapping of attributes that are believed to represent the same information.

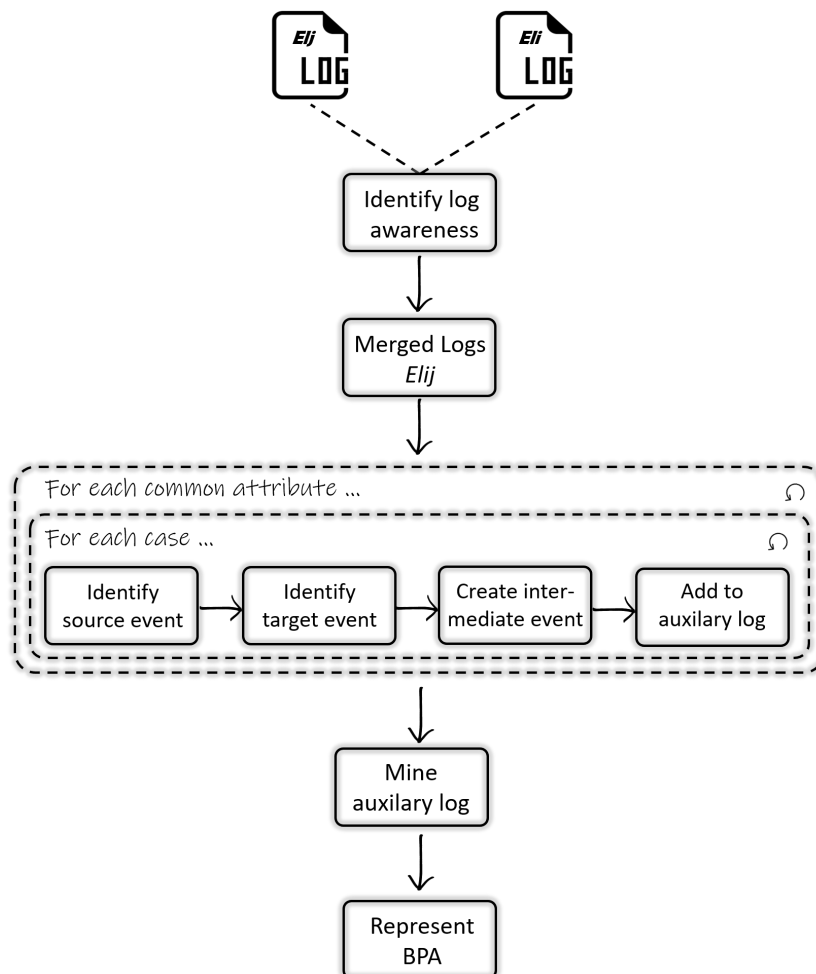


Figure 47: The overall method for discovering the information flow patterns between two pairs of event logs

Once the common attributes are identified, the search space for the data-flowing from one event log to another is reduced just to these attributes. From this point on, we consider one attribute at a time for determining whether there is a information flow that concerns the attribute at hand.

```

1 input: merged event log Elij
  output: auxiliary event log (Elaux)
  #i_event and j_event refer to events that originate
    respectively from Eli and Elj
  for case in Elij
  {
6   checked_event = first_i_event(case)
    if checked_event == Null
      continue #jump to the next case
    last_found_index = -1
    for event in case
11   {
      if event is i_event
      {
        if event.attribute.value <> checked_event.attribute.
          value
16       {
          target_index = event.index
          source_found = False
          for s=last_found_index to target_index-1
          {
21             if event_with_index(s) is j_event and
                event_with_index(s).attribute.value == event.
                  attribute.value
26             {
                source_event = event_with_index(s)
                target_event = event
                source_found = True
                last_found_index = s
                break
            }
          }
        if source_found
31        {
          intermediate_event = new event
          intermediate_event.case = source_event.case
          intermediate_event.name = source_event.name+
            target_event.name+attribute.name
          intermediate_event.timestamp = mean_value(source.
            event.timestamp, target.event.timestamp)
          add source_event to Elaux
36          add intermediate_event to Elaux
          add target_event to Elaux
        }
      }
      checked_event=event
41   }
  }
}

```

Listing 3: Information-flow instance discovery algorithm

Considering a single common attribute between two event logs, we cycle through all the pair of cases where one is aware of the other. For each pair of cases we consider only the events that access the attribute. Anytime a new attribute value appears in an event, i.e., the attribute

value is not present in all preceding events from the same case, we look at the opposing case to determine whether this value appears in any of the events that happened prior to the original one. If yes, we take the latest event where that occurs as the source of the information flow instance and the event with the new value as its target.

The source and the target event of the information instance is added to an auxiliary event log similar to the one used for the trigger flow. However, here we add one more event (between source and target) that is labeled as the composition of the source and target label plus the attribute at hand. As a timestamp, it takes the median time of the original events. This step is added because:

- a) there might exist more than one information flow between two cases related to the same attribute and
- b) information flows for different attributes need to be distinguishable from each other.

The algorithm for discovering the information flow is illustrated in Listing 3.

Once the auxiliary event log is completed, we apply any process mining discovery technique to discover the behavior of the information flow. The output is a behavioral model, usually represented as Petri net, that captures the behavior of the information flow for one common attribute. This model is then translated into a BPA. If two or more information flows overlap, i.e., they have the same source and target, they can be represented as one information flow that represent the passing of composed information.

Since the behavior of the trigger and information flow is a new concept, we extend the BPA representation from [50] with some new lightweight notions to capture exclusive or parallel patterns between trigger and information flows in the section below.

6.5 THE EXTENSION OF THE BPA'S GRAPHICAL REPRESENTATION

The state-of-the-art research on BPA does not suffice in capturing behavior of the trigger and information flow that we observed by analyzing a set of real-life event logs (see Section 6.7). In this section we address this gap by introducing an extension of the graphical representation of BPA (presented in Figure 48).

For representing the complex behavior of the trigger and information flow, we borrow the concept of gateways from BPMN 2.0 [62], specifically the *XOR* and *AND* gateway. Figure 48 shows how these gateways are used given a set of specific scenarios.

Figure 48 a) represents the case when there exists more than one start event in an event log, i.e., that cases do not always start with the same event instance. In contrast, Figure 48 b) represent the case where there is more than one event in the first event log that triggers the process

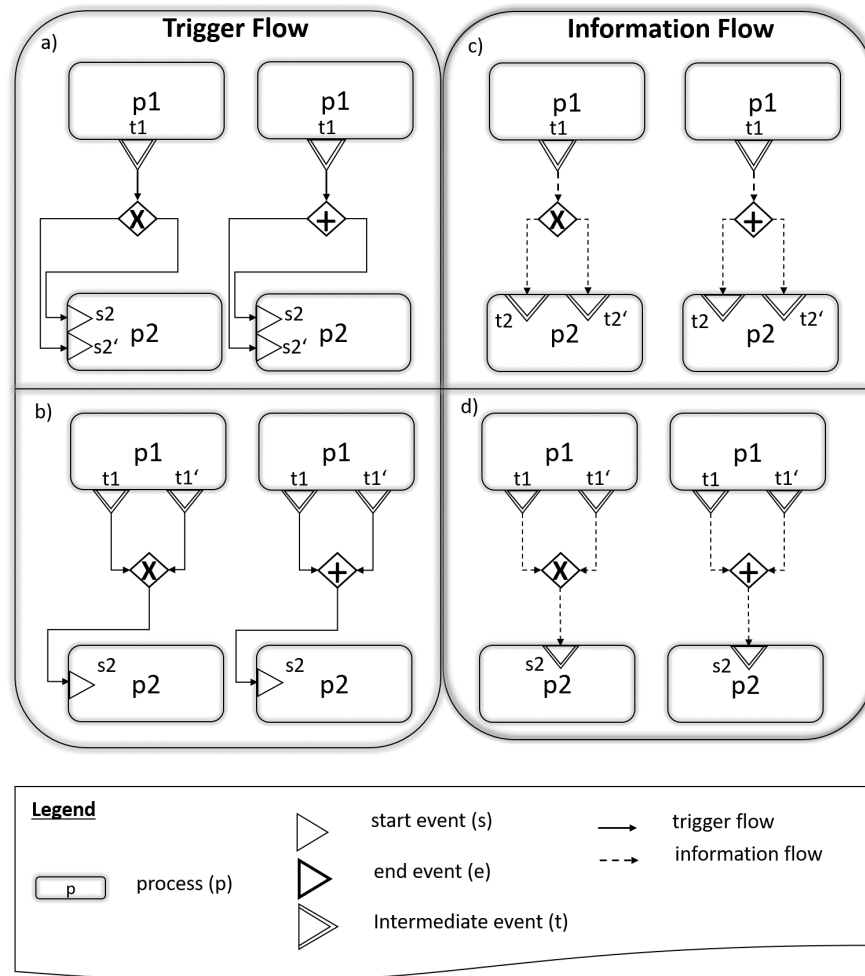


Figure 48: Complex behaviour of trigger and information flow

instantiation of the second event log. The same can be observed for the information flow in Figure 48 c) and d).

Despite the gateways being represented in the space between the processes, they represent the behavior of either the source events (for join gateways) or target events (for split gateways) of the respective trigger/information flow. For example, the difference between the trigger flow’s XOR and AND splits in Figure 48 a) consists in that, in the former, s_2 and s_2' start events are in an exclusive relation between each other, while in the latter, they are in parallel relation. Obviously, this information is found in the event log of process p_2 . This information is required in addition to the auxiliary logs to determine whether we use an XOR or AND gateway.

As it is illustrated in Figure 48, the orchestration of these gateways happens inside the processes. However, there might be more complex cases were, for example, a process is triggered only after two or more processes have been executed or, after a process is executed, there is a decision on which following process to trigger. Such cases are not cov-

ered in this thesis and require analyzing of more than two processes at once across the process repository. The gateways in those cases would lay outside the processes, similar to a process choreography setting, where enforcing gateways becomes much more complicated [97].

Depending on the process discovery approach applied to the auxiliary event log a behavior model is obtained. The majority of algorithms output a Petri Net model. Known techniques, like in [81], can be used to derive the process gateways from a Petri Net model.

6.6 RELATED WORK

Business process architecture is subject to extensive research work. The existing literature focuses mainly on organizing the process repositories within an organization by involving the domain experts. They are mostly used to ease the process classification and set up the hierarchical relations between processes.

Dijkman et al. [45] provide an overview of the prevailing approaches related to the business process architecture design process. The main goal was to investigate the state-of-the-art of the existing business process architecture design approaches and estimate their usefulness in practice. Through a literature study, they have proposed five classes of approaches: goal-based, action-based, object-based, reference model-based and function-based. For each class, first, the structure was defined (e.g., goal-based structure) and afterward, the business process architecture was subsequently designed based on that structure. Different types of goals were identified (e.g., realization, influence [73]), and focusing on a different goal, leads to a different goal structure, which potentially leads to a different business process architecture [5]. Approaches related to *action-based* defines first an action structure consisting of business process activities and their relations (i.e., so-called business action). Unlike business processes, the business action lies therein that business action assumes that all humans and, consequently, business actions follow a certain pattern and phases. A pattern helps to identify the (sub)processes, while the phases determine when a certain (sub)process ends and another one begins, considering the transitions between phases. The main relations identified within the action-based approaches are [70]: decomposition, trigger flow and phasing relation.

In contrast, in the *object-based* approach, a business process object is designed first, for example, as a UML class diagram and then the business process architecture is derived by looking at the business objects and their relations in the class diagram. The types of relations identified within this class are: decomposition, state transition and generalization. To speed up the process of designing a business process architecture, one could start from an existing business process architecture (called reference model) and adapt it to design a new one. Decomposition and generalization are two types of relations considered within this

type of approach. Considering the *function-based* approach, a function hierarchy is designed first, representing the decomposition of business functions into more detailed business functions. The functions hierarchy and business process architecture can be combined in two ways: 1) the business process architecture is organized based on the functions hierarchy and 2) the structure hierarchies are created by combining process and functions through decomposition relations. Each design approach was evaluated in terms of use and usefulness by considering 39 practitioners active in the BPM area.

Eid-Sabbagh and Weske in [49, 50] consider an action-based approach and focus on the formal conceptualization of the BPA. They introduce a set of patterns to analyze the interactions among several business processes within an organization, including anti-patterns (i.e., dead events, deadlocks) used to represent erroneous relations between business processes. Our work extends on their work for discovering the trigger and information flow patterns not by analyzing the process models but rather by discovering them directly from the event logs.

Conforti et al. [30, 31] address the composition relation between processes by providing an approach that discovers BPMN models, which contains sub-processes together with their interrupting and non-interrupting events from an event log. Afterward, traditional process mining techniques are applied to the event data belonging to both the sub-process and the original process. Similar to our approach, authors consider event log attributes to identify the events that are more likely to be part of the sub-process. Different from their solution, we consider a set of event logs to determine the trigger and information flow relations and represent them on a higher level like BPA.

