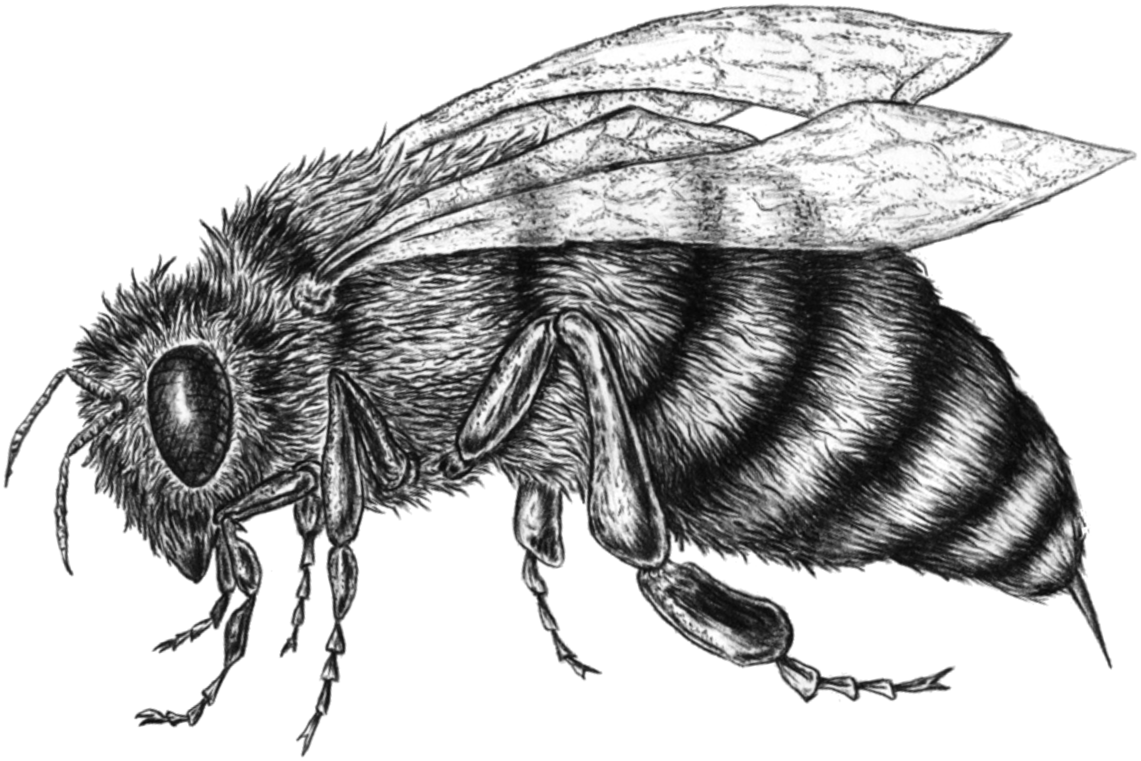


Process Mining on Databases

Extracting Event Data from Real-Life Data Sources

E. González López de Murillas

Process Mining on Databases: Extracting Event Data from Real-Life Data Sources

E. González López de Murillas

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

González López de Murillas, E.

Process Mining on Databases: Extracting Event Data from Real-Life Data Sources
by E. González López de Murillas.
Eindhoven: Technische Universiteit Eindhoven, 2018. Proefschrift.

A catalogue record is available from the Eindhoven University of Technology Library

ISBN 978-90-386-4704-3

Keywords: Process mining, Databases, Data extraction, Case notion discovery, Case notion recommendation, Event log building, Event data querying

SIKS Dissertation Series No. 2019-03
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Process Mining on Databases:
Extracting Event Data from Real-Life Data Sources

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op
woensdag 27 februari 2019 om 16:00 uur

door

Eduardo González López de Murillas

geboren te Pamplona, Spanje

Dedicated to my little nephew, Mateo.

Abstract

Process mining as a discipline is gaining importance in recent years. Its focus is on the analysis of business processes by means of process execution data. It is a common misconception to think that a process mining project starts as soon as the data are available. We tend to underestimate the time, knowledge, and number of iterations needed to gather relevant data and build an event log suitable for analysis. In many cases, the data extraction and preparation phase can take up to 80% of the project duration. Improving this time-consuming initial phase will have a large impact on the whole process mining project in terms of time, cost, and quality of insights.

In this thesis, we take on several challenges related to data extraction, multi-perspective event log building, and event data querying. We consider environments in which many business processes coexist, sharing data both in a centralized and distributed storage. The aim of this work is twofold. First, to ease the path for practitioners to get started with a process mining project in unknown environments when domain knowledge is not always available. Second, to provide the tools needed for event data querying and to build specialized event logs for a more effective process mining analysis. We propose several techniques that, together, constitute a pipeline connecting databases with existing process mining techniques:

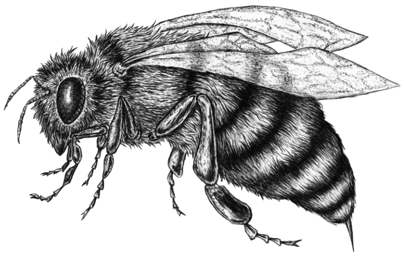
- *A meta-model for process mining on databases* (in Chapter 3). This meta-model aims to standardize the way to capture and structure both data and process aspects of an organization.

- *Data extraction adapters for several scenarios* (in Chapter 4). These adapters, developed for some mainstream scenarios, allow us to extract and transform data from databases to comply with the proposed meta-model structure.

- A technique to *discover and recommend case notions* in order to *build event logs* for process mining (in Chapter 5). This is achieved by exploring possible views on the data, correlating events by means of the underlying relations extracted from the original database, and assessing their "interestingness" with respect to some metrics.

- A data-aware process oriented query language (in Chapter 7). This query language provides a compact and easy way to express relevant queries in the business context by means of constructs that exploit the underlying meta-model.

The techniques presented in this thesis have been evaluated using representative sample datasets that resemble the most relevant challenges that can be found in real-life environments, as well as case studies with real databases. The results show that these techniques succeed at capturing a more comprehensive picture of the systems under study, improving the quality of the generated event logs, and supporting the user in the process of data extraction and log building. Moreover, they are a step ahead towards the goal of automatic data extraction and process mining analysis of complex IT systems. Implementations of our techniques are publicly available and licensed as open source.



Preface

Honeybees are fascinating creatures. These small insects are incredibly cooperative. They organize themselves according to a very well defined societal structure, divided mainly into reproductive and non-reproductive groups. Individually, they would not be able to survive. However, as a team they can build complicated housing structures called hives where they live, breed, store food, and protect themselves from external agents.

These incredible animals do not only produce honey, but also wax, their main construction material. Both products are highly valuable for us humans due to the nutritive value of honey, and the versatility of the wax as a material in many manufacturing processes. However, the shape in which beehives appear in nature does not allow human beekeepers to take advantage of the honey and wax production without partially or entirely destroying the hive.

Nowadays, most organizations produce data as a byproduct of their business activity, similarly to the hard-working bees producing wax and honey for their survival. In many cases, the produced data is crucial for the functioning of the organization, while at the same time it acts as a record of the execution of their business processes. However, many of such organizations struggle to get the maximal potential from their data. One of the reasons is the disconnect between the data structure that supports their business processes and the data structure that would be suitable for the analysis of their process data.

We believe that business process execution and analysis can coexist, sharing common data sources in a sustainable manner. In order to achieve such an equilibrium, data structures need to be reshaped, similarly to the way that beekeepers provide a structure to beehives. When used responsibly, artificial beehives allow the beekeeper to benefit from the production of honey without disrupting the colony. This thesis proposes a way to structure data in order to facilitate data extraction and to enable the analysis of business processes in real-life environments.

Contents

List of Figures

List of Tables

Swarm hanging from a branch.
"Cours complet d'apiculture", Georges de Layens and Gaston Bonnier, 1897

1 Introduction

The goal of process mining is to analyze event data of business processes in order to obtain insights into the behavior and execution of processes, such as the most common paths, deviations, performance measurements, etc. In order to obtain these insights, process mining techniques rely on the existence of event logs.

In reality, however, event logs are not always readily available. In fact, it is almost never the case that event logs are generated and stored in the desired format ready for analysis. In general, obtaining event logs is a very time consuming and not straightforward task that can require the involvement of business analysts, domain experts, IT specialists, and database administrators. In many cases, up to 80% of the time and effort, and 50% of the cost is spent during the data extraction and preparation phases [121]. Far from the common belief, the construction of event logs is not a one-time task. As shown in Figure 1.1, it is a manual and iterative process that starts with a data exploration phase, continues with repeated event log extractions and mining, and ends with the desired event log that contains the necessary features and cases in order to carry out the desired process mining analysis. The variety of systems used in organizations adds difficulty to the event log extraction task, forcing the analysts to develop ad-hoc queries and perform data preprocessing specifically tailored to each situation.

The focus of this thesis is to research methods to extract event data and support the user in the process of generating event logs suitable for process mining analysis. In this chapter, first, we provide an introduction to process mining in Section 1.1. In Section 1.2 we focus on process mining in the context of databases. Section 1.3 presents the challenges to face when dealing with databases in a process mining project. Finally, we list the contributions of this thesis and its structure in Section 1.4.

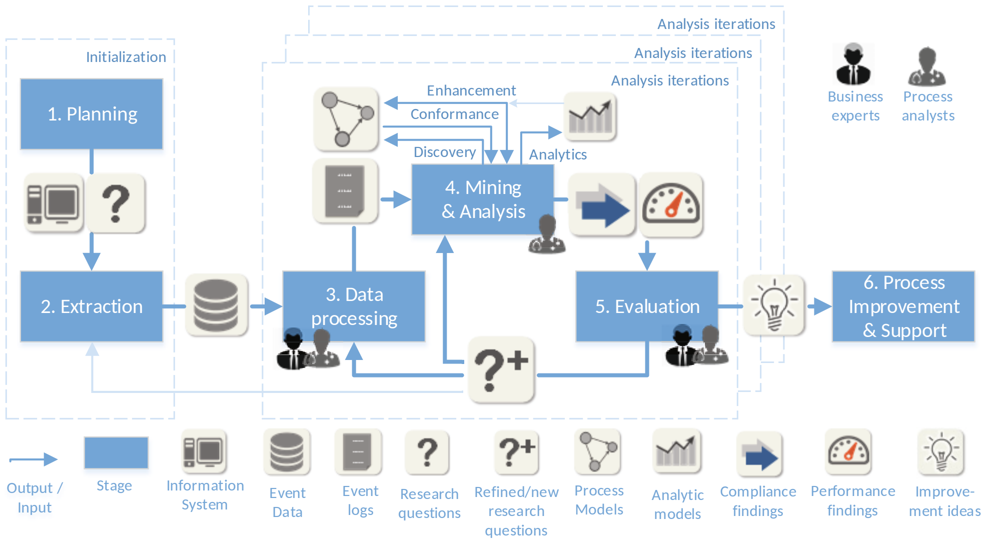


Figure 1.1: An overview of the PM^2 methodology [114].

1.1 Event Data and Process Mining

The field of process mining offers a wide variety of methods to analyze event data. Process discovery, conformance and compliance checking, performance analysis, process monitoring and prediction, and operational support are some of the techniques that process mining provides to better understand and improve business processes. Most of these techniques rely on the existence of an event log. Section 1.1.1 provides details on what event data is and how event logs can be built. Section 1.1.2 gives an introduction to process mining and some of the most common types of analysis that can be performed.