Dumas et al. [47] propose an approach to support the detection of the clones of big business process repositories. Each process model within one repository is indexed to identify duplicate fragments, which can later be refactored into shared sub-processes. Other approaches dealing with the clone detection are recently researched in the process mining area [12, 51, 118]. Unlike our approach, the refactoring is based only on the activity labels since the process model is considered as a single source of information.

Hierarchical process models are discovered from the event log by Lu et al. in [82]. The processes are represented as multi-level interleaved sub-processes. The activity trees are used to define the hierarchical relations between the process model activities. The proposed approach can be fully automated or fully supervised by a process expert. While our approach can be fully automated given that the attribute labels are consistent across the event log repository.

Mendling et al. [92] argue that most of the approaches applied to a collection of business processes deal with the verification techniques based on the decomposition. Their goal is mostly focused on representing the hierarchical relations between a set of business processes [81].

That being said, the relations between a collection of business processes are ignored.

Malinova et al. [83, 84] evaluate the use of business process architecture in practice. The authors use the term process map which is defined as a synonym of the business process architecture. They interviewed 11 companies and evaluated 15 process maps in practice and observe that a fine-granular process is related to exactly one process model activity. This type of relation is captured by: the Hierarchical Process Architecture, the Pipeline Process Architecture and the Divisional Process Architecture. The Hierarchical Process Architecture is similar to the decomposition relation defined in [50] in which the business process model is decomposed into other business process models, each representing a sub process. They observed that organizations include at least three decomposition levels. The Pipeline Process Architecture is considered as a specialization of the previous relation. While the Divisional Process Architecture is defined as an extension of the two previously explained relations, where the processes are divided into units and each unit is further decomposed based on the previous types of relation. The authors emphasize that the business process architecture does not only helps the organizations to understand their processes better, but also improves the performance analysis. In addition, they argue that the organization's structure is an important artifact for designing the business process architecture. In this thesis, rather than discovering the decomposition relations between a pair of business process model, we are focused more on discovering the trigger and information flow relations between these processes.

To the best of our knowledge, none of these approaches focuses on identifying the business process architecture in term of trigger and information flow from a set of event logs that are extracted from the organization's database with the original purpose of discovering the designated process models.

6.7 EVALUATION

To evaluate the effectiveness of our approach, we are using the real-life event logs provided in the scope of the BPI Challenge 2017 [131] and BPI Challenge 2020 [132]. To evaluate the feasibility of our algorithms (presented in Section 6.4), we implemented them in Python using the PM4PY framework [17]. The implemented algorithms are available on GitHub repository².

The BPI challenge 2017 deals with loan applications. There are two event logs provided in the scope of this challenge: 1) the *Application* event log is about customer applications for obtaining a bank credit and it contains 1.160.405 events and 31.509 cases; 2) the *Offer* event log is about credit offers proposed by the financial institution as a response

² GitHub: https://github.com/DorinaBano/BPA_from_event_log

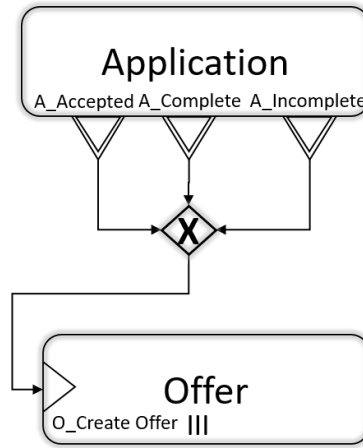


Figure 49: The discovered trigger flow (simple representation) of the BPI Challenge 2017

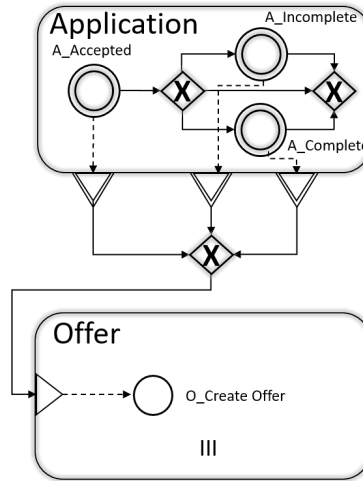


Figure 50: The discovered trigger flow (detailed representation) of the BPI Challenge 2017

to the customer’s application and it contains 93.846 events and 42.995 cases. Both event logs contain events from January 2016 to February 2017. Before applying the BPA discovery approach, we first clean the *Application* log of any event that is also found in the *Offer* log. This step is performed because we are only interested in discovering the BPA’s trigger and information flow patterns instead of the composition or specialization patterns where common events are of the most importance.

Following our approach, we first check whether these events logs are aware of each other according to Definition 6.1. Indeed that is the case: The *ApplicationID* attribute of the *Offer* event log refers to the *CaseID* attribute of the *Application* event log in all cases, hence $A_{W_{Offer,Application}} = 1$.

After determining the awareness, the event logs are joined into one event log, which is subject to the two algorithms presented in sec-

tion 6.4. The output auxiliary event log is mined using the Inductive Visual Miner algorithm [76]. The resulting process architecture model representing the trigger flow is obtained and illustrated in Figures 49 and 50.

We discovered that the application process has a 1 – to – n trigger relation with the offer process in that each application process instance can trigger more than one offer process instance. Specifically, there are three intermediate events in the application process as sources of the trigger flow: *A_Accepted*, *A_Complete* and *A_Incomplete* which triggers the offer process through the *O_CreateOffer* start event (see Figure 49). Looking at the internal behavior of the application process, represented in 50, we can observe that the offer process is always instantiated by the *A_Accepted* event. In addition, within the same application process instance, the offer process can be instantiated again by either *A_Complete* or *A_Incomplete* events. Looking only at the activity labels (*A_Incomplete*, *A_Complete*) of the events which instantiate the offer process might be confusing because they contradict each other. However, this is happening due to the fact that the application process can trigger more than one offer process (offer process depicted as a multi-instance in Figure 50).

With respect to information flow, the only common attribute that is shared between the event logs and is not static is the user associated with the activity being performed. In this sense, Figure 51 depicts the implicit information carried by particular employees when they switch to performing different tasks between the two processes. From the BPA depicted in Figure 51 we can understand in the positive case of the application getting accepted:

- the user who returns the offer (*O_Returned*) was also involved in validating the application (*A_Validate*)
- the user who cancels the offer (*O_Cancelled*) is previously involved with canceling (*A_Cancelled*) the application
- the user who successfully ends the application processes *A_Pending* (i.e., is always at the end of a positive application process) is the same one who accepted the offer (*O_Accepted*)

A detailed representation of the BPA is illustrated in Figure 52.

The BPI Challenge 2020 is concerned with five event logs and it contains information about travel expense claims from the Eindhoven University of Technology (TU/e) located in the Netherlands. Two types of trips are distinguished in the provided event logs: 1) domestic trips - the employee applies for reimbursement after the trip has taken place and 2) international declaration trips - the permission has to be approved by the supervisor before the trip take place.

Three out of five provided event logs are aware of each other (based on Definition 6.2): 1) *PrePaidTravelCost* which contains 2,099 cases and

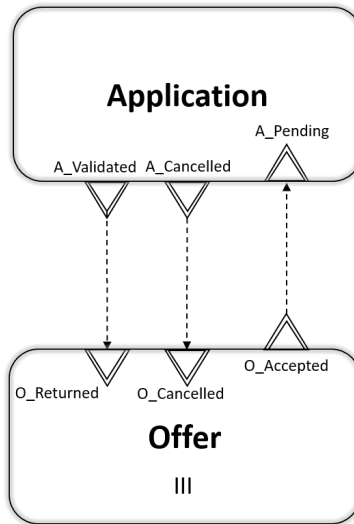


Figure 51: The discovered information flow (simple representation) of the BPI Challenge 2017

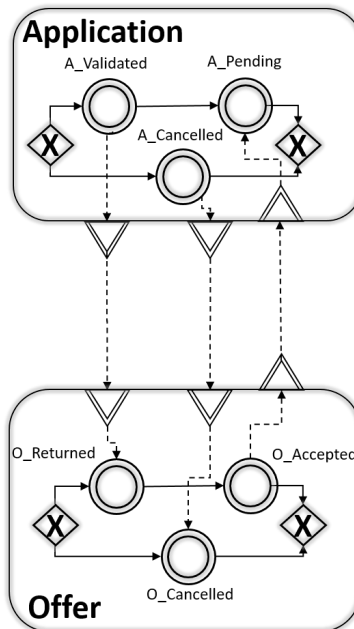


Figure 52: The discovered information flow (detailed representation) of the BPI Challenge 2017

18,246 events; 2) *InternationalDeclaration* with 6,449 cases, 72151 events; 3) *Permit* which holds 7,065 cases, 86,581 events. The *Permit id* attribute of the *PrePaidTravelCost* event log refers to the *CaseID* attribute of the *Permit* event log. In addition, the *Permit id2* attribute of the *InternationalDeclaration* event log refers to the *CaseID* attribute of the *Permit* event log. This implies that *PrePaidTravelCost* and *InternationalDeclaration* are both aware of *Permit* event log. Again, we cleaned the duplicate events for each pair of event logs under consideration.

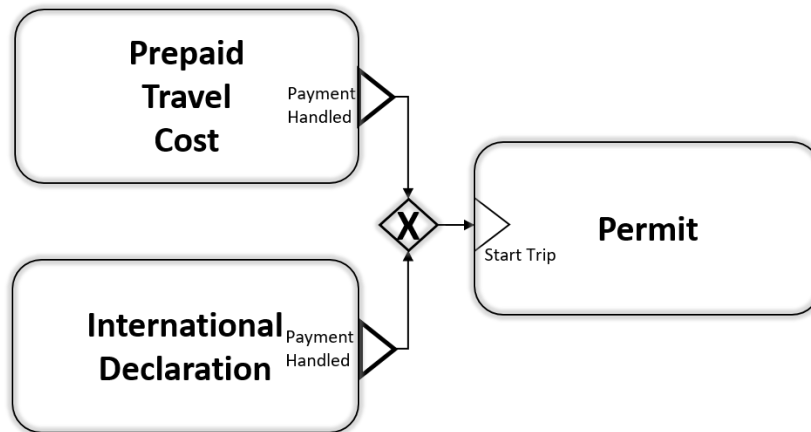


Figure 53: The discovered trigger flows of the BPI Challenge 2020

The BPI Challenge 2020 does not exhibit any information flow pattern as most of the attributes in the event logs are static, in that they do not change inside a case.

The resulting BPA with respect to the trigger flow for the aware event logs is illustrated in the Figure 53. The start event *Start Trip* of the *Permit* log is triggered either by the *Payment Handled* event of the *PrePaid-TravelCost* event log or by the same event of the *InternationalDeclaration* event log. This means that whenever the last event *Payment Handled* occurs in both event logs (i.e., *Prepaid Travel Cost* and *International Declaration*) the start event *Start Trip* of the *Permit* log is triggered. This can be explained by the fact that there are cases in the *Permit* event log where there no payments being handled. For those cases, the payment is handled in the previous processes. After the payment is handled the trip can start.

Looking at the discovered BPAs from these real-life event logs, we can conclude that the new insights, coming from the BPA's trigger and information flow, cannot be obtain by individually analyzing the event logs or, even better, just analyzing the process models. Therefore, by applying our method is not only possible to derive the relations between different business processes, each discovered from a different log, but it also possible to involve exclusive and parallel behavior between trigger and information flows.

6.8 SUMMARY

In this chapter, we provide an approach for discovering a business process architecture from a set of event logs that are extracted from the execution of processes within the same organization. The business process architecture provides a high-level perspective on an enterprise's running processes and the relation between these processes. It aids

in identifying and understanding the complex interdependencies and relationships between the processes.

Our method focuses on two such interdependencies: the trigger flow and the information flow between two or more given processes. For each pair of aware cases pertaining to two different event logs we detect the source and the target of the trigger/information flow instance. The target of the trigger flow instance is always the start event, while the source can be an end or intermediate event. In contrast, the target of the information flow it can be an intermediate event.

The discovered trigger and information flow relation between two process models provide useful insights (e.g., to John, considering our story) as how information is shared between these processes and how the instantiation of a process depends on the execution of another process. The approach is not only able to identify such relations but also their complex behavior like, for example, a single process that spawns multiple instances of another process. To graphically visualize such complex behavior we have provided an extension of a BPA notion.

The approach's feasibility and effectiveness is evaluated by being applied to two real-life event logs. Looking at the discovered BPAs from these real-life event logs, we can conclude that the new insights, coming from the BPA's trigger and information flow, cannot be obtain by individually analyzing the event logs or, even better, just analyzing the process models. We show that the business process architecture provides an holistic view of the whole system and reveals process interdependencies that otherwise are not captured while organizing the business process models.

CONCLUSIONS

In this chapter we summarize the main contributions of this thesis, and briefly discuss limitations before providing an outlook of future work.