1.1.1 Event Data

Event data is the fuel for process mining. It provides information about the execution of all kinds of processes and enables us to analyze what happened, when, and, sometimes, in what context. This kind of data can be obtained from many sources, such as information system logs, databases, and even paper records. In the scope of this thesis, we will focus on event logs obtained from databases.

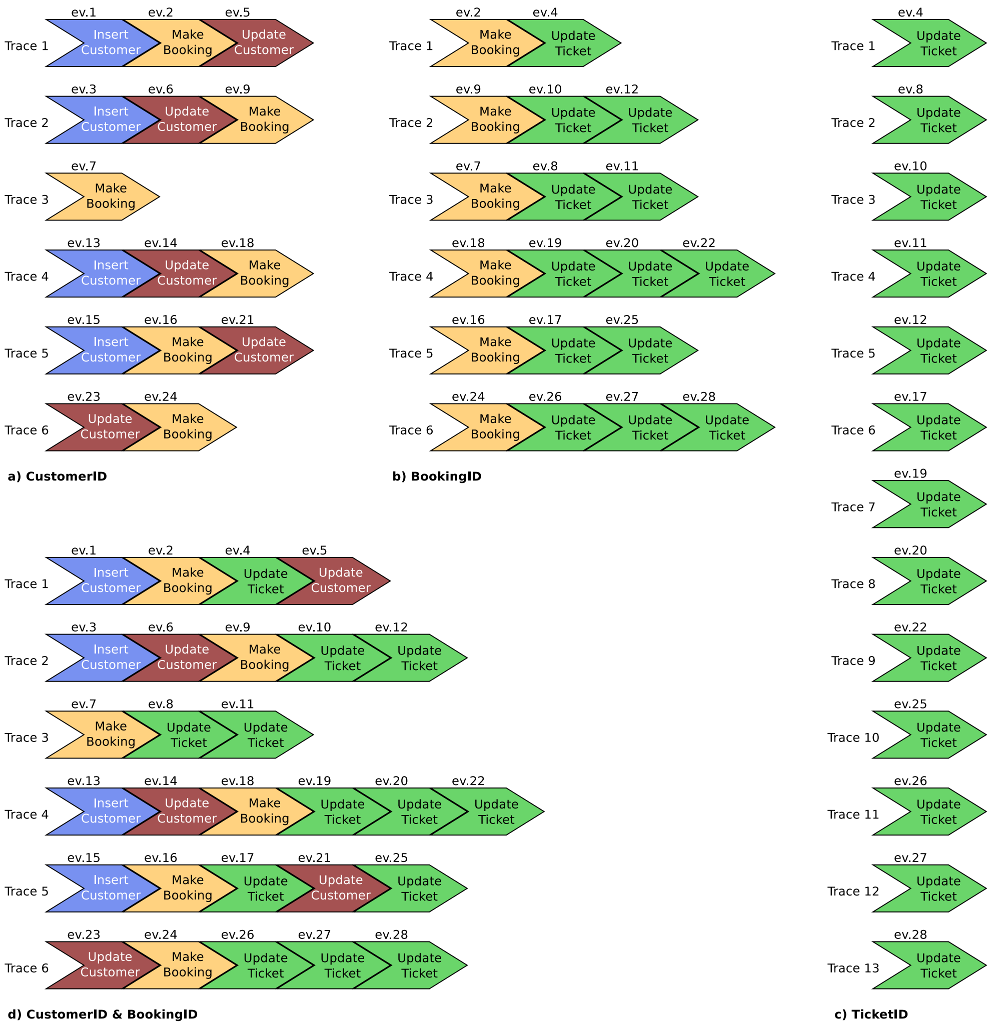
Events are the smallest pieces of event data that we can use to perform a process mining analysis; they represent the occurrence of a specific action. An event is usually characterized by three basic attributes: a timestamp, an activity name, and a case identifier. The timestamp indicates when the event occurred. The activity name gives an indication of what kind of activity was performed. Finally, a case identifier tells us to which instance of the underlying process this event belongs. The timestamps of a collection of events provide some sort of order, while the case identifiers provide

Table 1.1: Example of an event log obtained from the database of a ticket selling system.

Event	Case	CustomerID	BookingID	TicketID	Activity	Time stamps
1	1	101			Insert Customer	2014-11-27 15:55:35
2	1	101	201		Make Booking	2014-11-27 15:58:23
3	2	102			Insert Customer	2014-11-27 15:59:10
4	1		201	039	Update Ticket	2014-11-27 16:03:25
5	1	101			Update Customer	2014-11-27 16:20:13
6	2	102			Update Customer	2014-11-27 16:22:17
7	3	103	203		Make Booking	2014-11-27 17:12:50
8	3		203	055	Update Ticket	2014-11-27 17:15:22
9	2	102	202		Make Booking	2014-11-27 17:23:45
10	2		202	073	Update Ticket	2014-11-27 17:55:15
11	3		203	084	Update Ticket	2014-11-27 18:07:50
12	3		202	065	Update Ticket	2014-11-27 18:13:32
13	4	104			Insert Customer	2014-11-28 10:12:46
14	4	104			Update Customer	2014-11-28 10:20:35
15	5	105			Insert Customer	2014-11-28 11:38:13
16	5	105	205		Make Booking	2014-11-28 12:01:15
17	5		205	061	Update Ticket	2014-11-28 12:02:05
18	4	104	204		Make Booking	2014-11-29 22:40:21
19	4		204	023	Update Ticket	2014-11-29 22:45:12
20	4		204	048	Update Ticket	2014-11-29 23:01:51
21	5	105			Update Customer	2014-11-29 23:05:10
22	4		204	032	Update Ticket	2014-11-29 23:15:28
23	6	106			Update Customer	2014-11-30 13:34:26
24	6	106	206		Make Booking	2014-11-30 13:38:14
25	5		205	017	Update Ticket	2014-11-30 14:45:08
26	6		206	020	Update Ticket	2014-11-30 14:56:42
27	6		206	021	Update Ticket	2014-11-30 15:10:45
28	6		206	022	Update Ticket	2014-11-30 15:25:36
...	...	...	...	...	...	...

a way to correlate events per process instance or trace. Activity names are typically used by miners to discover the activities (or transitions) that belong to the discovered process model.

Table 1.1 shows an example of an event log corresponding to a ticket selling system. We see several events that correspond to actions performed by the system such as customer insertions ("Insert Customer"), customer data changes ("Update Customer"), bookings ("Make Booking"), and ticket reservations ("Update Ticket"). Also, we see that each event has a corresponding timestamp. However, when it comes to case identifiers, several options are left open. Figure 1.2 shows four different event logs that can be built using the events in Table 1.1 when considering different case notions. When customer identifiers (*CustomerID*) are used to correlate events, we obtain the traces in Figure 1.2.a where each trace contains events belonging to the same customer. Another option is to group the events based on the booking they refer to (*BookingID*) resulting in the event log in Figure 1.2.b. Considering the ticket identifier (*TicketID*) as the case notion we obtain the event log shown in Figure 1.2.c where every trace refers to a unique ticket update. However, combining customer and booking identifiers provides a more comprehensive view of the process, resulting in the event log depicted in Figure 1.2.d. Depending on which field is chosen, different event logs will be constructed and, therefore, different process models will be discovered.



a) CustomerID
Trace 1: ev.1 Insert Customer, ev.2 Make Booking, ev.5 Update Customer
Trace 2: ev.3 Insert Customer, ev.6 Update Customer, ev.9 Make Booking
Trace 3: ev.7 Make Booking
Trace 4: ev.13 Insert Customer, ev.14 Update Customer, ev.18 Make Booking
Trace 5: ev.15 Insert Customer, ev.16 Make Booking, ev.21 Update Customer
Trace 6: ev.23 Update Customer, ev.24 Make Booking
b) BookingID
Trace 1: ev.2 Make Booking, ev.4 Update Ticket
Trace 2: ev.9 Make Booking, ev.10 Update Ticket, ev.12 Update Ticket
Trace 3: ev.7 Make Booking, ev.8 Update Ticket, ev.11 Update Ticket
Trace 4: ev.18 Make Booking, ev.19 Update Ticket, ev.20 Update Ticket, ev.22 Update Ticket
Trace 5: ev.16 Make Booking, ev.17 Update Ticket, ev.25 Update Ticket
Trace 6: ev.24 Make Booking, ev.26 Update Ticket, ev.27 Update Ticket, ev.28 Update Ticket
c) TicketID
Trace 1: ev.4 Update Ticket
Trace 2: ev.8 Update Ticket
Trace 3: ev.10 Update Ticket
Trace 4: ev.11 Update Ticket
Trace 5: ev.12 Update Ticket
Trace 6: ev.17 Update Ticket
Trace 7: ev.19 Update Ticket
Trace 8: ev.20 Update Ticket
Trace 9: ev.22 Update Ticket
Trace 10: ev.25 Update Ticket
Trace 11: ev.26 Update Ticket
Trace 12: ev.27 Update Ticket
Trace 13: ev.28 Update Ticket
d) CustomerID & BookingID
Trace 1: ev.1 Insert Customer, ev.2 Make Booking, ev.4 Update Ticket, ev.5 Update Customer
Trace 2: ev.3 Insert Customer, ev.6 Update Customer, ev.9 Make Booking, ev.10 Update Ticket, ev.12 Update Ticket
Trace 3: ev.7 Make Booking, ev.8 Update Ticket, ev.11 Update Ticket
Trace 4: ev.13 Insert Customer, ev.14 Update Customer, ev.18 Make Booking, ev.19 Update Ticket, ev.20 Update Ticket, ev.22 Update Ticket
Trace 5: ev.15 Insert Customer, ev.16 Make Booking, ev.17 Update Ticket, ev.21 Update Customer, ev.25 Update Ticket
Trace 6: ev.23 Update Customer, ev.24 Make Booking, ev.26 Update Ticket, ev.27 Update Ticket, ev.28 Update Ticket

Figure 1.2: Different event logs that can be built depending on the selected case notion.

1.1.2 Process Mining

The aim of process mining [109] is to analyze event data in order to provide insights into the execution of processes in reality. Many techniques can be used, depending on the insights we want to obtain and the questions we need to answer. Process mining techniques can be classified according to their relation to three main tasks: process discovery, conformance checking, and process enhancement.

Process discovery

Process discovery techniques focus on discovering process models that describe the behavior of systems as has been observed in event data. Different algorithms use different representations, each of them having different strengths. Figure 1.3 shows the result of constructing the directly-follows graph based on the 6 traces in Table 1.1. We see patterns like the repetition of “Update Ticket” several times per trace and the fact that most of the times it is preceded by the execution of “Make Booking”.

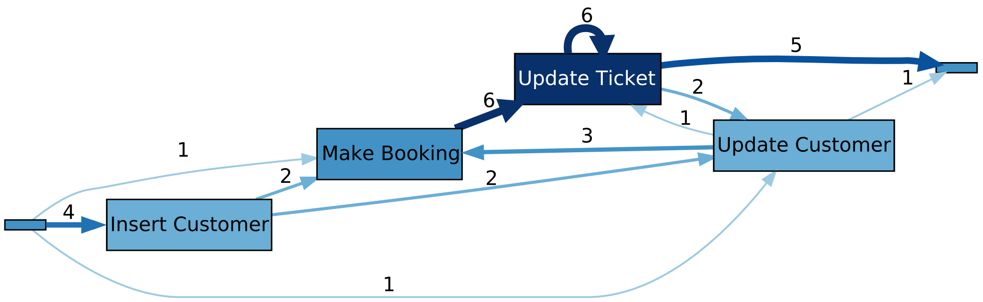


Figure 1.3: Directly-follows graph based on the event log in Table 1.1.