7.1 CONTRIBUTION SUMMARY

This thesis deals with the utilization of the event log data in the area of process mining with a focus on the event log generation, process discovery and process enhancement (see Figure 10). Thus, considering the more general business process management lifecycle [139], this thesis touches upon the analysis and evaluation phase, to which the process mining field belongs to.

This thesis aims to equip the process expert with a set of methods that allows her/him to exploit the event logs to their full capacity even when the access to the original database system and the domain knowledge is limited or unavailable. First, we raise the question of whether it is possible to generate an event log in the lack of a classical database system access. We show how that is indeed the case by introducing a method that generates an event log only from a redo log, which stores the data operations and ensures consistency and fault tolerance in case of system failure (e.g., power failure). Then we show how a data model that is implicitly contained in an event log can be extracted from it and put in the hands of the process expert to understand the underlying data perspective in the context of the running process. Even further, we show how this data is manipulated by deriving the data objects that are affected and impact the running process. Indeed the impact of the data spans across processes. To that end, for the first time, we show the process interdependencies (based on the data dependencies) in an organization by mining the organization's business process architecture. Each contribution of this thesis is depicted in Figure 54 and summarized below in particular:

1. The first introduced contribution (annotated with 1 in Figure 54) in this thesis is a method that enables event log extraction from the transactions made by the running information system without directly accessing the whole database system. This method does not require extensive domain knowledge. The data model representing the database schema is discovered by just considering the information stored in the form of redo log entries. This includes the discovery of the data model classes, attributes and the relations between them. The intervention of the domain expert is required after the data model is discovered and before the

event log extraction step takes place. The domain expert selects the root class in the discovered data model that represents the case notion in the to-be-discovered process model. Based on the discovered data model and the selected case notion, the event log is extracted by correlating the redo log entries with the event log cases. The extracted event log conforms to the XES standard and can be tailored to the process mining experts for applying traditional process mining techniques (e.g., discovery, conformance checking).

The feasibility of the proposed method is evaluated based on two synthetic redo logs: the MIMIC and Ticket Selling redo logs. To realize this, we focus on the database schema comparison. The discovered database schema by our method is compared to the original one showing their similarity hence proving our method's effectiveness.

2. In the following contribution, we introduce a method that discovers a data model from an event log (annotated with 2 in Figure 54), which constitutes a reverse-engineering method that aims to discover the original data structure from which the event log data is originally derived. The proposed method complements the discovered process model and increases its understandability. It shows how the event log attributes can be grouped into data model classes and how these classes are related to each other. Specifically, we look at the relation between the event log and attributes and activities, i.e., which activity writes which attribute. To realize this, first, an intermediate representation called Attribute-Access relationship diagram (A2A diagram) is extracted from the original event log capturing how the event log activities (on the process model level) access the event log attributes and how many times this access relation holds. Based on the information stored in the A2A diagram, a set of rules are applied in order to group the event log attributes into data model classes. These rules are organized based on two aspects: non/isolated attributes and non/isolated activities. An attribute is called isolated if all activities that access it (i.e., write a value) do not access other attributes. Likewise, an activity is called isolated if all the attributes it accesses are not accessed by any other activity. The discovered data model serves the role of a complementary artifact that provides domain context to the discovered process model, thus improving the overall understandability of the organization's running process. We argue that this type of information is neither explicitly captured by the process model nor by the event log, hence such a method is useful to unearth the valuable information implicitly found in event logs.

The feasibility of the proposed method is illustrated based on two real-life event logs: the Road Traffic Fine Management and MIMIC event log. It is shown that the discovered data model is similar to the original data structure provided by the specification document of the MIMIC dataset.

To show the applicability of the introduced method, two application scenarios are presented. In the first scenario, the data model is discovered from an unlabeled event log and used as the main structure to map the unlabeled events to the event log cases. The second scenario relies on discovering a data model that facilitates the automation and generation of a process-aware digital twin via a low-code development paradigm.

3. The next contribution consists of a method that is able to discover the data objects lifecycles by considering as sole input an event log (annotated with 3 in Figure 54). Different from the previous method in which we track how the event log activities access the event log attributes, in this method, we track how the event log activities modify (i.e., change the value) the event log attributes for each case (this method is case aware) reflecting how the business process activities, on the model level of abstraction, modify the data during the process execution. The behavior of each singular event log attribute is discovered and plays an important role in grouping the event log attributes into data objects. Subsequently, the discovered data objects are used to enrich the process model with a data-flow perspective complementing the control perspective. In addition, we enrich the process model with statistical information that reveals how often activity instances manipulate the data objects in the historical execution. This helps the process expert to understand the importance and the impact of certain data objects during the process execution. All in all, the data objects accompanying the process model are an intrinsic part of process discovery in that they improve the understating of the process model to accurately represent what has happened in the real world.

The method's feasibility is checked based on two real-life event logs: the Road Traffic Fine Management and Hospital Billing event log. It is shown that enriching the process model with the data objects and their flow delivers information that proves to be useful when it comes to the process model understandability.

4. Last but not least, the final contribution of this paper is a method that automatically discovers on a higher level of abstraction how an organization's business processes ultimately affect each other, sometimes even depending on the well execution of each other. Concretely, our method discovers the relations between a pair of business processes that are beforehand discovered from different

event logs extracted from the same database (annotated with 4 in Figure 54). These process model relations/dependencies are represented as trigger flow (i.e., a process triggers the instantiation of another process) and information flow (i.e., a process exchanges data with another process). These relations are discovered between a pair of event logs that are aware, i.e., there exist a significant number of event log cases that contain explicit information about the existence of certain cases pertaining the opposite event log. Since these flows are directional, we detect the target and the source of each relation. The target of a trigger flow is always a start event and the source can be an end or an intermediate event from the opposite event log. In contrast, the target of the information flow can not be an end event and the source can not be a start event. We analyze the causality relationship between these events for discovering the trigger flow relation, and the information contained in the event attributes for discovering the information flow relation. The trigger and the information flow are captured in a business process architecture modeling language.

Process architectures are usually modeled top-down by interviewing process experts. In this thesis, for the first time, we introduce a method that discovers such process architecture from the historical execution of an organization's business processes. Besides the trigger and information flow, the proposed method is able to capture the behavior inside the processes involved in a trigger or information flow relation, e.g., a process instance might be triggered by two mutually exclusive throwing events. With such information, it is possible to understand when a certain activity is always instantiating or sending information to another process or if this holds only for a subset of cases. To graphically represent and visualize such behavior, the business process architecture is extended with additional notions borrowed from BPMN, specifically the XOR and AND split/join gateways.

To evaluate the feasibility of our method, we are using two real-life event logs provided in the scope of the BPI Challenge 2017 and BPI Challenge 2020. It is shown that the relations between processes are not always defined as 1-to-1 but there are also cases in which two processes are in a 1-to-n trigger flow relation, meaning that one process instance can instantiate more than one instance from the opposite process. In addition, it is shown that some events are always involved in a relation between two pairs of processes and some others only for a subset of cases. To accommodate this kind of complex behavior we extended the business process architecture's graphical representation.

Each method is totally independent from each other and can be used as standalone anytime the requirements are fulfilled. They are all im-

plemented as a prototype in Python except the one for the event log extraction from the Redo Logs, which is implemented in Scala.

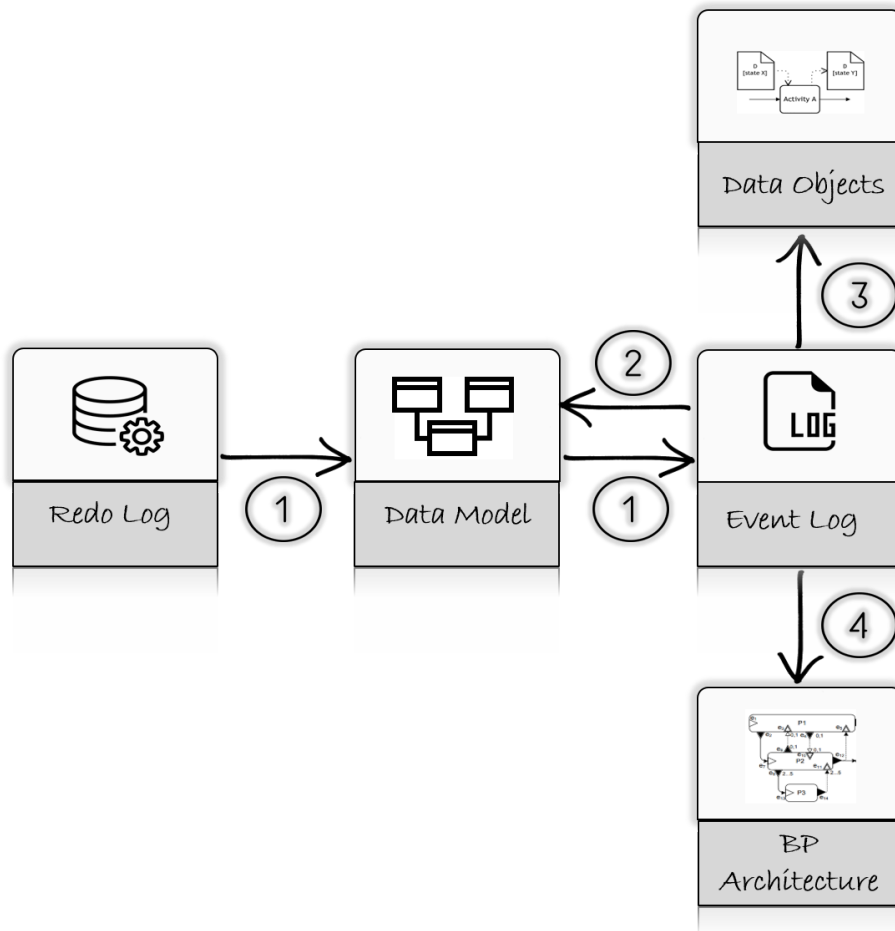


Figure 54: The overall contributions of this thesis

7.2 LIMITATIONS

The method presented in this thesis for discovering a data model from an event log considers as input a single event log. However, within an organization, it is possible to extract more than one event log, each serving a different business goal. Applying our method to a set of event logs would lead to discovering a dedicated data model for each log. Since the event logs are extracted from the same database, it might happen that several logs contain information about the same database's attributes, but they might cover different time periods. Therefore, the discovered data models might lead to inconsistent information regarding the common attributes, which might not fully represent the original database schema. This is due to a single event log having a narrow and limited view of the overall behavior inside an organization. Discovering

dependencies between several process models that are extracted from the same organization's database is considered in Chapter 6. However, it covers only the information exchanges between the corresponding processes and not a global data model that represents the holistic data perspective affected by these processes.

In addition, the method for discovering the data model from an event log is able to discover the association relations between a pair of data model classes based on the information pertained in the A2A diagram. However, languages like UML [108] support other types of relations such as inheritance or composition. Discovering such types of relations might be helpful for the process expert to understand better the discovered processes and the data dependency pertaining to the event log. For example, if an association with a parent class is discovered, the same association with the child classes would have been missed.

A similar limitation holds for the relations discovered between a pair of business processes, defined via the trigger and information flow relations. In the literature, other types of relations can be discovered in the scope of BPA. For example, specialization implies that one process is a specialized version of another process. Discovering other types of relations between business processes might draw a more accurate picture of the overall architecture and better support the process expert with a holistic view of the running processes.

In addition, the relations between processes can be handled in a synchronous or asynchronous environment. Decker and Weske [43] assume that two processes exchange synchronous messages in a choreography setting. However, the trigger and information flow relations discovered by our method are assumed to be asynchronous. Our methods need to be extended accordingly to capture this property.

7.3 FUTURE WORK

This section briefly explains some directions for future work that can extend the contributions presented in this thesis.

- Applying all the methods presented in this thesis in a real-life setting. Since the event log in reality might contain more cases, attributes and events it is interesting to measure the performance of these methods and apply some optimization techniques where it is necessary. In addition, it would be interesting to track what concrete issues might appear which is currently not foreseen by these methods.
- Introducing our methods to a few process experts to observe and measure how they apply them to real-life data can reveal important insights. It would be interesting to know what challenges the process experts might face during the application process. In addition, we want to know how much the information discovered

from these methods can support the process experts in better understanding the process model and the event log. One option to achieve this is to equip them with the process model, the event log and a list of questions related to them. Two scenarios might be useful to consider:

- In the first scenario, the process experts are asked to answer the corresponding question by just considering as input the event log and the discovered process model.
- In the second scenario, besides the event log and the process model, the process experts are also equipped with the data model, the data objects enhanced on top of the discovered process model, and the relations between the processes represented as business process architecture.

For both scenarios, we would like to check the time required by the process experts to run both experiments. It might also be worth discussing the challenges they faced in each setting in a similar way as in [140].

- Currently, the method for discovering a data model from an event log is limited to support only the classes related to the association relations. However, other types of relations exist, such as composition, aggregation and inheritance [108]. The discovery of such UML constructs can yield a data model which is closer to reality.
- Next, one can try to apply the methods for discovering the data model and data object lifecycles not to a single event log but to a set of logs that are all extracted from the same database but represent different case notions, thus yielding different process models. Extracting a global data model from a set of event logs might deliver a data model that is more closely related to reality and, consequently, represents better the organization's data structure. Usually, the event logs are characterized by their flattened structure and when a single event log is extracted from the organization's database, it is likely that important data is not captured. Even the data which is captured is seen only from a single perspective. One solution to achieve this is to apply the proposed method to each event log and then extend the method to cover the schema matching like the one presented in [109, 113].
- Future work includes investigating other types of relations between processes in a BPA, namely, the specialization and composition relations. These would require the investigation of exact same events that appear across multiple event logs to determine whether a process is a composition of another one or a detailed process of a more general one.

- It might also be interesting to discover the trigger and information flow by also considering a global data model, which is upfront discovered from the same set of event logs all extracted from the same database. The business process architecture is derived based on the data model structure. In literature, these types of derivations are called goal-based approaches [45] since each class in the data model defines a goal, which has to be reflected in the business process architecture. It might also be interesting to investigate and analyze the results of the architecture derived from the data model structure and compare it with the BPA output by the method introduced in this thesis.

To conclude, this thesis equips John with a set of methods which aims to derive an event log in the absence of a proper database access, extract the underlying data model from an event log and, finally, understand how the data impacts the process not only in isolation but also how it manifests dependencies between co-existing business processes.

BIBLIOGRAPHY

- [1] IEEE standard for eXtensible event stream (XES) for achieving interoperability in event logs and event streams. *IEEE Std 1849-2016*, pages 1–50, 2016. doi: 10.1109/IEEESTD.2016.7740858. (Cited on page 21.)
- [2] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. Profiling relational data: a survey. *VLDB J.*, 24(4):557–581, 2015. doi: 10.1007/s00778-015-0389-y. URL <https://doi.org/10.1007/s00778-015-0389-y>. (Cited on page 29.)
- [3] Arya Adriansyah, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Conformance checking using cost-based fitness analysis. In *Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2011, Helsinki, Finland, August 29 - September 2, 2011*, pages 55–64. IEEE Computer Society, 2011. doi: 10.1109/EDOC.2011.12. URL <https://doi.org/10.1109/EDOC.2011.12>. (Cited on page 20.)
- [4] Amine Abbad Andaloussi, Andrea Burattin, and Barbara Weber. Toward an automated labeling of event log attributes. In Jens Gulden, Iris Reinhartz-Berger, Rainer Schmidt, Sérgio Guerreiro, Wided Guédria, and Palash Bera, editors, *Enterprise, Business-Process and Information Systems Modeling - 19th International Conference, BPMDS 2018, 23rd International Conference, EMMSAD 2018, Held at CAiSE 2018, Tallinn, Estonia, June 11-12, 2018, Proceedings*, volume 318 of *Lecture Notes in Business Information Processing*, pages 82–96. Springer, 2018. doi: 10.1007/978-3-319-91704-7_6. URL https://doi.org/10.1007/978-3-319-91704-7_6. (Cited on page 61.)
- [5] Annie I. Antón, W. Michael McCracken, and Colin Potts. Goal decomposition and scenario analysis in business process reengineering. In Gerard Wijers, Sjaak Brinkkemper, and Anthony I. Wasserman, editors, *Advanced Information Systems Engineering, CAiSE'94, Utrecht, The Netherlands, June 6-10, 1994, Proceedings*, volume 811 of *Lecture Notes in Computer Science*, pages 94–104. Springer, 1994. doi: 10.1007/3-540-58113-8_164. URL https://doi.org/10.1007/3-540-58113-8_164. (Cited on page 103.)
- [6] Dorina Bano and Mathias Weske. Discovering data models from event logs. In Gillian Dobbie, Ulrich Frank, Gerti Kappel, Stephen W. Liddle, and Heinrich C. Mayr, editors, *Conceptual Modeling - 39th International Conference, ER 2020, Vienna, Austria, November 3-6, 2020, Proceedings*, volume 12400 of *Lecture Notes in Computer Science*, pages 62–76. Springer, 2020. doi: 10.1007/978-3-030-62522-1_5. URL https://doi.org/10.1007/978-3-030-62522-1_5. (Cited on pages 45 and 82.)
- [7] Dorina Bano, Tom Lichtenstein, Finn Klessascheck, and Mathias Weske. Database-less extraction of event logs from redo logs. In Witold Abramowicz, Sören Auer, and Elzbieta Lewanska, editors, *24th International Conference on Business Information Systems, BIS 2021, Hannover, Germany, June 15-17, 2021*, pages 73–82, 2021. doi: 10.52825/bis.v1i.66. URL <https://doi.org/10.52825/bis.v1i.66>. (Cited on page 25.)

- [8] Dorina Bano, Adriatik Nikaj, and Mathias Weske. Discovering business process architectures from event logs. In Artem Polyvyanyy, Moe Thandar Wynn, Amy Van Looy, and Manfred Reichert, editors, *Business Process Management Forum - BPM Forum 2021, Rome, Italy, September 06-10, 2021, Proceedings*, volume 427 of *Lecture Notes in Business Information Processing*, pages 162–177. Springer, 2021. doi: 10.1007/978-3-030-85440-9_10. URL https://doi.org/10.1007/978-3-030-85440-9_10. (Cited on page 91.)
- [9] Dorina Bano, Francesca Zerbato, Barbara Weber, and Mathias Weske. Enhancing discovered process models with data object lifecycles. In *25th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2021, Gold Coast, Australia, October 25-29, 2021*, pages 124–133. IEEE, 2021. doi: 10.1109/EDOC52215.2021.00023. URL <https://doi.org/10.1109/EDOC52215.2021.00023>. (Cited on page 67.)
- [10] Dorina Bano, Judith Michael, Bernhard Rumpe, Simon Varga, and Mathias Weske. Process-aware digital twin cockpit synthesis from event logs. *J. Comput. Lang.*, 70:101121, 2022. doi: 10.1016/j.cola.2022.101121. URL <https://doi.org/10.1016/j.cola.2022.101121>. (Cited on pages 45, 61, 63, and 64.)
- [11] Andrea Batch and Niklas Elmqvist. The interactive visualization gap in initial exploratory data analysis. *IEEE Trans. Vis. Comput. Graph.*, 24(1):278–287, 2018. doi: 10.1109/TVCG.2017.2743990. URL <https://doi.org/10.1109/TVCG.2017.2743990>. (Cited on page 26.)
- [12] Ira D. Baxter, Andrew Yahin, Leonardo Mendonça de Moura, Marcelo Sant’Anna, and Lorraine Bier. Clone detection using abstract syntax trees. In *1998 International Conference on Software Maintenance, ICSM 1998, Bethesda, Maryland, USA, November 16-19, 1998*, pages 368–377. IEEE Computer Society, 1998. doi: 10.1109/ICSM.1998.738528. URL <https://doi.org/10.1109/ICSM.1998.738528>. (Cited on page 104.)
- [13] Dina Bayomie, Iman M. A. Helal, Ahmed Awad, Ehab Ezat, and Ali El Bastawissi. Deducing case ids for unlabeled event logs. In Manfred Reichert and Hajo A. Reijers, editors, *Business Process Management Workshops - BPM 2015, 13th International Workshops, Innsbruck, Austria, August 31 - September 3, 2015, Revised Papers*, volume 256 of *Lecture Notes in Business Information Processing*, pages 242–254. Springer, 2015. doi: 10.1007/978-3-319-42887-1_20. URL https://doi.org/10.1007/978-3-319-42887-1_20. (Cited on page 61.)
- [14] Dina Bayomie, Ahmed Awad, and Ehab Ezat. Correlating unlabeled events from cyclic business processes execution. In Selmin Nurcan, Pnina Soffer, Marko Bajec, and Johann Eder, editors, *Advanced Information Systems Engineering - 28th International Conference, CAiSE 2016, Ljubljana, Slovenia, June 13-17, 2016. Proceedings*, volume 9694 of *Lecture Notes in Computer Science*, pages 274–289. Springer, 2016. doi: 10.1007/978-3-319-39696-5_17. URL https://doi.org/10.1007/978-3-319-39696-5_17. (Cited on page 61.)
- [15] Dina Bayomie, Claudio Di Ciccio, Marcello La Rosa, and Jan Mendling. A probabilistic approach to event-case correlation for process mining. In Alberto H. F. Laender, Barbara Pernici, Ee-Peng Lim, and José Palazzo M.

- de Oliveira, editors, *Conceptual Modeling - 38th International Conference, ER 2019, Salvador, Brazil, November 4-7, 2019, Proceedings*, volume 11788 of *Lecture Notes in Computer Science*, pages 136–152. Springer, 2019. doi: 10.1007/978-3-030-33223-5_12. URL https://doi.org/10.1007/978-3-030-33223-5_12. (Cited on page 61.)
- [16] Ekaterina Bazhenova, Susanne Bülow, and Mathias Weske. Discovering decision models from event logs. In Witold Abramowicz, Rainer Alt, and Bogdan Franczyk, editors, *Business Information Systems - 19th International Conference, BIS 2016, Leipzig, Germany, July, 6-8, 2016, Proceedings*, volume 255 of *Lecture Notes in Business Information Processing*, pages 237–251. Springer, 2016. doi: 10.1007/978-3-319-39426-8_19. URL https://doi.org/10.1007/978-3-319-39426-8_19. (Cited on page 81.)
- [17] Alessandro Berti, Sebastiaan J. van Zelst, and W. van der Aalst. Process mining for python (PM4Py): Bridging the gap between process and data science. In *Proceedings of the ICPM Demo Track 2019*, pages 13–16, 2019. URL <http://ceur-ws.org/Vol-2374/>. (Cited on pages 82 and 105.)
- [18] Alessandro Berti, Gyunam Park, Majid Rafiei, and Wil M. P. van der Aalst. An event data extraction approach from SAP ERP for process mining. In Jorge Munoz-Gama and Xixi Lu, editors, *Process Mining Workshops - ICPM 2021 International Workshops, Eindhoven, The Netherlands, October 31 - November 4, 2021, Revised Selected Papers*, volume 433 of *Lecture Notes in Business Information Processing*, pages 255–267. Springer, 2021. doi: 10.1007/978-3-030-98581-3_19. URL https://doi.org/10.1007/978-3-030-98581-3_19. (Cited on pages 1 and 26.)
- [19] Francis Bordeleau, Benoît Combemale, Romina Eramo, Mark van den Brand, and Manuel Wimmer. Towards model-driven digital twin engineering: Current opportunities and future challenges. In Önder Babur, Joachim Denil, and Birgit Vogel-Heuser, editors, *Systems Modelling and Management - First International Conference, ICSMM 2020, Bergen, Norway, June 25-26, 2020, Proceedings*, volume 1262 of *Communications in Computer and Information Science*, pages 43–54. Springer, 2020. doi: 10.1007/978-3-030-58167-1_4. URL https://doi.org/10.1007/978-3-030-58167-1_4. (Cited on page 63.)
- [20] Jagadeesh Chandra J. C. Bose, R. S. Mans, and Wil M. P. van der Aalst. Wanna improve process mining results? In *IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2013, Singapore, 16-19 April, 2013*, pages 127–134. IEEE, 2013. doi: 10.1109/CIDM.2013.6597227. URL <https://doi.org/10.1109/CIDM.2013.6597227>. (Cited on page 74.)
- [21] Drazen Brdjanin, Goran Banjac, and Slavko Maric. Automated synthesis of initial conceptual database model based on collaborative business process model. In Ana Madevska Bogdanova and Dejan Gjorgjevikj, editors, *ICT Innovations 2014 - World of Data, Ohrid, Macedonia, 1-4 October, 2014*, volume 311 of *Advances in Intelligent Systems and Computing*, pages 145–156. Springer, 2014. doi: 10.1007/978-3-319-09879-1_15. URL https://doi.org/10.1007/978-3-319-09879-1_15. (Cited on pages 55 and 56.)