However, the previous model does not provide clear execution semantics, since there is no information about whether splits in the model are actually choices or parallel execution paths. Other algorithms are able to discover these constructs and provide a better understanding of the behavior of the process. Figure 1.4 shows a process tree discovered by the Inductive Miner [62]. There, we see that “Update Ticket” is executed in parallel with “Update Customer” and “Make Booking”. This means that these activities can be executed in any order.

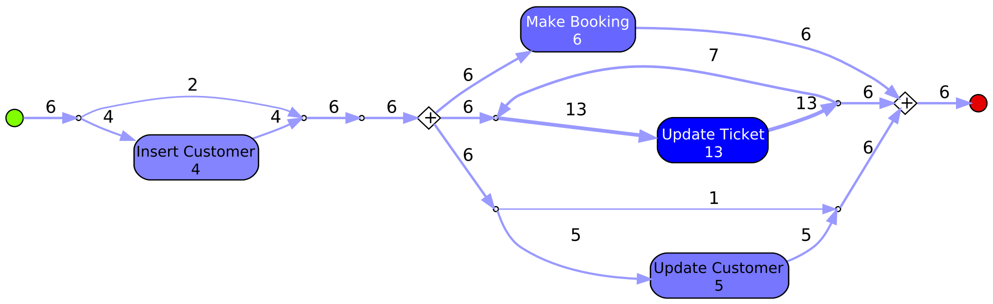


Figure 1.4: Process tree discovered with Inductive Miner [62] based on the event log in Table 1.1.

A discovered process model can be useful on its own, as a representation of a process based on the observed event data. Also, it can be used, together with event data, as the input for other process mining tasks, such as conformance checking or process enhancement.

Conformance checking

Conformance and compliance checking deals with the problem of comparing the expected behavior with the actual observed behavior of a process. It is rarely the case that the observed behavior of processes in reality exactly matches the expected behavior. This can be due to many causes: human error, flexible processes, exceptional cases not considered by the initial design, noise, fraud, etc.

One of the challenges when tackling the conformance and compliance problem is to correctly relate event data to the process model used as a guideline. Another challenge is to provide meaningful diagnostics to the user by means of reliable metrics. Alignment computation [111] is a method that deals with this problem, providing insights into the differences between observed behavior and reference process models.

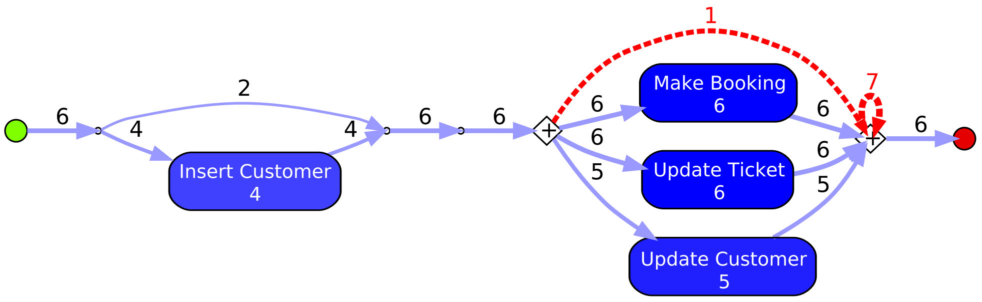


Figure 1.5: Deviations displayed on top of a process tree (discovered with Inductive Miner [62]) based on the event log in Table 1.1.

Figure 1.5 shows a model discovered with the Inductive Miner, on top of which deviations have been plotted. These deviations, represented as red dashed arrows, show alternative paths taken during the execution of the process (as observed in the event data) that were not possible in the reference process model. The first read arc at the top represents one alternative path, initially not supported by the model, that was taken in the event log by skipping the execution of the activities *Make Booking*, *Update Ticket*, and *Update Customer*. The second red arc (with a frequency of 7), represents the execution of activities in the event log that were not supported by the model at that point. The detection of these alternative paths is possible thanks to the application of conformance checking, which is able to match elements of the process model to the event data and identifies the observed behavior that cannot be described by the reference model.

Process enhancement

Process enhancement methods use event data to provide additional diagnostics to improve existing process models. Their aim is to change or extend a process model. For instance, process enhancement can be used to repair a reference model in order to capture observed behavior that was not originally allowed. Figure 1.5 shows an example of this, enabling alternative paths that the original model did not consider. Another use of process enhancement is the representation of performance diagnostics on top of a model. Figure 1.6 shows performance metrics (sojourn time) for a discovered process model based on recorded event data. Using different metrics allows the identification of bottlenecks, throughput times, frequencies, etc.

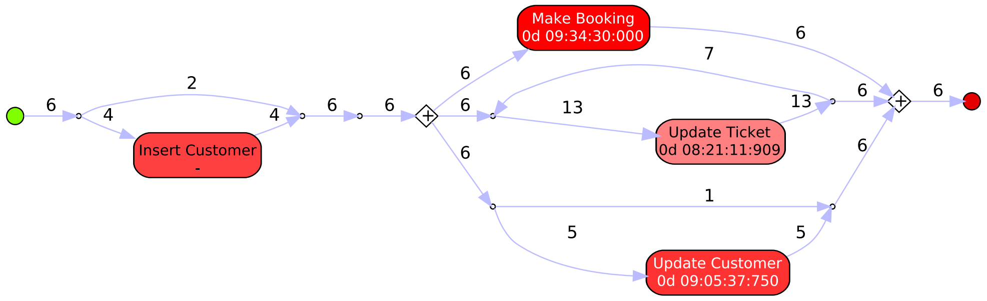


Figure 1.6: Performance metrics (sojourn time) displayed on top of a process tree (discovered with Inductive Miner [62]) based on the event log in Table 1.1.

1.2 Process Mining on Databases

Process mining analysis, when applied on databases, pursues the same goal as process mining on any other environment. However, there are several characteristics that make this environment particularly challenging and promising at the same time:

- Normally, **several processes coexist** in the scope of a single database. One possibility is that these processes are disjoint in terms of the events that relate to them (Figure 1.12). It can also be the case that they are interconnected and share some data aspect, which can be analyzed together (Figure 1.10). Another option is that each process represents a different view on the same data (Figure 1.11).

- The notion of **event log** and **trace** is **not strictly defined**, especially if the system under study is not process-aware. Data objects and implicit events are stored in a database, but it is rare to find them explicitly correlated in traces (Figure 1.7). The analyst is free to choose how to correlate them to build event logs (Figures 1.8 and 1.9). Event correlation in this context is enabled by data relations implicit in the database schema, but the challenge is to decide which of the multiple relations to take into account. Also, many case notions can be chosen, each one representing a different view on the data. Table 1.1 shows an example of a sparse set of events where, in order to be able to correlate all the events belonging to a full process instance, we need to consider the transitive relations between tickets and customers by means of bookings. This is an additional challenge present in most database contexts.

- **Event data** might **not** be **explicitly recorded**, and needs to be extracted and interpreted before it can be analyzed. This is the case when timestamps, activity names, resource, and life-cycle data appeared scattered through the data model. We find this problem too when events are not recorded in the tables, but stored by the RDBMS as a transaction log (or redo-log) at a lower level, in binary or plain files.

- *Data incompleteness* is a common issue when dealing with real-life datasets. This issue becomes critical when important data attributes are missing. This is the case when we find timestamps that indicate the occurrence of an event, without knowing what is the corresponding activity name. Also, timestamps can be missing, either always or sometimes, which represents another example of event data incompleteness.

- It is often the case that **data relations** are **not explicitly defined** in the database schema, but the constraints are being kept at the application level. If the application is proprietary and documentation is not available, we need to discover these relations before the events can be correlated.

- There is a **huge variety of database vendors** and systems. Each of them uses different technologies and records data in a different way. Despite the existence of SQL as the de facto standard among query languages, each database system implements a dialect of SQL, adding new commands or introducing syntax variations with respect to the SQL specification.

All these aspects make the application of process mining on databases a challenging but potentially rewarding task. The fact that, every day, almost every organization stores large amounts of data about their business process instances in a structured manner in databases, represents an extraordinary opportunity to bring process mining to practice at a large scale. This thesis focuses on supporting the process of event data extraction, event log building, and event data querying, based on the first two guiding principles (GP1 and GP2) indicated in [110]. We follow these two guiding principles to tackle the challenges faced when applying process mining on databases. The next section explores these challenges more exhaustively.

1.3 Challenges in Process Mining on Databases

Over the last years, we have witnessed an increase of interest in the field of process mining. Important academic developments took place, while at the same time many

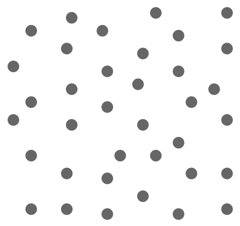


Figure 1.7: Space of events obtained from a data store.

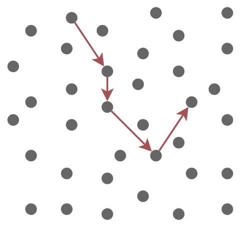


Figure 1.8: A case as a path through the space of events.

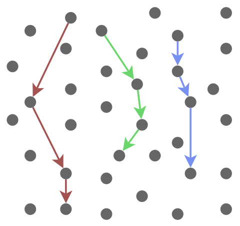


Figure 1.9: A process as a collection of cases.

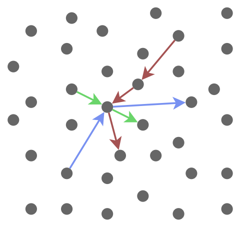


Figure 1.10: The same event can belong to several cases of the same or different processes.

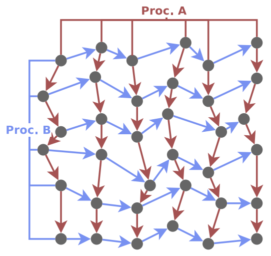


Figure 1.11: Processes as different views on the same set of events.

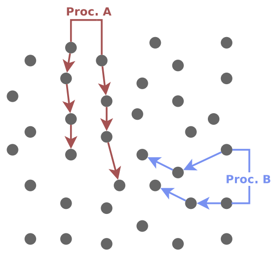


Figure 1.12: Processes describing disjoint events.

companies became aware of the business potential of process mining. As a result, a number of commercial tools incorporating process mining capabilities became available to the market.

Despite these developments, several challenges remain an open question. An example is the case of the data extraction and preparation phase. Nowadays, obtaining event data and getting the right event logs is still one of the most time-consuming tasks in any process mining project. The process mining manifesto [110], published by the IEEE Task Force on Process Mining, presents several guiding principles and challenges in the field. This section lists some challenges relevant in the scope of this thesis, and proposes a few additional ones of particular importance in database environments. All of these challenges are related to the problems of event data extraction, event log building, and event data querying, as depicted in Figure 1.13.