- [22] Drazen Brdjanin, Danijela Banjac, Goran Banjac, and Slavko Maric. An online business process model-driven generator of the conceptual database model. *Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics*, 2018. (Cited on pages 55 and 56.)
- [23] Marius Breitmayer and Manfred Reichert. Towards the discovery of object-aware processes. In Johannes Manner, Stephan Haarmann, Stefan Kolb, and Oliver Kopp, editors, *Proceedings of the 12th ZEUS Workshop on Services and their Composition, Potsdam, Germany, February 20-21, 2020*, volume 2575 of *CEUR Workshop Proceedings*, pages 1–4. CEUR-WS.org, 2020. URL <http://ceur-ws.org/Vol-2575/paper1.pdf>. (Cited on pages 55 and 56.)
- [24] Andrea Burattin and Roberto Vigo. A framework for semi-automated process instance discovery from decorative attributes. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, part of the IEEE Symposium Series on Computational Intelligence 2011, April 11-15, 2011, Paris, France*, pages 176–183. IEEE, 2011. doi: 10.1109/CIDM.2011.5949450. URL <https://doi.org/10.1109/CIDM.2011.5949450>. (Cited on page 61.)
- [25] Diego Calvanese, Silvio Ghilardi, Alessandro Gianola, Marco Montali, and Andrey Rivkin. Formal modeling and smt-based parameterized verification of data-aware BPMN. In Thomas T. Hildebrandt, Boudewijn F. van Dongen, Maximilian Röglinger, and Jan Mendling, editors, *Business Process Management - 17th International Conference, BPM 2019, Vienna, Austria, September 1-6, 2019, Proceedings*, volume 11675 of *Lecture Notes in Computer Science*, pages 157–175. Springer, 2019. doi: 10.1007/978-3-030-26619-6_12. URL https://doi.org/10.1007/978-3-030-26619-6_12. (Cited on page 68.)
- [26] Peter P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976. doi: 10.1145/320434.320440. URL <https://doi.org/10.1145/320434.320440>. (Cited on page 17.)
- [27] Michele Chinosi and Alberto Trombetta. BPMN: an introduction to the standard. *Comput. Stand. Interfaces*, 34(1):124–134, 2012. doi: 10.1016/j.csi.2011.06.002. URL <https://doi.org/10.1016/j.csi.2011.06.002>. (Cited on page 13.)
- [28] Carlo Combi, Barbara Oliboni, Mathias Weske, and Francesca Zerbatò. Conceptual modeling of processes and data: Connecting different perspectives. In Juan Trujillo, Karen C. Davis, Xiaoyong Du, Zhanhuai Li, Tok Wang Ling, Guoliang Li, and Mong-Li Lee, editors, *Conceptual Modeling - 37th International Conference, ER 2018, Xi'an, China, October 22-25, 2018, Proceedings*, volume 11157 of *Lecture Notes in Computer Science*, pages 236–250. Springer, 2018. doi: 10.1007/978-3-030-00847-5_18. URL https://doi.org/10.1007/978-3-030-00847-5_18. (Cited on page 68.)
- [29] Carlo Combi, Barbara Oliboni, Mathias Weske, and Francesca Zerbatò. Conceptual modeling of inter-dependencies between processes and data.

- In Hisham M. Haddad, Roger L. Wainwright, and Richard Chbeir, editors, *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018, Pau, France, April 09-13, 2018*, pages 110–119. ACM, 2018. doi: 10.1145/3167132.3167141. URL <https://doi.org/10.1145/3167132.3167141>. (Cited on page 68.)
- [30] Raffaele Conforti, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. Beyond tasks and gateways: Discovering BPMN models with subprocesses, boundary events and activity markers. In Shazia Wasim Sadiq, Pnina Soffer, and Hagen Völzer, editors, *Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings*, volume 8659 of *Lecture Notes in Computer Science*, pages 101–117. Springer, 2014. doi: 10.1007/978-3-319-10172-9_7. URL https://doi.org/10.1007/978-3-319-10172-9_7. (Cited on page 104.)
- [31] Raffaele Conforti, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. BPMN miner: Automated discovery of BPMN process models with hierarchical structure. *Inf. Syst.*, 56:284–303, 2016. doi: 10.1016/j.is.2015.07.004. URL <https://doi.org/10.1016/j.is.2015.07.004>. (Cited on page 104.)
- [32] Estrela Ferreira Cruz, Ricardo Jorge Machado, and Maribel Yasmina Santos. From business process modeling to data model: A systematic approach. In João Pascoal Faria, Alberto Rodrigues da Silva, and Ricardo Jorge Machado, editors, *8th International Conference on the Quality of Information and Communications Technology, QUATIC 2012, Lisbon, Portugal, 2-6 September 2012, Proceedings*, pages 205–210. IEEE Computer Society, 2012. doi: 10.1109/QUATIC.2012.31. URL <https://doi.org/10.1109/QUATIC.2012.31>. (Cited on pages 54 and 56.)
- [33] Estrela Ferreira Cruz, Ricardo J. Machado, and Maribel Yasmina Santos. Deriving a data model from a set of interrelated business process models. In Slimane Hammoudi, Leszek A. Maciaszek, and Ernest Teniente, editors, *ICEIS 2015 - Proceedings of the 17th International Conference on Enterprise Information Systems, Volume 2, Barcelona, Spain, 27-30 April, 2015*, pages 49–59. SciTePress, 2015. doi: 10.5220/0005366100490059. URL <https://doi.org/10.5220/0005366100490059>. (Cited on pages 54 and 56.)
- [34] Dusanka Dakic, Darko Stefanovic, Teodora Lolic, Dajana Narandzic, and Nenad Simeunovic. Event log extraction for the purpose of process mining: A systematic literature review. In Gabriela Prostean, Juan José Lavios Villahoz, Laura Brancu, and Gyula Bakacsi, editors, *Innovation in Sustainable Management and Entrepreneurship*, pages 299–312, Cham, 2020. Springer International Publishing. ISBN 978-3-030-44711-3. (Cited on pages 1 and 26.)
- [35] Vinicius Stein Dani, Henrik Leopold, Jan Martijn E. M. van der Werf, Xixi Lu, Iris Beerepoot, Jelmer Jan Koorn, and Hajo A. Reijers. Towards understanding the role of the human in event log extraction. In Andrea Marrella and Barbara Weber, editors, *Business Process Management Workshops - BPM 2021 International Workshops, Rome, Italy, September 6-10, 2021, Revised Selected Papers*, volume 436 of *Lecture Notes in Business Information Processing*, pages 86–98. Springer, 2021.

- doi: 10.1007/978-3-030-94343-1_7. URL https://doi.org/10.1007/978-3-030-94343-1_7. (Cited on page 1.)
- [36] Thomas H Davenport. *Process innovation: reengineering work through information technology*. Harvard Business Press, 1993. (Cited on page 11.)
- [37] Massimiliano de Leoni and Wil M. P. van der Aalst. Data-aware process mining: discovering decisions in processes using alignments. In Sung Y. Shin and José Carlos Maldonado, editors, *Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, Coimbra, Portugal, March 18-22, 2013*, pages 1454–1461. ACM, 2013. doi: 10.1145/2480362.2480633. URL <https://doi.org/10.1145/2480362.2480633>. (Cited on page 68.)
- [38] Massimiliano de Leoni and Wil M. P. van der Aalst. Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming. In Florian Daniel, Jianmin Wang, and Barbara Weber, editors, *Business Process Management - 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings*, volume 8094 of *Lecture Notes in Computer Science*, pages 113–129. Springer, 2013. doi: 10.1007/978-3-642-40176-3_10. URL https://doi.org/10.1007/978-3-642-40176-3_10. (Cited on pages 80 and 81.)
- [39] Eduardo González López de Murillas, Wil M. P. van der Aalst, and Hajo A. Reijers. Process mining on databases: Unearthing historical data from redo logs. In Hamid Reza Motahari-Nezhad, Jan Recker, and Matthias Weidlich, editors, *Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings*, volume 9253 of *Lecture Notes in Computer Science*, pages 367–385. Springer, 2015. doi: 10.1007/978-3-319-23063-4_25. URL https://doi.org/10.1007/978-3-319-23063-4_25. (Cited on page 74.)
- [40] Eduardo González López de Murillas, Wil M. P. van der Aalst, and Hajo A. Reijers. Process mining on databases: Unearthing historical data from redo logs. In Hamid Reza Motahari-Nezhad, Jan Recker, and Matthias Weidlich, editors, *Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings*, volume 9253 of *Lecture Notes in Computer Science*, pages 367–385. Springer, 2015. doi: 10.1007/978-3-319-23063-4_25. URL https://doi.org/10.1007/978-3-319-23063-4_25. (Cited on pages 27, 35, 36, 37, and 41.)
- [41] Eduardo González López de Murillas, G. E. Hoogendoorn, and Hajo A. Reijers. Redo log process mining in real life: Data challenges & opportunities. In Ernest Teniente and Matthias Weidlich, editors, *Business Process Management Workshops - BPM 2017 International Workshops, Barcelona, Spain, September 10-11, 2017, Revised Papers*, volume 308 of *Lecture Notes in Business Information Processing*, pages 573–587. Springer, 2017. doi: 10.1007/978-3-319-74030-0_45. URL https://doi.org/10.1007/978-3-319-74030-0_45. (Cited on pages 26, 27, and 36.)
- [42] Eduardo González López de Murillas, Hajo A. Reijers, and Wil M. P. van der Aalst. Case notion discovery and recommendation: automated event log building on databases. *Knowl. Inf. Syst.*, 62(7):2539–2575, 2020. doi: 10.1007/s10115-019-01430-6. URL <https://doi.org/10.1007/s10115-019-01430-6>. (Cited on page 26.)

- [43] Gero Decker and Mathias Weske. Interaction-centric modeling of process choreographies. *Inf. Syst.*, 36(2):292–312, 2011. doi: 10.1016/j.is.2010.06.005. URL <https://doi.org/10.1016/j.is.2010.06.005>. (Cited on page 116.)
- [44] Kiarash Diba, Kimon Batoulis, Matthias Weidlich, and Mathias Weske. Extraction, correlation, and abstraction of event data for process mining. *WIRES Data Mining Knowl. Discov.*, 10(3), 2020. doi: 10.1002/widm.1346. URL <https://doi.org/10.1002/widm.1346>. (Cited on pages 1, 26, 46, and 68.)
- [45] Remco M. Dijkman, Irene T. P. Vanderfeesten, and Hajo A. Reijers. Business process architectures: overview, comparison and framework. *Enterp. Inf. Syst.*, 10(2):129–158, 2016. doi: 10.1080/17517575.2014.928951. URL <https://doi.org/10.1080/17517575.2014.928951>. (Cited on pages 103 and 118.)
- [46] R.M. Dijkman, I.T.P. Vanderfeesten, and H.A. Reijers. *The road to a business process architecture : an overview of approaches and their use*. BETA Working Paper. Technische Universiteit Eindhoven, 2011. (Cited on pages 2, 14, and 92.)
- [47] Marlon Dumas, Luciano García-Bañuelos, Marcello La Rosa, and Reina Uba. Fast detection of exact clones in business process model repositories. *Inf. Syst.*, 38(4):619–633, 2013. doi: 10.1016/j.is.2012.07.002. URL <https://doi.org/10.1016/j.is.2012.07.002>. (Cited on page 104.)
- [48] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management, Second Edition*. Springer, 2018. ISBN 978-3-662-56508-7. doi: 10.1007/978-3-662-56509-4. URL <https://doi.org/10.1007/978-3-662-56509-4>. (Cited on pages 4 and 11.)
- [49] Rami-Habib Eid-Sabbagh and Mathias Weske. Analyzing business process architectures. In Camille Salinesi, Moira C. Norrie, and Oscar Pastor, editors, *Advanced Information Systems Engineering - 25th International Conference, CAiSE 2013, Valencia, Spain, June 17-21, 2013. Proceedings*, volume 7908 of *Lecture Notes in Computer Science*, pages 208–223. Springer, 2013. doi: 10.1007/978-3-642-38709-8_14. URL https://doi.org/10.1007/978-3-642-38709-8_14. (Cited on pages 2 and 104.)
- [50] Rami-Habib Eid-Sabbagh, Remco M. Dijkman, and Mathias Weske. Business process architecture: Use and correctness. In Alistair Barros, Avigdor Gal, and Ekkart Kindler, editors, *Business Process Management - 10th International Conference, BPM 2012, Tallinn, Estonia, September 3-6, 2012. Proceedings*, volume 7481 of *Lecture Notes in Computer Science*, pages 65–81. Springer, 2012. doi: 10.1007/978-3-642-32885-5_5. URL https://doi.org/10.1007/978-3-642-32885-5_5. (Cited on pages xiii, 2, 14, 15, 93, 101, 104, and 105.)
- [51] Chathura C. Ekanayake, Marlon Dumas, Luciano García-Bañuelos, Marcello La Rosa, and Arthur H. M. ter Hofstede. Approximate clone detection in repositories of business process models. In Alistair Barros, Avigdor Gal, and Ekkart Kindler, editors, *Business Process Management - 10th International Conference, BPM 2012, Tallinn, Estonia, September 3-6, 2012*.