1.3.1 Challenge 1: Finding, Merging, and Cleaning Event Data

This challenge corresponds to Challenge C1 as mentioned in [110]. According to C1, extracting event data in a form suitable for process mining represents a considerable effort. Some issues to face when extracting event data are:

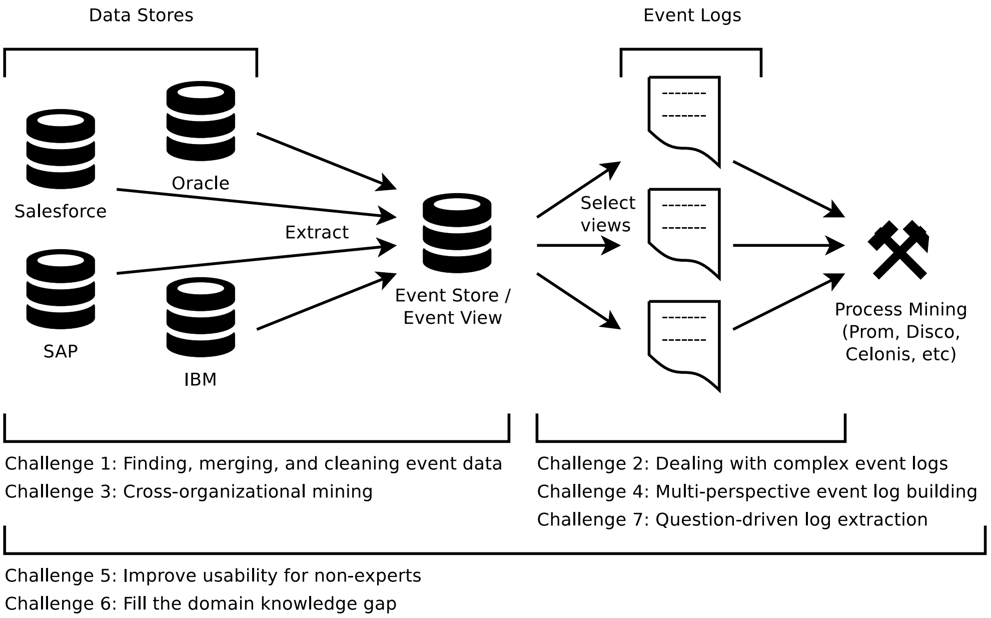

Data Stores
Event Logs
Oracle
Salesforce
Extract
SAP
IBM
Event Store / Event View
Select views
Process Mining (Prom, Disco, Celonis, etc)
Challenge 1: Finding, merging, and cleaning event data
Challenge 3: Cross-organizational mining
Challenge 2: Dealing with complex event logs
Challenge 4: Multi-perspective event log building
Challenge 7: Question-driven log extraction
Challenge 5: Improve usability for non-experts
Challenge 6: Fill the domain knowledge gap

Figure 1.13: Overview of the challenges when connecting data stores with process mining techniques.

- **Distribution** of data across various sources: the information needs to be retrieved and merged in order to be analyzed.
- **Nature** of data: usually, data found in databases are “object centric” instead of “process centric”. However, processes usually involve events that belong to objects of different classes.
- **Incompleteness** of event data: this makes it difficult to directly relate events to a process instance. Event timestamps may be missing as well. Data interpolation and inference may be necessary to derive the missing pieces of information.
- **Outliers** in data: extracted event logs may contain exceptional behavior. How to identify this behavior and classifying it either as noise or exceptions worth of further analysis is a question to be answered.
- **Granularity** of events: very low-level events are often too detailed for the purpose of the analysis, while very high-level events hide interesting and complex behavior. Very fine-grained timestamps can be mixed with coarse date information as well, complicating the ordering of events and, therefore, the analysis.
- **Context** of event occurrence: this context may help to explain the behavior observed in data. If not taken into account at the data extraction phase, important details can be missing. However, incorporating excessive contextual information can complicate the analysis, introducing too many features.

These problems need to be addressed by new tools or techniques, aimed at obtaining high-quality event logs. This can only be achieved by *"treating event data as first-class citizens"* (Guiding Principle GP1 [110]).

1.3.2 Challenge 2: Dealing with Complex Event Logs Having Diverse Characteristics

This challenge corresponds to Challenge C2 as mentioned in [110]. C2 states that event logs may have very different characteristics in terms of size, quality, granularity, structure, etc. These aspects contribute to the difficulty of the analysis. Very large event logs may pose scalability issues, while very small event logs do not provide enough evidence to obtain reliable results. Also, the structure plays an important role in the complexity of an event log, independently of its size. High variability in structure among the traces contributes to more unstructured process models, more difficult to interpret than other logs in which most of the cases follow the same path. So far, organizations deal with these issues by investing a considerable amount of time and work to identify event logs suitable for process mining analysis.

1.3.3 Challenge 3: Cross-Organizational Mining

This challenge corresponds to Challenge C7 as mentioned in [110]. C7 states that it is common to find processes that span across several organizations. It is often the case that each participant handles a part of the whole process at hand. Event logs may become available for analysis from each organization involved. However, focusing on each of these logs individually does not help to understand the process as a whole. The analysis of end-to-end cross-organizational processes requires event logs from different sources to be merged. Correlating events across organizations is not a trivial task and still represents an open challenge to apply process mining in this kind of scenario.

1.3.4 Challenge 4: Multi-Perspective Event Log Building

Modern organizations are driven by many processes. Some of these processes can be completely disjoint (procurement vs. salaries), while others can intersect in some aspects (procurement vs. invoicing). In the latter case, the extracted event data may belong to several processes and, in order to analyze each one separately, we need to adopt the right perspective. Choosing one or another perspective requires a different way to correlate the events to build event logs. Organizations achieve this by means of trial and error, going back to the original source of data and rebuilding logs from different perspectives by manually specifying correlation rules and executing complicated queries. It is not trivial to define a general approach that facilitates multi-perspective log building and enables process mining techniques to show common points of intersection between coexisting processes.

1.3.5 Challenge 5: Improve Usability for Non-Experts

This challenge corresponds to Challenge C10 as mentioned in [110]. According to C10, one of the goals of process mining is to provide “living process models”. These models must hold a link with event data, so new data can be used to discover new behavior, and to remain up-to-date with the current state. All this requires easy-to-use, intuitive interfaces and automatic methods that hide the complexity from the user. When it comes to databases, usability is important due to the vast amount of information to be analyzed by users. Additionally, the lack of homogeneity in terms of structure complicates the development of tools to automatically extract and process the data.

It is necessary to develop new techniques that focus on the automation of data extraction and log building, assisting the user on this tedious task, and requiring as little intervention and manual work as possible.

1.3.6 Challenge 6: Fill the Domain Knowledge Gap in Event Log Extraction

Nowadays, event log extraction from databases is driven mainly by domain knowledge. This means looking at specific attributes of the data to extract events and correlating them in order to build event logs. In the cases when such knowledge is not available, it is necessary to use intuition or explorative methods to identify event information and case notions. Some of these explorative methods require a lot of manual work. This problem becomes intractable when tackling databases that involve tens, hundreds, or thousands of tables. In these cases, manual explorative approaches are inefficient, due to the huge cost in time, and ineffective, given that important data can be missed. New approaches are needed to fill the domain knowledge gap in database environments, in order to enable automatic extraction of data with some guarantees.

1.3.7 Challenge 7: Question-Driven Log Extraction

Event log extraction is not a trivial task, especially in the context of databases, where many processes and data perspectives can be mixed in a single and complex data schema. Extracting and correlating all the existing events as a whole often results in extremely complex event logs, where behavioral patterns are mixed into a meaningless and difficult to interpret “spaghetti” process. To avoid this, during the data extraction phase, we should focus on particular perspectives, cases, or tables, in order to answer concrete questions. This falls in line with the Guiding Principle GP2 [110], which says that *“log extraction should be driven by questions”*.

Without concrete questions, it becomes very hard to obtain meaningful event data, especially within a database context where hundreds or thousands of tables can be involved. It is necessary to develop techniques able to query data in order to select the relevant events for a more effective analysis and more meaningful results.

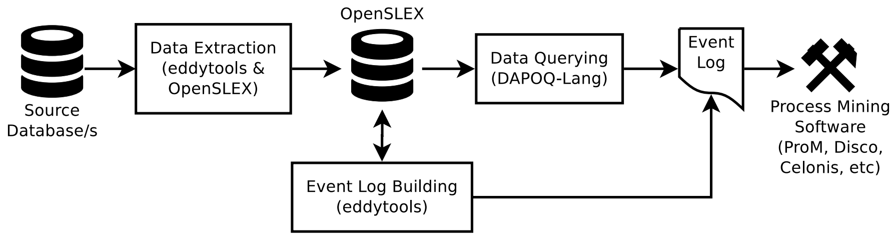

OpenSLEX
Source Database/s
Data Extraction (eddytools & OpenSLEX)
Data Querying (DAPOQ-Lang)
Event Log
Process Mining Software (ProM, Disco, Celonis, etc)
Event Log Building (eddytools)

Figure 1.14: Overview of the contributions of this thesis, in the context of a pipeline connecting databases with existing process mining techniques.

1.4 Contributions and Structure of this Thesis

In this section, we list the main contributions of this thesis and provide an outline of the chapters with a short summary of their content.

1.4.1 Thesis Contributions

The contributions of this thesis can be grouped into three blocks according to the phase of a process mining project to which they relate: **data extraction**, **event log building**, and **data querying**. Figure 1.14 shows a diagram of the pipeline connecting the source databases to the existing process mining techniques. When compared to the situation depicted in Figure 1.13, we see that contributions have been made at several stages of the process. Contributions at the data extraction level focus on the definition of a standard target format and on supporting the extraction from different sources. With respect to event log building, contributions focus on assisting event log generation from different perspectives. Finally, contributions on data querying aim at supporting analysts to analyze data in a process mining context, e.g., to obtain event logs that fit specific business questions. More details about the contributions are provided below:

Data Extraction

The data extraction phase consists of the process of accessing, extracting, and transforming the data to a standard format, i.e., *all the operations to be performed before we can start making sense out of the data.* With respect to this phase, we present the following contributions:

- *A meta-model for process mining on databases (OpenSLEX)* (Chapter 3, addresses challenges 1.3.1 and 1.3.2). This meta-model combines data and process aspects in a single structure in order to capture a more comprehensive picture of the system under study. At the same time, it serves as a standard format to decouple data extraction from later phases like event log building, data querying, or process mining analysis.

- *Data extraction adapters for several environments* (Chapter 4, addresses challenges 1.3.1, 1.3.2, and 1.3.3). The meta-model mentioned above is tested on three different environments, presenting the mapping to the proposed meta-model both formally and in practice. Domain knowledge about the architecture of the source data is used during the data extraction phase in order to adapt it to the proposed meta-model.

Event Log Building

The event log building phase deals with the issue of correlating events in order to build traces and logs. This is an important phase, since there are many criteria to group events together. A case notion defines the specific criteria to follow in order to correlate two events together in the same trace. To help in carrying out this task, we present the following contributions:

- *Case notion discovery and recommendation (eddytools)* (Chapter 5, addresses challenges 1.3.4, 1.3.5, and 1.3.6). We propose a framework to assist the analyst in the event log building phase by providing methods to discover and recommend interesting case notions, and to build the corresponding event logs:
  - *Multiple perspectives on event data* (Addresses Challenge 1.3.4). We propose a technique to define case notions based on the underlying data schema, providing different ways to correlate events, and therefore multiple views on the data. This allows us to automatically define candidate case notions based on the data schema at hand. Also, when domain knowledge is available, custom case notions can be defined using this formalism.
  - *Automated event log building based on data schema* (Addresses challenges 1.3.4 and 1.3.6). We provide an algorithm to build event logs given a case notion and a dataset extracted from a database. The combination of this algorithm with the proposed case notion discovery technique provides an automated approach to event log building.
  - *Case notion recommendation framework* (Chapter 5, addresses challenges 1.3.4, 1.3.5, and 1.3.6). We propose a method for case notion ranking based on their potential "interestingness". This reduces the computation time, avoiding the generation of meaningless event logs. The goal is to help analysts to focus on the most interesting views on the event data in the exploratory phases of the analysis or at times when domain knowledge is not available.

Data Querying

Data querying is a phase that can be performed at any moment, i.e., before and after the data extraction, and before and after the event log building. When performed before the data extraction, we need to query considering the original structure of the data, as well as the specific system in which the data are stored, e.g., Oracle

RDBMS. However, when performed after the extraction, we can assume a common structure (in our case the proposed meta-model) regardless of the source of data. This helps to standardize the querying process. In order to provide an easier and more standard way to query data in the process mining context, we present the following contribution:

- *Data-aware process oriented query language (DAPOQ-Lang)* (Chapter 7, addresses Challenge 1.3.7). With this query language we aim at simplifying the query writing step when carrying out a process mining project. We provide a set of constructs that makes it possible to write more compact and readable queries, to obtain insights when exploring the data, and to build specialized event logs exploiting domain knowledge about the data in an easier way.

1.4.2 Thesis Structure

This thesis is structured into 9 chapters, covering the following topics:

Chapter 1: Introduction. This chapter provides an introduction to the topic and aim of this thesis. Challenges and contributions are discussed as well.

Chapter 2: Preliminaries. In this chapter, we introduce the basic mathematical notation and formal definitions required to understand the rest of this thesis.

Chapter 3: OpenSLEX: A Meta-Model for Process Mining. We propose a meta-model for process mining, especially focused on database environments, that combines data and process aspects of an organization. This chapter is based on [40,42].

Chapter 4: OpenSLEX in Practice: Data Extraction and Querying. In this chapter, we evaluate our meta-model on three different real-life environments. We provide the adapters needed to extract and transform data in order to comply with the OpenSLEX meta-model. We also performed a limited analysis of the extracted data as a sample of the opportunities for analysis enabled by this technique. This chapter is based on [38,40,42,44].

Chapter 5: Case Notion Discovery and Recommendation. We propose a method to compute candidate case notions based on the schema of the database under study, as well as a technique to predict the possible "interestingness" of the resulting event log based on certain metrics. The accuracy of the prediction is evaluated with respect to an established ground truth. This chapter is based on [39].

Chapter 6: Process Mining Techniques Applied: Data Properties and Opportunities. In this chapter we present a survey of process mining techniques that can be applied on event logs obtained from databases, according to the characteristics of the data at hand. The aim is to provide a tease on the types of analyses that can be performed on the data obtained by the techniques that we presented in the previous chapters.

Chapter 7: Data-Aware Process Oriented Querying. We exploited the structure of the OpenSLEX meta-model to create a query language focused on ease of use and compactness. We provide a specification of the syntax and semantics, together with a performance evaluation. The most common use cases are listed together with sample queries that demonstrate the benefits of this query language. This chapter is based on [41, 43].

Chapter 8: Case Study: Process Mining on a Health Information System. The techniques presented in previous chapters are tested in this case study on a healthcare system. The whole process mining pipeline is presented, from the first contact with the original database to the process mining analysis of the resulting event logs.

Chapter 9: Conclusion. We conclude this thesis with a summary of the achievements and lessons learned, together with ideas for future work.

The honeybee: a, worker; b, queen; c, drone.
"Beekeeping: a discussion of the life of the honeybee and of the production of honey",
Everett Franklin Phillips, 1923

2 Preliminaries

In this chapter, we recall some basic mathematical concepts like sets, tuples, powersets, functions, partial functions, sequences, and multi-sets. Also, we present a notion of database at a formal level, a description of the ETL (extract, transform, load) process, and a formalization of event logs and Petri nets.

2.1 Notations

In this section, we introduce some basic mathematical concepts and notations, which will be widely used across the chapters of this thesis.

Definition 1 (Set) *A set is a well-defined, possibly infinite, unsorted collection of distinct objects. The objects that belong to a set are called elements. A set is usually denoted by a capital letter such as X, and it can be described listing its elements between curly braces, e.g., $X = \{a, b, c\}$. We use $\in$ to express that an element belongs to a set, e.g., $a \in X$. An empty set is denoted by $\emptyset$, and the cardinality of a set (number of elements) is denoted by $|X|$. We can perform the following operations on sets: union ($X \cup Y$), intersection ($X \cap Y$), and difference ($X \setminus Y$). A set X is a subset of set Y ($X \subseteq Y$) when all the elements of X are included in Y. A set X is a strict subset of Y ($X \subset Y$) if X is a subset of Y and they are not equal.*

Definition 2 (Tuple) *A tuple is a finite ordered list of elements. A tuple can be expressed by listing its elements in order between parentheses, e.g., $A = (a, b, c)$. Also, it can be expressed as belonging to the Cartesian product of other sets, e.g., $A = B \times C$.*

Definition 3 (Powerset) *The powerset of a set X ($\mathscr{P}(X)$) is the set of all possible subsets of X, including X and the empty set ($\emptyset$), i.e., $Y \in \mathscr{P}(X) \iff Y \subseteq X$.*

Definition 4 (Function and partial function) *A function f from X to Y, maps an element in X to an element in Y, and is expressed as* $f \in X \rightarrow Y$. *The domain of f is X if for every element in X, f is defined and maps it to a value in Y, i.e.,* $\mathbf{dom}(f) = X \iff \forall x \in X : f(x) \in Y$. *The range of f is the set of the images of all elements in the domain, i.e.,* $\mathbf{range}(f) = \{f(x) \mid x \in \mathbf{dom}(f)\}$. *A partial function* $g \in X \not\rightarrow Y$ *is a function whose domain is a subset of X, i.e.,* $\mathbf{dom}(g) \subseteq X$*, which means that g does not need to be defined for all values in X.*

Definition 5 (Sequence) *For a given set A,* A^* *is the set of all non-empty finite sequences over A plus the empty sequence. A non-empty finite sequence over A of length* n *is a mapping* $\sigma \in \{1, ..., n\} \rightarrow A$. *Such a sequence is represented by a string, i.e.,* $\sigma = \langle a_1, a_2, ..., a_n \rangle$ *where* $a_i = \sigma(i)$ *for* $1 \leq i \leq n$. $|\sigma|$ *denotes the length of the sequence, i.e.,* $|\sigma| = n$.

Definition 6 (Multi-set (bag)) *Given a finite set A, a multi-set M over A is a function* $M : A \rightarrow \mathbb{N}$. $\mathbb{B}(A)$ *denotes the set of all multi-sets over a finite domain A. The size of a multi-set is denoted as* $|M|$*, and is defined as* $|M| = \sum_{a \in A} M(a)$.

A multi-set M over a set A is a set in which each element of A may occur multiple times. An example multi-set can be expressed as $M = [a, b^2, c^4]$ where $a, b, c \in A$, $M(a) = 1$, $M(b) = 2$, and $M(c) = 4$.

2.2 Databases

Within the scope of this thesis, databases represent our main source of data. For this reason, it is important to clearly define the assumptions made with respect to the structure of the information that we can find in them. We assume that a relational database will always comply with the following requirements:

- A data model is defined,
- There is a set of class/table names,
- There is a primary key defined for each class,
- A foreign key can only refer to an existing primary key,
- There must be a one-to-one mapping between the attributes of a foreign key and the attributes of the primary key that it refers to.

Even if these requirements are fulfilled, they may not be available to the analyst. In that case, some limitations exist with respect to the techniques that can be applied to carry out a process mining project. In the coming chapters, we will propose ways to overcome these limitations, and still be able to perform the analysis.

In this thesis, we deal with environments that use a relational database to store all the relevant information. Therefore, all of them share some characteristics. In further chapters we provide a formalization of a standard meta-model for storage of data obtained from such sources. But first, Definition 7 formalizes the concept of source data model for a certain relational database:

Definition 7 (Source Data Model) *Assume V to be some universe of values. A source data model is a tuple $SDM = (C, A, classAttr, val, PK, FK, keyClass, keyRel, keyAttr, refAttr)$ such that:*

- *C is a set of class names,*
- *A is a set of attribute names,*
- *$classAttr \in C \rightarrow \mathscr{P}(A)$ is a function mapping each class onto a set of attribute names. Each attribute $a \in A$ must belong to the set of attributes mapped to one class $c \in C$. A_c is a shorthand denoting the set of attributes of class $c \in C$, i.e., $A_c = classAttr(c)$,*
- *$val \in A \rightarrow \mathscr{P}(V)$ is a function mapping each attribute onto a set of values. $V_a = val(a)$ is a shorthand denoting the set of possible values of attribute $a \in A$,*
- *PK is a set of primary and unique key names,*
- *FK is a set of foreign key names,*
- *PK and FK are disjoint sets, that is $PK \cap FK = \emptyset$. To facilitate further definitions, the shorthand K is introduced, which represents the set of all keys: $K = PK \cup FK$,*
- *$keyClass \in K \rightarrow C$ is a function mapping each key name to a class. K_c is a shorthand denoting the set of keys of class $c \in C$ such that $K_c = \{k \in K \mid keyClass(k) = c\}$,*
- *$keyRel \in FK \rightarrow PK$ is a function mapping each foreign key onto a primary key,*
- *$keyAttr \in K \rightarrow \mathscr{P}(A)$ is a function mapping each key onto a set of attributes, such that $\forall k \in K : keyAttr(k) \subseteq A_{keyClass(k)}$,*
- *$refAttr \in (FK \times A) \not\rightarrow A$ is a function mapping each pair of a foreign key and an attribute onto an attribute from the corresponding primary key, i.e., $dom(refAttr) = \{(k, a) \in FK \times A \mid a \in keyAttr(k)\}$, and $\forall k \in FK : \forall a \in keyAttr(k) : (refAttr(k, a) \in keyAttr(keyRel(k))$. Also, every attribute of the referred primary key must be mapped to an attribute of the referring foreign key, i.e., $\forall k \in FK : \forall a \in keyAttr(keyRel(k)) : \exists a' \in keyAttr(k) : refAttr(k, a') = a$. Finally, each attribute of the referred primary key can only be mapped to one attribute from each referring foreign key $\forall k \in FK : \forall a, a' \in keyAttr(k) : refAttr(k, a) = refAttr(k, a') \implies a = a'$.*

The following notations in Definition 8 will help us to express mappings and objects in a compact way.

Definition 8 (Notation) *Let $SDM = (C, A, classAttr, val, PK, FK, keyClass, keyRel, keyAttr, refAttr)$ be a source data model.*

- *$M^{SDM} = \{map \in A \not\rightarrow V \mid \forall a \in dom(map) : map(a) \in V_a\}$ is the set of mappings,*
- *$O^{SDM} = \{(c, map) \in C \times M^{SDM} \mid dom(map) = classAttr(c)\}$ is the set of all possible objects of SDM.*

Another aspect of databases is the concept of *source object model.* Definition 9 describes this concept, which corresponds to a snapshot of the database at a certain moment in time.