- Proceedings*, volume 7481 of *Lecture Notes in Computer Science*, pages 302–318. Springer, 2012. doi: 10.1007/978-3-642-32885-5_24. URL https://doi.org/10.1007/978-3-642-32885-5_24. (Cited on page 104.)
- [52] Rik Eshuis and Pieter Van Gorp. Synthesizing object life cycles from business process models. *Softw. Syst. Model.*, 15(1):281–302, 2016. doi: 10.1007/s10270-014-0406-4. URL <https://doi.org/10.1007/s10270-014-0406-4>. (Cited on page 80.)
- [53] Dirk Fahland. Artifact-centric process mining. In Sherif Sakr and Albert Y. Zomaya, editors, *Encyclopedia of Big Data Technologies*. Springer, 2019. doi: 10.1007/978-3-319-63962-8_93-1. URL https://doi.org/10.1007/978-3-319-63962-8_93-1. (Cited on page 81.)
- [54] Diogo R. Ferreira and Daniel Gillblad. Discovering process models from unlabelled event logs. In Umeshwar Dayal, Johann Eder, Jana Koehler, and Hajo A. Reijers, editors, *Business Process Management, 7th International Conference, BPM 2009, Ulm, Germany, September 8-10, 2009. Proceedings*, volume 5701 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 2009. doi: 10.1007/978-3-642-03848-8_11. URL https://doi.org/10.1007/978-3-642-03848-8_11. (Cited on pages 61 and 63.)
- [55] Silvio Ghilardi, Alessandro Gianola, Marco Montali, and Andrey Rivkin. Delta-bpmn: A concrete language and verifier for data-aware BPMN. In Artem Polyvyanyy, Moe Thandar Wynn, Amy Van Looy, and Manfred Reichert, editors, *Business Process Management - 19th International Conference, BPM 2021, Rome, Italy, September 06-10, 2021, Proceedings*, volume 12875 of *Lecture Notes in Computer Science*, pages 179–196. Springer, 2021. doi: 10.1007/978-3-030-85469-0_13. URL https://doi.org/10.1007/978-3-030-85469-0_13. (Cited on page 46.)
- [56] Stewart Green and Martyn A. Ould. A framework for classifying and evaluating process architecture methods. *Softw. Process. Improv. Pract.*, 10(4):415–425, 2005. doi: 10.1002/spip.244. URL <https://doi.org/10.1002/spip.244>. (Cited on pages 3 and 92.)
- [57] Giancarlo Guizzardi. *Ontological foundations for structural conceptual models*. PhD thesis, University of Twente, October 2005. (Cited on page 17.)
- [58] Christian W. Günther and Wil M. P. van der Aalst. Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings*, volume 4714 of *Lecture Notes in Computer Science*, pages 328–343. Springer, 2007. doi: 10.1007/978-3-540-75183-0_24. URL https://doi.org/10.1007/978-3-540-75183-0_24. (Cited on page 68.)
- [59] Tugba Gurgen Erdogan and Ayca Tarhan. A goal-driven evaluation method based on process mining for healthcare processes. *Applied Sciences*, 8(6), 2018. ISSN 2076-3417. doi: 10.3390/app8060894. URL <https://www.mdpi.com/2076-3417/8/6/894>. (Cited on page 26.)

- [60] Peter Gyngell. Reengineering the corporation: A manifesto for business revolution: Michael hammer and james champy harpercollins publishers USA published in UK by nicholas brealey publishing london (1993) 216 pp. *J. Strateg. Inf. Syst.*, 3(4):339–345, 1994. doi: 10.1016/0963-8687(94)90038-8. URL [https://doi.org/10.1016/0963-8687\(94\)90038-8](https://doi.org/10.1016/0963-8687(94)90038-8). (Cited on page 11.)
- [61] Constantin Houy, Peter Fettke, Peter Loos, Wil M. P. van der Aalst, and John Krogstie. Business process management in the large. *Bus. Inf. Syst. Eng.*, 3(6):385–388, 2011. doi: 10.1007/s12599-011-0181-5. URL <https://doi.org/10.1007/s12599-011-0181-5>. (Cited on page 11.)
- [62] OMG: <https://www.omg.org/spec/BPMN>. *Object Management Group. Business Process Model and Notation (BPMN)*. Version 2.0.2., Jan. 2014. (Cited on pages 17 and 101.)
- [63] Richard Hull. Artifact-centric business process models: Brief survey of research results and challenges. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems: OTM 2008, OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Monterrey, Mexico, November 9-14, 2008, Proceedings, Part II*, volume 5332 of *Lecture Notes in Computer Science*, pages 1152–1163. Springer, 2008. doi: 10.1007/978-3-540-88873-4_17. URL https://doi.org/10.1007/978-3-540-88873-4_17. (Cited on page 68.)
- [64] Mieke Jans and Pnina Soffer. From relational database to event log: Decisions with quality impact. In Ernest Teniente and Matthias Weidlich, editors, *Business Process Management Workshops - BPM 2017 International Workshops, Barcelona, Spain, September 10-11, 2017, Revised Papers*, volume 308 of *Lecture Notes in Business Information Processing*, pages 588–599. Springer, 2017. doi: 10.1007/978-3-319-74030-0_46. URL https://doi.org/10.1007/978-3-319-74030-0_46. (Cited on page 26.)
- [65] Mieke Jans, Pnina Soffer, and Toon Jouck. Building a valuable event log for process mining: an experimental exploration of a guided process. *Enterp. Inf. Syst.*, 13(5):601–630, 2019. doi: 10.1080/17517575.2019.1587788. URL <https://doi.org/10.1080/17517575.2019.1587788>. (Cited on page 26.)
- [66] Mieke Jans, Jochen De Weerd, Benoît Depaire, Marlon Dumas, and Gert Janssenswillen. Conformance checking in process mining. *Inf. Syst.*, 102: 101851, 2021. doi: 10.1016/j.is.2021.101851. URL <https://doi.org/10.1016/j.is.2021.101851>. (Cited on page 21.)
- [67] Dominik Janssen, Felix Mannhardt, Agnes Koschmider, and Sebastiaan J. van Zelst. Process model discovery from sensor event data. In Sander J. J. Leemans and Henrik Leopold, editors, *Process Mining Workshops - ICPM 2020 International Workshops, Padua, Italy, October 5-8, 2020, Revised Selected Papers*, volume 406 of *Lecture Notes in Business Information Processing*, pages 69–81. Springer, 2020. doi: 10.1007/978-3-030-72693-5_6. URL https://doi.org/10.1007/978-3-030-72693-5_6. (Cited on page 64.)
- [68] Feng Jiang, Ling Ma, Tim Broyd, and Ke Chen. Digital twin and its implementations in the civil engineering sector. *Automation in Construction*, 130:103838, 2021. ISSN 0926-5805. doi: <https://doi.org/10.1016/>

- j.autcon.2021.103838. URL <https://www.sciencedirect.com/science/article/pii/S0926580521002892>. (Cited on page 63.)
- [69] Lan Jiang and Felix Naumann. Holistic primary key and foreign key detection. *J. Intell. Inf. Syst.*, 54(3):439–461, 2020. doi: 10.1007/s10844-019-00562-z. URL <https://doi.org/10.1007/s10844-019-00562-z>. (Cited on page 29.)
- [70] Paul Johannesson. Representation and communication - a speech act based approach to information systems design. *Inf. Syst.*, 20(4):291–303, 1995. doi: 10.1016/0306-4379(95)00015-V. URL [https://doi.org/10.1016/0306-4379\(95\)00015-V](https://doi.org/10.1016/0306-4379(95)00015-V). (Cited on page 103.)
- [71] Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3:160035, 2016. (Cited on pages 37 and 63.)
- [72] Timotheus Kampik and Mathias Weske. Event log generation: An industry perspective. In Adriano Augusto, Asif Gill, Dominik Bork, Selmin Nurcan, Iris Reinhartz-Berger, and Rainer Schmidt, editors, *Enterprise, Business-Process and Information Systems Modeling - 23rd International Conference, BPMDS 2022 and 27th International Conference, EMM-SAD 2022, Held at CAiSE 2022, Leuven, Belgium, June 6-7, 2022, Proceedings*, volume 450 of *Lecture Notes in Business Information Processing*, pages 123–136. Springer, 2022. doi: 10.1007/978-3-031-07475-2_9. URL https://doi.org/10.1007/978-3-031-07475-2_9. (Cited on pages xiii and 20.)
- [73] Vagelio Kavakli and Pericles Loucopoulos. Goal-driven business process analysis application in electricity deregulation. *Inf. Syst.*, 24(3):187–207, 1999. doi: 10.1016/S0306-4379(99)00015-0. URL [https://doi.org/10.1016/S0306-4379\(99\)00015-0](https://doi.org/10.1016/S0306-4379(99)00015-0). (Cited on page 103.)
- [74] George Koliadis, Aditya K. Ghose, and Srinivas Padmanabhuni. Towards an enterprise business process architecture standard. In *2008 IEEE Congress on Services, Part I, SERVICES I 2008, Honolulu, Hawaii, USA, July 6-11, 2008*, pages 239–246. IEEE Computer Society, 2008. doi: 10.1109/SERVICES-1.2008.60. URL <https://doi.org/10.1109/SERVICES-1.2008.60>. (Cited on pages 7 and 92.)
- [75] Jochen Malte Küster, Ksenia Ryndina, and Harald C. Gall. Generation of business process models for object life cycle compliance. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings*, volume 4714 of *Lecture Notes in Computer Science*, pages 165–181. Springer, 2007. doi: 10.1007/978-3-540-75183-0_13. URL https://doi.org/10.1007/978-3-540-75183-0_13. (Cited on page 80.)
- [76] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Process and deviation exploration with inductive visual miner. In Lior Limonad and Barbara Weber, editors, *Proceedings of the BPM Demo Sessions 2014*

Co-located with the 12th International Conference on Business Process Management (BPM 2014), Eindhoven, The Netherlands, September 10, 2014, volume 1295 of *CEUR Workshop Proceedings*, page 46. CEUR-WS.org, 2014. URL <http://ceur-ws.org/Vol-1295/paper19.pdf>. (Cited on pages xiv, 1, 20, 41, 56, 58, 68, 72, 82, and 107.)

- [77] Henrik Leopold, Sergey Smirnov, and Jan Mendling. On the refactoring of activity labels in business process models. *Inf. Syst.*, 37(5):443–459, 2012. doi: 10.1016/j.is.2012.01.004. URL <https://doi.org/10.1016/j.is.2012.01.004>. (Cited on page 76.)
- [78] Guangming Li, Eduardo González López de Murillas, Renata Medeiros de Carvalho, and Wil M. P. van der Aalst. Extracting object-centric event logs to support process mining on databases. In *Information Systems in the Big Data Era - CAiSE Forum 2018, Tallinn, Estonia, June 11-15, 2018, Proceedings*, volume 317, pages 182–199. Springer, 2018. doi: 10.1007/978-3-319-92901-9_16. URL https://doi.org/10.1007/978-3-319-92901-9_16. (Cited on page 55.)
- [79] Tom Lichtenstein, Dorina Bano, and Mathias Weske. Attribute-driven case notion discovery for unlabeled event logs. In Andrea Marrella and Barbara Weber, editors, *Business Process Management Workshops - BPM 2021 International Workshops, Rome, Italy, September 6-10, 2021, Revised Selected Papers*, volume 436 of *Lecture Notes in Business Information Processing*, pages 111–122. Springer, 2021. doi: 10.1007/978-3-030-94343-1_9. URL https://doi.org/10.1007/978-3-030-94343-1_9. (Cited on pages 45 and 61.)
- [80] Ying Liu, Lin Zhang, Yuan Yang, Longfei Zhou, Lei Ren, Fei Wang, Rong Liu, Zhibo Pang, and M. Jamal Deen. A novel cloud-based framework for the elderly healthcare services using digital twin. *IEEE Access*, 7: 49088–49101, 2019. doi: 10.1109/ACCESS.2019.2909828. URL <https://doi.org/10.1109/ACCESS.2019.2909828>. (Cited on page 63.)
- [81] Niels Lohmann, Eric Verbeek, and Remco M. Dijkman. Petri net transformations for business processes - A survey. *Trans. Petri Nets Other Model. Concurr.*, 2:46–63, 2009. doi: 10.1007/978-3-642-00899-3_3. URL https://doi.org/10.1007/978-3-642-00899-3_3. (Cited on pages 55, 103, and 104.)
- [82] Xixi Lu, Avigdor Gal, and Hajo A. Reijers. Discovering hierarchical processes using flexible activity trees for event abstraction. In Boudewijn F. van Dongen, Marco Montali, and Moe Thandar Wynn, editors, *2nd International Conference on Process Mining, ICPM 2020, Padua, Italy, October 4-9, 2020*, pages 145–152. IEEE, 2020. doi: 10.1109/ICPM49681.2020.00030. URL <https://doi.org/10.1109/ICPM49681.2020.00030>. (Cited on page 104.)
- [83] Monika Malinova and Jan Mendling. The effect of process map design quality on process management success. In *21st European Conference on Information Systems, ECIS 2013, Utrecht, The Netherlands, June 5-8, 2013*, page 160, 2013. URL http://aisel.aisnet.org/ecis2013_cr/160. (Cited on page 105.)
- [84] Monika Malinova, Henrik Leopold, and Jan Mendling. An empirical investigation on the design of process architectures. In *11. Internationale*