Definition 9 (Source Object Model) *Let* $SDM = (C, A, classAttr, val, PK, FK, keyClass, keyRel, keyAttr, refAttr)$ *be a source data model. A source object model of SDM is a set* $SOM \subseteq O^{SDM}$ *of objects.* $U^{SOM}(SDM) = \mathscr{P}(O^{SDM})$ *is the set of all object models of SDM.*

To ensure the validity of the source object model, i.e., the fulfillment of all the constraints such as primary and foreign keys, we introduce the concept of *valid source object model* in Definition 10.

Definition 10 (Valid Source Object Model) *Assume* $SDM = (C, A, classAttr, val, PK, FK, keyClass, keyRel, keyAttr, refAttr)$ *to be a source data model.* $VSOM \subseteq U^{SOM}(SDM)$ *is the set of valid source object models. An object model* $SOM \in VSOM$ *if and only if the following requirements hold:*

- $\forall (c, map) \in SOM : (\forall k \in K_c \cap FK : (\exists (c', map') \in SOM : keyClass(keyRel(k)) = c' \wedge (\forall a \in keyAttr(k) : map(a) = map'(refAttr(k, a)))))$*, i.e., referenced objects must exist,*
- $\forall (c, map), (c, map') \in SOM : (\forall k \in K_c \cap PK : ((\forall a \in keyAttr(k) : map(a) = map'(a)) \implies map = map'))$*, i.e., PK values must be unique.*

For the remainder of this thesis, we assume that any database we deal with will be defined by a data model according to Definition 7, and a valid source object model as described in Definition 10. Also, incomplete data in the form of empty values is allowed, as long as it does not conflict with the constrains set by the data model. This means that primary and foreign key constraints must be satisfied.

2.3 ETL: Extract, Transform, Load

Extract, transform, load (ETL) [57] is a process commonly executed when working with databases. It refers to the following steps involved in the preparation of data from databases for analysis or querying:

1. Extract: data are extracted from homogeneous or heterogeneous data sources, e.g., monolithic databases, and distributed databases with the same or different structure,
2. Transform: data are transformed in order to be stored in the appropriate format required for the purpose of analysis, e.g., data quality and consistency standards are enforced, and data are normalized and merged,
3. Load: data are loaded in the final database, e.g., a data store or a data warehouse.

ETL is a common process when dealing with data from different sources within organizations to be integrated in a common data warehouse. The next chapter proposes a meta-model to structure data obtained from databases in order to enable the application of process mining techniques.

2.4 Event Logs

In this section we present the definition of event log as proposed in [109]. A process consists of cases, and a case consists of events. Events within a case are ordered, and can have attributes. Examples of typical attribute names are activity, time, costs, and resource.

Definition 11 (Universes [109]) *We define the following universes:*

- *ϵ denotes the set of all possible event identifiers,*
- *AN denotes the set of all possible attribute names,*
- *V denotes the set of all possible attribute values,*
- *$\mathscr{A}$ denotes the set of all possible activity names,*
- *I denotes the set of all possible process instance identifiers,*
- *TS denotes the set of all possible timestamps,*
- *CA denotes the set of all possible cases.*

Definition 12 (Event, Attribute [109]) *Let ϵ be the set of all possible event identifiers, and AN the set of all possible attribute names. Events can be described by several attributes, e.g. time of occurrence, resource involved, name of the performed activity, etc. Given an event $e \in \epsilon$ and a name $n \in AN$, we say that $\#_n(e)$ is the value of attribute n for event e. If event e does not have attribute n, then $\#_n(e) = \perp$ (null value).*

We define some shorthands for the most common, although optional, event attributes:

- $\#activity(e)$ is the activity associated to event e,
- $\#time(e)$ is the timestamp of event e,
- $\#resource(e)$ is the resource associated to event e,
- $\#trans(e)$ is the transaction type, also known as life-cycle, associated to event e.

Some conventions apply for these attributes. In particular, timestamps should be non-descending in the event log. Also, we assume the timestamp universe TS such that $\#time(e) \in TS$ for any $e \in \epsilon$.

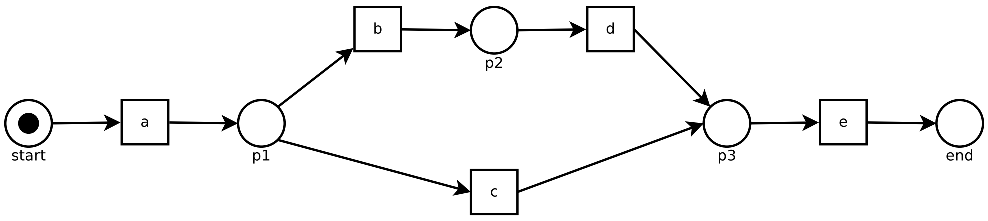


Figure 2.1: Example of a Petri net with initial marking *start* and final marking *end*.

Definition 13 (Case, trace, event log [109]) *Let CA be the universe of all possible cases. Cases have attributes. For any case $c \in CA$ and name $n \in AN : \#_n(c)$ is the value of attribute n for case c. If c does not have an attribute named n, then $\#_n(c) = \perp$. Each case has a mandatory attribute trace, $\#_{trace}(c) \in \epsilon^*$.*

A trace is a finite sequence of events $\sigma \in \epsilon^*$ such that each event appears only once, i.e., for $1 \leq i < j \leq |\sigma| : \sigma(i) \neq \sigma(j)$.

An event log is a set of cases $L \subseteq CA$. If an event log contains timestamps, then the ordering in a trace should respect these timestamps, i.e., for any $c \in L$, i and j such that $1 \leq i < j \leq |\hat{c}| : \#_{time}(\hat{c}(i)) \leq \#_{time}(\hat{c}(j))$.

2.5 Process Models

An event log usually describes the observed behavior of a certain process. It is possible to capture and abstract such behavior in a *process model*. A process model uses graphs to describe the causal dependencies between the activities of a process.

Petri nets are one of the oldest and most well-studied process modeling languages. The notation is simple and intuitive, yet powerful enough to allow us to model complex structures. One important characteristic of Petri nets is that they are executable and they support concurrency.

A Petri net can be formalized as a bipartite graph, consisting of *places* and *transitions*. *Tokens* can flow through the network following certain firing rules. The state of a Petri net is determined by its *marking*, which captures the distribution of tokens over places. The initial marking of a network determines its marking right before the execution starts.

Definition 14 (Petri net [109]) *A Petri net is a triplet $N = (P, T, F)$ where P is a finite set of places, T is a finite set of transitions such that $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the flow relation. A marked Petri net is a pair (N, M) where $N = (P, T, F)$ is a Petri net and where $M \in \mathbb{B}(P)$ is a multi-set over P denoting the marking of the net. The set of all marked Petri nets is denoted $\mathcal{N}$.*

Figure 2.1 depicts a sample Petri net, which can be formalized as follows: $P = \{start, p1, p2, p3, end\}$, $T = \{a, b, c, d, e\}$, and $F = \{(start, a), (a, p1), (p1, b), (p1, c), (b, p2), (c, p3), (p2, d), (d, p3), (p3, e), (e, end)\}$.

Given a Petri net $N = (P, T, F)$, we say that, for any $x \in P \cup T$, $\bullet x = \{y | (y, x) \in F\}$ denotes the set of input nodes and $x\bullet = \{y | (x, y) \in F\}$ denotes the set of output nodes. For a marking M of net N, a transition $t \in T$ is *enabled* if each of its input places $\bullet t$ contains at least one token, and it is denoted as $(N, M)[t\rangle$. If a transition t is enabled, it can be *fired*. When a transition fires, one token is removed from each of the input places $\bullet t$ and one token is put in each of the output places $t\bullet$. The expression $(N, M)[t\rangle$ denotes that t is enabled at marking M. $(N, M)[t\rangle(N, M')$ denotes that firing the enabled transition t results in marking M'.

Given a marked Petri net (N, M_0) with the initial marking $M_0 = [start]$, as shown in Figure 2.1, we say that the empty sequence $\sigma = \langle\rangle$ is enabled in (N, M_0), i.e., $\langle\rangle$ is a firing sequence of (N, M_0). The firing sequence $\sigma = \langle a, b\rangle$ is also enabled. Firing σ results in marking $[p2]$. A marking M is reachable from the initial marking M_0 if and only if there exists a sequence of enabled transitions σ whose firings lead from M_0 to M.

In order to relate transitions in the Petri net to activities, we defined a labeled Petri net.

Definition 15 (Labeled Petri net [109]) *A labeled Petri net is a tuple $N = (P, T, F, A, l)$ where (P, T, F) is a Petri net as defined in Definition 14, $A \subseteq \mathscr{A}$ is a set of activity labels, and $l \in T \to A$ is a labeling function.*

Multiple transitions may be labeled equally. Each label, which we call activity, represents the observable action. When a transition is not observable, it is labeled with the reserved label τ, and is considered as a *silent* transition.

2.6 Process Mining

The concept of *event log* has been introduced in Section 2.4. Also, *process models* and *Petri nets* have been presented in Section 2.5. In this section, we describe two of the most important process mining tasks, process discovery and conformance checking.

2.6.1 Process Discovery

Process discovery is the task of discovering process models based on an event log. The goal is to provide insights on the behavior of real-life processes. The contributions of this thesis do not focus on the process discovery task, but on the data extraction and preparation needed to obtain event logs suitable for process mining analysis. One of these types of analysis is process discovery. Therefore, our contributions help to enable the application of process discovery and other process mining techniques in environments that present certain data challenges.

Many discovery algorithms exist in the literature [64, 76, 122]. In this thesis we do not focus on any specific technique. However, we describe *process discovery in general as a function that takes an event log as an input and produces a process model as an output.* Figure 2.2 shows a Petri net discovered from the sample event log proposed in Chapter 1 using the Inductive Miner.

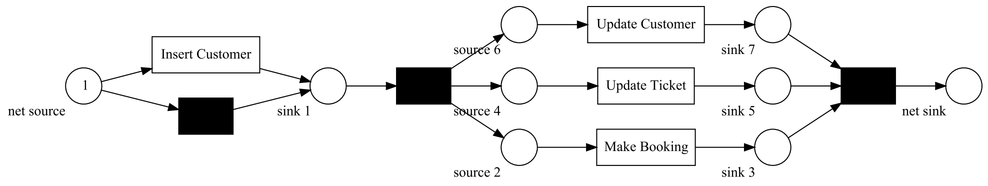


Figure 2.2: Petri net discovered using the Inductive Miner [64] from the sample event log in Table 1.1.

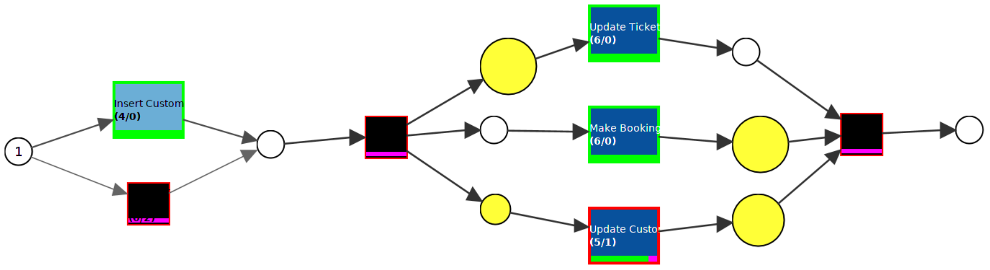


Figure 2.3: Conformance checking result from the alignment of the Petri net in Figure 2.2 with the event log in Table 1.1.

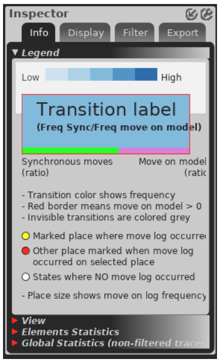
A process model can be described as a set of process runs (or sequences). Given a labeled Petri net $N = (P, T, F, A, l)$ with an initial marking M_0 and a final marking M_f, an alternative way to describe this process model is by computing the set S of all possible firing sequences that lead from marking M_0 to marking M_f. In some cases, this set of sequences can be infinite (e.g., in the presence of loops).

When discovering a process model from an event log, the set of possible sequences S allowed by such model does not need to be equal to the behavior observed in the event log. The discovered model could allow for more behavior (sequences) than the ones observed in the log. Also, the event log could contain more sequences than the ones allowed by the discovered process model. The differences between what is observed in an event log and what is allowed by a process model can be due to different reasons. Normally, we want discovery algorithms to avoid over-fitting to the event log, i.e., be able to generalize. This means that we are interested in process models that allow for more behavior than the one observed in the event log.

2.6.2 Conformance Checking and Alignments

One of the aims of the *conformance checking* task is to aligning observed behavior captured in an event log with the allowed or normative behavior allowed by a process model. Also, it allows to find deviations between observed and normative behavior.

The alignment technique [111] is a conformance checking method that aligns behavior between log and model. For each trace existing in the event log, it computes the closest run in the process model. The result is a set of *alignments*. Each alignment



Synchronous moves (ratio) Move on model (ratic

- Transition color shows frequency
- Red border means move on model > 0
- Invisible transitions are colored grey

🟡 Marked place where move log occurre
🔴 Other place marked when move log occurred on selected place
○ States where NO move log occurred

- Place size shows move on log frequency

Figure 2.4: Legend to interpret conformance checking results.

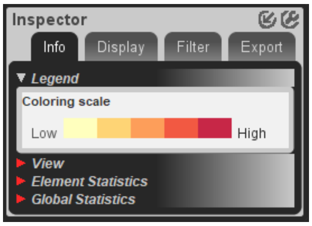
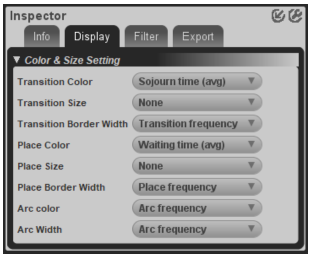


Figure 2.5: Legend to interpret performance analysis results.



Transition Color	Sojourn time (avg)
Transition Size	None
Transition Border Width	Transition frequency
Place Color	Waiting time (avg)
Place Size	None
Place Border Width	Place frequency
Arc color	Arc frequency
Arc Width	Arc frequency

Figure 2.6: Metrics displayed per node in a Petri net with performance analysis results.

can be defined as a sequence of moves, and each move relates an event in a trace of the event log to an instance of a transition in a firing sequence of a model. There are three types of move:

- A *synchronous move* represents a match between an observed event and an allowed event in the model,
- A *log move* happens when an observed event cannot be replayed by the model, i.e., none of the enabled transitions in the model are related to the observed event in the log,
- A *model move* happens when none of the behavior allowed by the model is observed in the event log, e.g., the execution of the only enabled transition is not observed in the event log.

Figure 2.3 depicts a Petri net with conformance information. Figure 2.4 shows the legend to interpret the colors and sizes of the augmented Petri net. We see that, when aligning the event log to the process, most of the transitions (*Insert Customer*, *Update Ticket*, and *Make Booking*) can be executed synchronously with the behavior observed in the log. However, in one of the cases, there was a model move in the transition *Update Customer*. This means that for a certain trace in the log, the firing sequence could not be perfectly aligned with the model, and the *Update Customer* transition had to be skipped. Also, we see that certain places have a yellow color. This means that a log move happened at that place when computing the alignment for certain traces. The size of the place node represents the move on log frequency.

Also, computing alignments allows us to display performance information on top of a process model. Figure 2.7 shows a Petri net augmented with performance statistics. Figure 2.5 shows the color code used. An intense red color indicates a high value, while the pale yellow represents a low one. As shown in Figure 2.6, the metric represented in

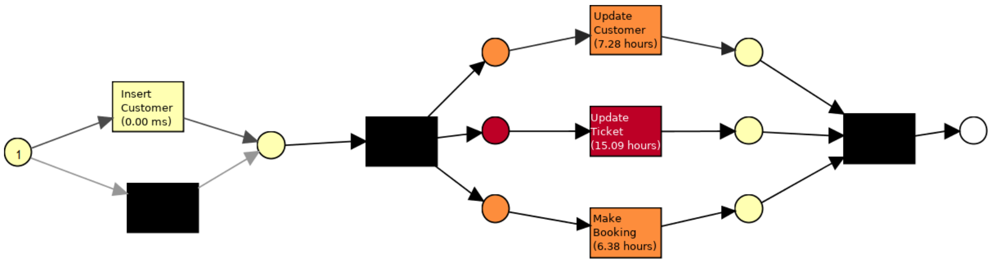
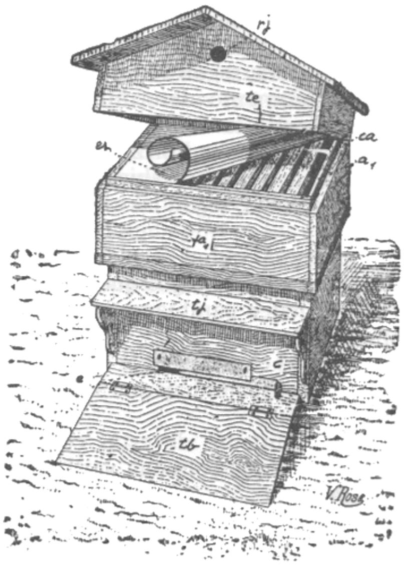

Insert Customer (0.00 ms)
1
Update Customer (7.28 hours)
Update Ticket (15.09 hours)
Make Booking (6.38 hours)

Figure 2.7: Performance information resulting from the alignment of the Petri net in Figure 2.2 with the event log in Table 1.1.

each transition of the model corresponds to the average *sojourn time*, i.e., time spent in a particular state. The metric represented in the places is the average waiting time, i.e., time between the completion of the previous activity and the start of the next one.

2.7 Chapter Summary

In this chapter we introduced most of the basic definitions, concepts, and notation used throughout the rest of the chapters of this thesis. The next chapter, the first one of the contribution chapters, proposes a meta-model for process mining on databases.



Vertical hive with one single body and a brood chamber.

"Cours complet d'apiculture", Georges de Layens and Gaston Bonnier, 1897

3 OpenSLEX: A Meta-Model for Process Mining

In Chapter 1, we introduced the concept of process mining. Next, we set the scope of this thesis to the application of process mining on databases. Also, we presented some of the challenges to face when dealing with data stored in databases with the purpose of applying process mining to it. In Chapter 2, we introduced several concepts in process mining in general (event log, process model), and some focused on databases (database, data model, case notion, event log extraction).

In the following, we discuss the challenge of event data extraction from databases, as well as how the extract, transform, load process (ETL) can be improved with respect to the state of the art. The motivation is supported by a running example. Next, we propose a meta-model for process mining on databases, Open SQL Log Exchange (OpenSLEX), for which we provide a formalization. Also, we give details of an implementation of the meta-model. Finally, we discuss related work.

3.1 Introduction

The field of process mining offers a wide variety of techniques to analyze event data. Process discovery, conformance and compliance checking, performance analysis, process monitoring and prediction, and operational support are some of the techniques that process mining provides to better understand and improve business processes. However, most of these techniques rely on the existence of an event log.

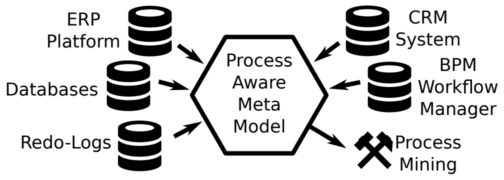


Figure 3.1: Data gathering from several systems to a meta-model

Obtaining event logs in real-life scenarios is not a trivial task. While this may be different in the future, it is not yet common to find logs exactly in the right form. In many occasions, such logs simply do not exist and need to be extracted from some sort of storage, like databases. In situations when a database exists that contains meaningful data, several approaches are available to extract events. The most general is the classical extraction in which events are manually obtained from the tables in the database. In that case, substantial domain knowledge is required to select the right data, which are normally scattered across tables. Some work has been done in this field to assist in the extraction and log generation task [13]. Also, studies have been performed on how to extract events in specific environments like SAP [55, 73, 104] or other ERP systems [85]. In [44] we presented a more general solution to extract events from databases, regardless of the application under study. The paper describes how to automatically obtain events from database systems that generate *redo-logs* (transaction logs) as a way to recover from failure. The extracted events represent data changes, and the technique makes it possible to build event logs even on systems that are not process-aware. All mentioned approaches aim at, eventually, generating an *event log*, i.e., a set of traces, each of them containing a set of *events*. These events represent operations or actions performed in the system under study and are grouped in traces following some criteria. However, there are multiple ways in which events can be selected and grouped into traces. Depending on the perspective we want to take on the data, we need to extract event logs differently. Also, a database contains a lot more information than just events. The extraction of events and its representation as a plain event log can be seen as a "lossy" process, which means that during this activity valuable information can get lost. Considering the prevalence of databases as a source for event logs, it makes sense to gather as much information as possible, combining the process view with the actual data.

We see that process mining techniques grow more and more sophisticated. Yet, the most time-consuming activity, *event log extraction*, is hardly supported. In big industrial database settings, where event data is scattered over hundreds of tables, and many processes coexist within the same environment, the queries used to extract event logs can become very complicated, difficult to write, and hard to modify. Ideally, users should be able to find events explicitly defined and stored in a centralized way. These events should be defined in such a way that event correlation and log building could be performed effortlessly and be easily adapted to the business questions to answer in each situation. Also, to discover meaningful data rules, these events should be annotated with enough data attributes.

This chapter aims at providing support to tackle the problem of *obtaining, transforming, organizing, and deriving data and process information from databases*, abstracting the raw data to high-level business concepts such as events, cases, and activities. This means that, after the ETL process is applied, users will no longer have to deal with low-level raw data distributed over tables (e.g., timestamps, activity names, and case ids as columns of different tables that need to be joined together). Conversely, users will be able to focus on the analysis, dealing only with familiar process elements such as events, cases, logs, and activities, or with data elements such as objects, object versions[1], object classes, and attributes. Also, the new OpenSLEX meta-model proposed in this work can be seen as a data warehouse schema that captures all the pieces of information necessary to apply process mining to database environments.

In order to build event logs with events from databases, languages like SQL are the natural choice for many users. The point of this thesis is not to replace the use of query languages, but to provide a common meta-model as a standard abstraction that ensures process mining applicability. Moreover, one of the main advantages of the adoption of a standard meta-model has to do with multi-perspective event log building and analysis. Many different event logs can be built from the information stored in a database. Looking at data from different perspectives requires ad-hoc queries that extract and correlate events in different ways (e.g. (a) order, delivery, and invoice events, versus (b) order, product, and supplier events). Our meta-model defines the abstract concepts that need to be extracted to enable this multi-perspective event log building in a more intuitive way.