- Tagung Wirtschaftsinformatik, Leipzig, Germany, February 27 – March 1, 2013*, page 75, 2013. URL <http://aisel.aisnet.org/wi2013/75>. (Cited on page 105.)
- [85] Felix Mannhardt. Hospital billing - event log. *4TU.ResearchData. Dataset.*, 2017. <https://doi.org/10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741>. (Cited on page 82.)
- [86] Felix Mannhardt and Daan Blinde. Analyzing the trajectories of patients with sepsis using process mining. In Jens Gulden, Selmin Nurcan, Iris Reinhartz-Berger, Wided Guédria, Palash Bera, Sérgio Guerreiro, Michael Fellmann, and Matthias Weidlich, editors, *Joint Proceedings of the Radar tracks at the 18th International Working Conference on Business Process Modeling, Development and Support (BPMDs), and the 22nd International Working Conference on Evaluation and Modeling Methods for Systems Analysis and Development (EMMSAD), and the 8th International Workshop on Enterprise Modeling and Information Systems Architectures (EMISA) co-located with the 29th International Conference on Advanced Information Systems Engineering 2017 (CAiSE 2017)*, Essen, Germany, June 12-13, 2017, volume 1859 of *CEUR Workshop Proceedings*, pages 72–80. CEUR-WS.org, 2017. URL <http://ceur-ws.org/Vol-1859/bpmds-08-paper.pdf>. (Cited on pages xvii and 83.)
- [87] Felix Mannhardt and Massimiliano de Leoni. Road traffic fine management process. *Eindhoven University of Technology, Eindhoven*, 2015. <https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5>. (Cited on pages xiv, 23, 56, 58, 63, and 82.)
- [88] Felix Mannhardt, Massimiliano de Leoni, Hajo A. Reijers, and Wil M. P. van der Aalst. Balanced multi-perspective checking of process conformance. *Computing*, 98(4):407–437, 2016. doi: 10.1007/s00607-015-0441-1. URL <https://doi.org/10.1007/s00607-015-0441-1>. (Cited on page 81.)
- [89] Felix Mannhardt, Massimiliano de Leoni, Hajo A. Reijers, and Wil M. P. van der Aalst. Data-driven process discovery - revealing conditional infrequent behavior from event logs. In Eric Dubois and Klaus Pohl, editors, *Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings*, volume 10253 of *Lecture Notes in Computer Science*, pages 545–560. Springer, 2017. doi: 10.1007/978-3-319-59536-8_34. URL https://doi.org/10.1007/978-3-319-59536-8_34. (Cited on page 81.)
- [90] Felix Mannhardt, Massimiliano de Leoni, Hajo A. Reijers, and Wil M. P. van der Aalst. Data-driven process discovery - revealing conditional infrequent behavior from event logs. In Eric Dubois and Klaus Pohl, editors, *Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings*, volume 10253 of *Lecture Notes in Computer Science*, pages 545–560. Springer, 2017. doi: 10.1007/978-3-319-59536-8_34. URL https://doi.org/10.1007/978-3-319-59536-8_34. (Cited on page 68.)
- [91] Jan Mendling. *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*, volume 6 of *Lecture*

- Notes in Business Information Processing*. Springer, 2008. ISBN 978-3-540-89223-6. doi: 10.1007/978-3-540-89224-3. URL <https://doi.org/10.1007/978-3-540-89224-3>. (Cited on page 11.)
- [92] Jan Mendling. Empirical studies in process model verification. *Trans. Petri Nets Other Model. Concurr.*, 2:208–224, 2009. doi: 10.1007/978-3-642-00899-3_12. URL https://doi.org/10.1007/978-3-642-00899-3_12. (Cited on page 104.)
- [93] Jan Mendling, Hajo A. Reijers, and Jan Recker. Activity labeling in process modeling: Empirical insights and recommendations. *Inf. Syst.*, 35(4):467–482, 2010. doi: 10.1016/j.is.2009.03.009. URL <https://doi.org/10.1016/j.is.2009.03.009>. (Cited on page 77.)
- [94] Andreas Meyer, Luise Pufahl, Dirk Fahland, and Mathias Weske. Modeling and enacting complex data dependencies in business processes. In Florian Daniel, Jianmin Wang, and Barbara Weber, editors, *Business Process Management - 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings*, volume 8094 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2013. doi: 10.1007/978-3-642-40176-3_14. URL https://doi.org/10.1007/978-3-642-40176-3_14. (Cited on pages 2, 18, 68, and 80.)
- [95] Farid Meziane, Nikos Athanasakis, and Sophia Ananiadou. Generating natural language specifications from UML class diagrams. *Requir. Eng.*, 13(1):1–18, 2008. doi: 10.1007/s00766-007-0054-0. URL <https://doi.org/10.1007/s00766-007-0054-0>. (Cited on page 46.)
- [96] Boleslaw Mikolajczak and Jian-Lun Chen. Workflow mining alpha algorithm - A complexity study. In Mieczyslaw A. Klopotek, Slawomir T. Wierzchon, and Krzysztof Trojanowski, editors, *Intelligent Information Processing and Web Mining, Proceedings of the International IIS: IIPWM'05 Conference held in Gdansk, Poland, June 13-16, 2005*, volume 31 of *Advances in Soft Computing*, pages 451–455. Springer, 2005. doi: 10.1007/3-540-32392-9_51. URL https://doi.org/10.1007/3-540-32392-9_51. (Cited on pages 1, 20, and 71.)
- [97] Adriatik Nikaj, Kimon Batoulis, and Mathias Weske. Rest-enabled decision making in business process choreographies. In Quan Z. Sheng, Eleni Stroulia, Samir Tata, and Sami Bhiri, editors, *Service-Oriented Computing - 14th International Conference, ICSOC 2016, Banff, AB, Canada, October 10-13, 2016, Proceedings*, volume 9936 of *Lecture Notes in Computer Science*, pages 547–554. Springer, 2016. doi: 10.1007/978-3-319-46295-0_34. URL https://doi.org/10.1007/978-3-319-46295-0_34. (Cited on page 103.)
- [98] Jan Recker and Jan Mendling. The state of the art of business process management research as published in the BPM conference - recommendations for progressing the field. *Bus. Inf. Syst. Eng.*, 58(1):55–72, 2016. doi: 10.1007/s12599-015-0411-3. URL <https://doi.org/10.1007/s12599-015-0411-3>. (Cited on page 1.)
- [99] Guy Redding, Marlon Dumas, Arthur H. M. ter Hofstede, and Adrian Iordachescu. Generating business process models from object behavior models. *Inf. Syst. Manag.*, 25(4):319–331, 2008.

- doi: 10.1080/10580530802384324. URL <https://doi.org/10.1080/10580530802384324>. (Cited on page 80.)
- [100] Manfred Reichert. Process and data: Two sides of the same coin? In Robert Meersman, Hervé Panetto, Tharam S. Dillon, Stefanie Rinderle-Ma, Peter Dadam, Xiaofang Zhou, Siani Pearson, Alois Ferscha, Sonia Bergamaschi, and Isabel F. Cruz, editors, *On the Move to Meaningful Internet Systems: OTM 2012, Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2012, Rome, Italy, September 10-14, 2012. Proceedings, Part I*, volume 7565 of *Lecture Notes in Computer Science*, pages 2–19. Springer, 2012. doi: 10.1007/978-3-642-33606-5_2. URL https://doi.org/10.1007/978-3-642-33606-5_2. (Cited on page 46.)
- [101] Simon Remy, Luise Pufahl, Jan-Philipp Sachs, Erwin P. Böttinger, and Mathias Weske. Event log generation in a health system: A case study. In Dirk Fahland, Chiara Ghidini, Jörg Becker, and Marlon Dumas, editors, *Business Process Management - 18th International Conference, BPM 2020, Seville, Spain, September 13-18, 2020, Proceedings*, volume 12168 of *Lecture Notes in Computer Science*, pages 505–522. Springer, 2020. doi: 10.1007/978-3-030-58666-9_29. URL https://doi.org/10.1007/978-3-030-58666-9_29. (Cited on pages 1 and 26.)
- [102] Marcello La Rosa, Marlon Dumas, Reina Uba, and Remco M. Dijkman. Business process model merging: An approach to business process consolidation. *ACM Trans. Softw. Eng. Methodol.*, 22(2):11:1–11:42, 2013. doi: 10.1145/2430545.2430547. URL <https://doi.org/10.1145/2430545.2430547>. (Cited on page 92.)
- [103] Michael Rosemann. Potential pitfalls of process modeling: part A. *Bus. Process. Manag. J.*, 12(2):249–254, 2006. doi: 10.1108/14637150610657567. URL <https://doi.org/10.1108/14637150610657567>. (Cited on page 92.)
- [104] Alexandra Rostin, Oliver Albrecht, Jana Bauckmann, Felix Naumann, and Ulf Leser. A machine learning approach to foreign key discovery. In *12th International Workshop on the Web and Databases, WebDB 2009, Providence, Rhode Island, USA, June 28, 2009*, 2009. URL http://webdb09.cse.buffalo.edu/papers/Paper30/rostin_et_al_final.pdf. (Cited on page 30.)
- [105] Alexandra Rostin, Oliver Albrecht, Jana Bauckmann, Felix Naumann, and Ulf Leser. A machine learning approach to foreign key discovery. In *12th International Workshop on the Web and Databases, WebDB 2009, Providence, Rhode Island, USA, June 28, 2009*, 2009. URL http://webdb09.cse.buffalo.edu/papers/Paper30/rostin_et_al_final.pdf. (Cited on page 31.)
- [106] Anne Rozinat and Wil M. P. van der Aalst. Decision mining in prom. In Schahram Dustdar, José Luiz Fiadeiro, and Amit P. Sheth, editors, *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings*, volume 4102 of *Lecture Notes in Computer Science*, pages 420–425. Springer, 2006. doi: 10.1007/11841760_33. URL https://doi.org/10.1007/11841760_33. (Cited on page 80.)

- [107] Anne Rozinat and Wil M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Inf. Syst.*, 33(1):64–95, 2008. doi: 10.1016/j.is.2007.07.001. URL <https://doi.org/10.1016/j.is.2007.07.001>. (Cited on pages 1 and 21.)
- [108] James E. Rumbaugh, Ivar Jacobson, and Grady Booch. *The unified modeling language reference manual*. Addison-Wesley-Longman, 1999. ISBN 978-0-201-30998-0. (Cited on pages 2, 17, 46, 116, and 117.)
- [109] Tomer Sagi and Avigdor Gal. Schema matching prediction with applications to data source discovery and dynamic ensembling. *VLDB J.*, 22(5):689–710, 2013. doi: 10.1007/s00778-013-0325-y. URL <https://doi.org/10.1007/s00778-013-0325-y>. (Cited on page 117.)
- [110] Mohammadreza Fani Sani, Sebastiaan J. van Zelst, and Wil M. P. van der Aalst. Conformance checking approximation using subset selection and edit distance. In Schahram Dustdar, Eric Yu, Camille Salinesi, Dominique Rieu, and Vik Pant, editors, *Advanced Information Systems Engineering - 32nd International Conference, CAiSE 2020, Grenoble, France, June 8-12, 2020, Proceedings*, volume 12127 of *Lecture Notes in Computer Science*, pages 234–251. Springer, 2020. doi: 10.1007/978-3-030-49435-3_15. URL https://doi.org/10.1007/978-3-030-49435-3_15. (Cited on page 21.)
- [111] Stefan Schönig, Claudio Di Ciccio, Fabrizio Maria Maggi, and Jan Mendling. Discovery of multi-perspective declarative process models. In Quan Z. Sheng, Eleni Stroulia, Samir Tata, and Sami Bhiri, editors, *Service-Oriented Computing - 14th International Conference, IC-SOC 2016, Banff, AB, Canada, October 10-13, 2016, Proceedings*, volume 9936 of *Lecture Notes in Computer Science*, pages 87–103. Springer, 2016. doi: 10.1007/978-3-319-46295-0_6. URL https://doi.org/10.1007/978-3-319-46295-0_6. (Cited on page 68.)
- [112] Arik Senderovich, Andreas Rogge-Solti, Avigdor Gal, Jan Mendling, and Avishai Mandelbaum. The ROAD from sensor data to process instances via interaction mining. In Selmin Nurcan, Prina Soffer, Marko Bajec, and Johann Eder, editors, *Advanced Information Systems Engineering - 28th International Conference, CAiSE 2016, Ljubljana, Slovenia, June 13-17, 2016. Proceedings*, volume 9694 of *Lecture Notes in Computer Science*, pages 257–273. Springer, 2016. doi: 10.1007/978-3-319-39696-5_16. URL https://doi.org/10.1007/978-3-319-39696-5_16. (Cited on page 64.)
- [113] Roei Shraga, Avigdor Gal, and Haggai Roitman. Adnev: Cross-domain schema matching using deep similarity matrix adjustment and evaluation. *Proc. VLDB Endow.*, 13(9):1401–1415, 2020. doi: 10.14778/3397230.3397237. URL <http://www.vldb.org/pvldb/vol13/p1401-shraga.pdf>. (Cited on page 117.)
- [114] Sebastian Steinau, Andrea Marrella, Kevin Andrews, Francesco Leotta, Massimo Mecella, and Manfred Reichert. DALEC: a framework for the systematic evaluation of data-centric approaches to process management software. *Softw. Syst. Model.*, 18(4):2679–2716, 2019. doi: 10.1007/s10270-018-0695-0. URL <https://doi.org/10.1007/s10270-018-0695-0>. (Cited on page 46.)

- [115] Sherry X. Sun, J. Leon Zhao, Jay F. Nunamaker Jr., and Olivia R. Liu Sheng. Formulating the data-flow perspective for business process management. *Inf. Syst. Res.*, 17(4):374–391, 2006. doi: 10.1287/isre.1060.0105. URL <https://doi.org/10.1287/isre.1060.0105>. (Cited on page 80.)
- [116] Niek Tax, Natalia Sidorova, Wil M. P. van der Aalst, and Reinder Haakma. Heuristic approaches for generating local process models through log projections. In *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016, Athens, Greece, December 6-9, 2016*, pages 1–8. IEEE, 2016. doi: 10.1109/SSCI.2016.7849948. URL <https://doi.org/10.1109/SSCI.2016.7849948>. (Cited on pages 1, 20, and 68.)
- [117] Nikola Trcka, Wil M. P. van der Aalst, and Natalia Sidorova. Data-flow anti-patterns: Discovering data-flow errors in workflows. In Pascal van Eck, Jaap Gordijn, and Roel J. Wieringa, editors, *Advanced Information Systems Engineering, 21st International Conference, CAiSE 2009, Amsterdam, The Netherlands, June 8-12, 2009. Proceedings*, volume 5565 of *Lecture Notes in Computer Science*, pages 425–439. Springer, 2009. doi: 10.1007/978-3-642-02144-2_34. URL https://doi.org/10.1007/978-3-642-02144-2_34. (Cited on page 68.)
- [118] Reina Uba, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. Clone detection in repositories of business process models. In Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf, editors, *Business Process Management - 9th International Conference, BPM 2011, Clermont-Ferrand, France, August 30 - September 2, 2011. Proceedings*, volume 6896 of *Lecture Notes in Computer Science*, pages 248–264. Springer, 2011. doi: 10.1007/978-3-642-23059-2_20. URL https://doi.org/10.1007/978-3-642-23059-2_20. (Cited on pages 76 and 104.)
- [119] Álvaro Valencia-Parra, Belén Ramos-Gutiérrez, Angel Jesus Varela-Vaca, María Teresa Gómez López, and Antonio Garcia Bernal. Enabling process mining in aircraft manufactures: extracting event logs and discovering processes from complex data. In Jan vom Brocke, Jan Mendling, and Michael Rosemann, editors, *Proceedings of the Industry Forum at BPM 2019 co-located with 17th International Conference on Business Process Management (BPM 2019), Vienna, Austria, September 1-6, 2019*, volume 2428 of *CEUR Workshop Proceedings*, pages 166–177. CEUR-WS.org, 2019. URL <http://ceur-ws.org/Vol-2428/paper15.pdf>. (Cited on page 26.)
- [120] Álvaro Valencia-Parra, Ángel Jesús Varela-Vaca, María Teresa Gómez López, and Paolo Ceravolo. CHAMALEON: framework to improve data wrangling with complex data. In Helmut Krcmar, Jane Fedorowicz, Wai Fong Boh, Jan Marco Leimeister, and Sunil Wattal, editors, *Proceedings of the 40th International Conference on Information Systems, ICIS 2019, Munich, Germany, December 15-18, 2019*. Association for Information Systems, 2019. URL https://aisel.aisnet.org/icis2019/data_science/data_science/16. (Cited on page 80.)
- [121] W. M. P. van der Aalst. *Three Good Reasons for Using a Petri-Net-Based Workflow Management System*, pages 161–182. Springer US, Boston, MA, 1998. ISBN 978-1-4615-5499-8. doi: 10.1007/978-1-4615-5499-8_10. URL https://doi.org/10.1007/978-1-4615-5499-8_10. (Cited on page 13.)

- [122] Wil M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag, Berlin, 2011. (Cited on pages 4 and 20.)
- [123] Wil M. P. van der Aalst. Extracting event data from databases to unleash process mining. In Jan vom Brocke and Theresa Schmiedel, editors, *BPM - Driving Innovation in a Digital World*, pages 105–128. Springer, 2015. doi: 10.1007/978-3-319-14430-6_8. URL https://doi.org/10.1007/978-3-319-14430-6_8. (Cited on pages 26, 27, and 36.)
- [124] Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition*. Springer, 2016. ISBN 978-3-662-49850-7. doi: 10.1007/978-3-662-49851-4. URL <https://doi.org/10.1007/978-3-662-49851-4>. (Cited on pages 1, 19, 21, 22, and 81.)
- [125] Wil M. P. van der Aalst. Object-centric process mining: Dealing with divergence and convergence in event data. In Peter Csaba Ölveczky and Gwen Salaün, editors, *Software Engineering and Formal Methods - 17th International Conference, SEFM 2019, Oslo, Norway, September 18-20, 2019, Proceedings*, volume 11724 of *Lecture Notes in Computer Science*, pages 3–25. Springer, 2019. doi: 10.1007/978-3-030-30446-1_1. URL https://doi.org/10.1007/978-3-030-30446-1_1. (Cited on page 46.)
- [126] Wil M. P. van der Aalst. A practitioner’s guide to process mining: Limitations of the directly-follows graph. In Maria Manuela Cruz-Cunha, Ricardo Martinho, Rui Rijo, Emanuel Peres, and Dulce Domingos, editors, *CENTERIS 2019 - International Conference on ENTERprise Information Systems / ProjMAN 2019 - International Conference on Project MANagement / HCist 2019 - International Conference on Health and Social Care Information Systems and Technologies 2019, Sousse, Tunisia*, volume 164 of *Procedia Computer Science*, pages 321–328. Elsevier, 2019. doi: 10.1016/j.procs.2019.12.189. URL <https://doi.org/10.1016/j.procs.2019.12.189>. (Cited on page 72.)
- [127] Wil M. P. van der Aalst. A practitioner’s guide to process mining: Limitations of the directly-follows graph. In Maria Manuela Cruz-Cunha, Ricardo Martinho, Rui Rijo, Emanuel Peres, and Dulce Domingos, editors, *CENTERIS 2019 - International Conference on ENTERprise Information Systems / ProjMAN 2019 - International Conference on Project MANagement / HCist 2019 - International Conference on Health and Social Care Information Systems and Technologies 2019, Sousse, Tunisia*, volume 164 of *Procedia Computer Science*, pages 321–328. Elsevier, 2019. doi: 10.1016/j.procs.2019.12.189. URL <https://doi.org/10.1016/j.procs.2019.12.189>. (Cited on page 21.)
- [128] Wil M. P. van der Aalst and A. J. M. M. Weijters. Process mining: a research agenda. *Comput. Ind.*, 53(3):231–244, 2004. doi: 10.1016/j.compind.2003.10.001. URL <https://doi.org/10.1016/j.compind.2003.10.001>. (Cited on page 1.)
- [129] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *WIREs Data Mining Knowl. Discov.*, 2(2):182–192, 2012. doi: 10.1002/widm.1045. URL <https://doi.org/10.1002/widm.1045>. (Cited on page 20.)

- [130] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *WIREs Data Mining Knowl. Discov.*, 2(2):182–192, 2012. doi: 10.1002/widm.1045. URL <https://doi.org/10.1002/widm.1045>. (Cited on pages 1 and 21.)
- [131] Boudewijn van Dongen. BPI Challenge 2017. <https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f>, 2017. URL <https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f>. (Cited on page 105.)
- [132] Boudewijn van Dongen. BPI Challenge 2020. <https://doi.org/10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b5>, 2020. URL <https://doi.org/10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b5>. (Cited on page 105.)
- [133] Boudewijn F. van Dongen, Ana Karla A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and Wil M. P. van der Aalst. The prom framework: A new era in process mining tool support. In Gianfranco Ciardo and Philippe Darondeau, editors, *Applications and Theory of Petri Nets 2005, 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005, Proceedings*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer, 2005. doi: 10.1007/11494744_25. URL https://doi.org/10.1007/11494744_25. (Cited on pages 20 and 80.)
- [134] Boudewijn F. van Dongen, Remco M. Dijkman, and Jan Mendling. Measuring similarity between business process models. In Zohra Bellahsene and Michel Léonard, editors, *Advanced Information Systems Engineering, 20th International Conference, CAiSE 2008, Montpellier, France, June 16-20, 2008, Proceedings*, volume 5074 of *Lecture Notes in Computer Science*, pages 450–464. Springer, 2008. doi: 10.1007/978-3-540-69534-9_34. URL https://doi.org/10.1007/978-3-540-69534-9_34. (Cited on page 76.)
- [135] Maikel L. van Eck, Natalia Sidorova, and Wil M. P. van der Aalst. Enabling process mining on sensor data from smart products. In *Tenth IEEE International Conference on Research Challenges in Information Science, RCIS 2016, Grenoble, France, June 1-3, 2016*, pages 1–12. IEEE, 2016. doi: 10.1109/RCIS.2016.7549355. URL <https://doi.org/10.1109/RCIS.2016.7549355>. (Cited on page 64.)
- [136] H. M. W. Verbeek, Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Xes, xesame, and prom 6. In Pnina Soffer and Erik Proper, editors, *Information Systems Evolution - CAiSE Forum 2010, Hammamet, Tunisia, June 7-9, 2010, Selected Extended Papers*, volume 72 of *Lecture Notes in Business Information Processing*, pages 60–75. Springer, 2010. doi: 10.1007/978-3-642-17722-4_5. URL https://doi.org/10.1007/978-3-642-17722-4_5. (Cited on page 80.)
- [137] Jianmin Wang, Tao Jin, Raymond K. Wong, and Lijie Wen. Querying business process model repositories - A survey of current approaches and issues. *World Wide Web*, 17(3):427–454, 2014. doi: 10.1007/s11280-013-0210-z. URL <https://doi.org/10.1007/s11280-013-0210-z>. (Cited on page 92.)

- [138] Matthias Weidlich, Artem Polyvyanyy, Nirmal Desai, Jan Mendling, and Mathias Weske. Process compliance analysis based on behavioural profiles. *Inf. Syst.*, 36(7):1009–1025, 2011. doi: 10.1016/j.is.2011.04.002. URL <https://doi.org/10.1016/j.is.2011.04.002>. (Cited on page 1.)
- [139] Mathias Weske. *Business Process Management - Concepts, Languages, Architectures, Third Edition*. Springer, 2019. ISBN 978-3-662-59431-5. doi: 10.1007/978-3-662-59432-2. URL <https://doi.org/10.1007/978-3-662-59432-2>. (Cited on pages xiii, 1, 4, 11, 12, 13, 14, 56, 92, and 111.)
- [140] Francesca Zerbato, Pnina Soffer, and Barbara Weber. Initial insights into exploratory process mining practices. In Artem Polyvyanyy, Moe Thandar Wynn, Amy Van Looy, and Manfred Reichert, editors, *Business Process Management Forum - BPM Forum 2021, Rome, Italy, September 06-10, 2021, Proceedings*, volume 427 of *Lecture Notes in Business Information Processing*, pages 145–161. Springer, 2021. doi: 10.1007/978-3-030-85440-9_9. URL https://doi.org/10.1007/978-3-030-85440-9_9. (Cited on pages 3 and 117.)
- [141] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M. Procopiuc, and Divesh Srivastava. On multi-column foreign key discovery. *Proc. VLDB Endow.*, 3(1):805–814, 2010. doi: 10.14778/1920841.1920944. URL http://www.vldb.org/pvldb/vldb2010/pvldb_vol3/R72.pdf. (Cited on page 31.)
- [142] Michael zur Muehlen and Michael Rosemann. Multi-paradigm process management. In Janis Grundspenkis and Marite Kirikova, editors, *CAiSE'04 Workshops in connection with The 16th Conference on Advanced Information Systems Engineering, Riga, Latvia, 7-11 June, 2004, Knowledge and Model Driven Information Systems Engineering for Networked Organisations, Proceedings, Vol. 2*, pages 169–175. Faculty of Computer Science and Information Technology, Riga Technical University, Riga, Latvia, 2004. (Cited on page 11.)

All links were last followed on June, 2022.

DECLARATION

I hereby confirm that I have authored this thesis independently and without use of others than the indicated sources. All passages which are literally or in general matter taken out of publications or other sources are marked as such. I am aware of the examination regulations and this thesis has not been previously submitted elsewhere.

Potsdam, August, 2022

Dorina Bano