Additionally, as a result of the adoption of the proposed meta-model, it becomes easier to connect the event recording system[2] of enterprises with analysis tools, generating different views on the data in a flexible way. Also, this work presents a comprehensive integration of process and data information in a consistent and unified format. All of this is supported by our implementation. Moreover, the provided solution has the benefit of being universal, being applicable regardless of the specific system in use. Figure 3.1 depicts an environment in which the process information of a company is distributed over several systems of different types, like ERPs, CRMs, BPM managers, database systems, redo-logs, etc. In such a heterogeneous environment, the goal is to extract, transform, and derive data from all sources, consolidating it into a common representation. By putting all pieces together, analysis techniques like process mining can be readily applied.

The remainder of this chapter is structured as follows: Section 3.2 presents a running example used throughout the chapter. Section 3.3 explains the proposed meta-model and provides a formalization. Implementation details are presented in

[1]In this context, the term "version" or "object version" refers to an instance of an object at a point in time (e.g. the database object corresponding to a specific customer had two different values for the attribute "address" at different points in time, representing two versions of the same customer object). This is different of the usual meaning of "version" in the software context as a way to distinguish different code releases (e.g. version 1.1).

[2]An event recording system stores event data that reflects the execution of transactions and activities, the occurrence of exceptions, and the reception of signals and messages, at a certain moment in time.

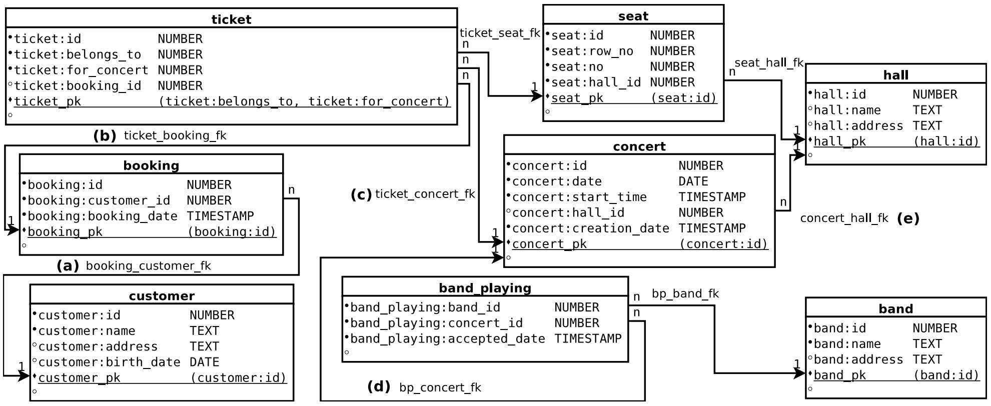


Figure 3.2: Data schema of the example database

Section 3.4. Section 3.5 discusses the related work and, finally, Section 3.6 presents the conclusions.

3.2 Running Example

In this section, we propose a running example to explain and illustrate our approach. Assume we want to analyze a setting where concerts are organized and concert tickets are sold. A database is used to store all the information related to concerts, concert halls (*hall*), seats, tickets, bands, performance of bands in concerts (*band_playing*), customers, and bookings. Figure 3.2 shows the data schema of the database in use. In it we see many different elements of the involved process represented. Let us consider now a complex question that could be raised from a business point of view: *What is the process followed by customers between 18 and 25 years old who bought tickets for concerts of band X?* This question represents a challenge starting from the given database for several reasons:

1. The database does not provide an integrated view of process and data. Therefore, questions related to the execution of the underlying process with respect to some of the elements cannot be directly answered with a query.

2. The current database schema fits the purpose of storing the information in this specific setting, but it does not have enough flexibility to extend its functionality allocating new kinds of data such as events or objects of a different nature.

3. The setting lacks execution information in an accessible way (events, traces, and logs are missing so one cannot apply process mining directly) and there is no assistance on how to extract or derive this information from the given data.

4. If we plan to use the data as they are, we need to adapt to the way it is stored for every question we want to answer.

All these reasons make the analysis complex. *At best, any analysis of this sort can only be carried out by the extraction of a highly specialized event log by creating a complex ad-hoc query. On top of that, the extraction will need to be repeated for new questions that require a new viewpoint on the data.*

If we consider that, for the present database, some sort of event recording system is in place, the extraction of additional events becomes feasible. Listing 3.1 shows an example of an ad-hoc query to answer the sample question posted above. This query makes use of the event data distributed over the database (*booking*, *band*, etc.), as depicted in Figure 3.2, together with events recorded by a redo-log (or transaction log) system provided by the RDBMS (*redo_logs_mapping* table). The way to access redo-log information has been simplified in this example for the sake of clarity. The first part of the query retrieves *ticket booked* events, making use of the *booking_-date* timestamp stored in the table *booking*. These events are united to the ones corresponding to *customer* events. These last ones are obtained from redo-logs, due to the lack of such event information in the data schema. Next, events belonging to the same customer are correlated by means of the *caseid* attribute, while the cases are restricted to the ones belonging to customers aged 18-25 at the time the booking was made. Additionally, we need to keep only the bookings made on bands named "X" at the moment of the booking (a band could change its name at any point in time). This is an especially tricky step since we need to look into the redo-logs to check if the name was different at the time the booking was made.

Query 3.1: Sample SQL query to obtain a highly customized event log from the database

```
SELECT * FROM (
 SELECT
  "ticket booked" as activity,
  BK.booking_date as timestamp,
  BK.customer_id as caseid
 FROM booking as BK
 UNION
 SELECT
  concat(RL.operation," customer profile") as activity,
  RL.timestamp as timestamp,
  rl_value_of('id') as caseid
 FROM redo_logs_mapping as RL
 WHERE
  RL.table = "CUSTOMER"
 ) as E,
 booking as BK,
 customer as CU,
 ticket as T,
 concert as C,
 band_playing as BP,
 band as B
WHERE
 E.caseid = CU.id AND
 CU.id = BK.customer_id AND
 BK.id = T.booking_id AND
 T.for_concert = C.id AND
 C.id = BP.concert_id AND
 BP.band_id = B.id AND
 "0018-00-00 00:00:00" <= (BK.booking_date - CU.birth_date) AND
 "0025-00-00 00:00:00" >= (BK.booking_date - CU.birth_date) AND
 (
  (B.name = "X" AND
   B.id NOT IN
    (SELECT rl_value_of('id') as id
     FROM redo_logs_mapping as RL
     WHERE
      RL.table = "BAND"
    )
  )
  OR
  (B.id IN
    (SELECT rl_value_of('id') as id
     FROM redo_logs_mapping as RL
     WHERE
      RL.table = "BAND" AND
      rl_new_value_of('name') = "X" AND
      RL.timestamp <= BK.id AND
     ORDER BY RL.timestamp DESC LIMIT 1
    )
  )
  OR
  (B.id IN
    (SELECT rl_value_of('id') as id
     FROM redo_logs_mapping as RL
     WHERE
      RL.table = "BAND" AND
      rl_old_value_of('name') = "X" AND
      RL.timestamp >= BK.id AND
     ORDER BY RL.timestamp ASC LIMIT 1
    )
  )
 )
```

It is evident that extracting customized event logs that answer very specific questions, while maintaining the original data schema, is possible. However, such queries will have to be adapted or rewritten for every different setting or event recording system. The fact that users, in particular business analysts, need to be knowledgeable about the specifics of how event data is stored represents an important challenge to overcome. This is one of the main reasons why so much time and effort is consumed during the ETL phase, which would better be devoted to analysis. This work aims at supporting the ETL phase and ultimately at decoupling it from the analysis by providing a set of familiar abstractions readily available for business analysts.

3.3 Meta-Model

As has been shown before, a need exists for a way to store execution information in a structured way, something that accepts data from different sources and allows building further analysis techniques independently from the origin of these data. Efforts in this field have already been made, as can be observed in [1] with the IEEE XES standard. This standard defines a structure to manage and manipulate logs, containing events and traces and the corresponding attributes. Therefore, XES is a good format to represent behavior. However, an XES file is just one view on the data and, despite being an extensible format, it does not provide a predefined structure to store all the linked information we want to consider.

Because of this, it seems necessary to define a structured way to store additional information that can be linked to the classical event log. This new way to generalize and store information must provide sufficient details about the process, the data types, and the relations between all the elements, making it possible to answer questions at the business level, while looking at two different perspectives: data and process.

3.3.1 Requirements

To be able to combine the data and process perspectives in a single structure, it is important to define a set of requirements that a meta-model must fulfill. It seems reasonable to define requirements that consider backwards-compatibility with well-established standards, support of additional information, its structure, and the correlation between process and data views:

1. The meta-model must be compatible with the current meta-model of XES, i.e., any XES log can be transformed into the new meta-model and back without loss of information,
2. It must be possible to store several logs in the new meta-model, avoiding event duplication,
3. Logs stored in the same meta-model can share events and belong to different processes,
4. It must be possible to store some kind of process representation in the meta-model,
5. The meta-model must allow storing additional information, like database objects, together with the events, traces, processes, and the correlation between all these elements,
6. The structure of additional data must be precisely modeled,
7. All information mentioned must be self-contained in a single storage format, easy to share and exchange, similarly to the way that XES logs are handled.

The following section describes the proposed meta-model, which complies with these requirements, providing a formalization of the concepts along with explanations.

3.3.2 Formalization

Considering the typical environments subject to study in the process mining field, we can say that it is common to find systems backed up by some sort of database storage system. Regardless of the specific technology behind these databases, all of them have in common that the data are structured in some way, e.g., similar data are stored in the same tables with values for the same attributes. We can describe our meta-model as a way to integrate process and data perspectives, providing flexibility on its inspection and assistance to reconstruct the missing parts. Figure 3.3 shows a high-level representation of the meta-model. On the right-hand side, the data perspective is considered, while the left-hand side models the process view. Assuming that the

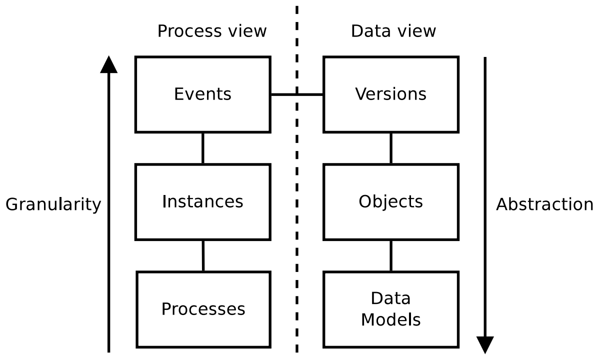


Figure 3.3: Diagram of the meta-model at a high level of abstraction

starting points of our approach are data, we see that the less abstract elements of the meta-model, *events* and *versions*, are related, which provides the connection between the process and data view. These are the basic blocks of the whole structure and, usually, the rest can be derived from them.

The data side considers three elements: **data models**, **objects**, and **versions**. The data models provide a schema describing the objects of the database. The objects represent the unique entities of data that ever existed or will exist in our database, while the versions represent the specific values of the attributes of an object during a period of time. Versions represent the evolution of objects through time. The process side considers **events**, **instances**, and **processes**. Processes describe the behavior of the system. Instances are traces of execution for a given process, being sets of events ordered through time. These events represent the most fine-granular kind of execution data, denoting the occurrence of an activity or action at a certain point in time.

In Chapter 2, we already formalized the structure of a database. Definition 7 represents the existing structure of the source data schema. In this section, we present a more abstract and generic representation of databases and event data sources, in line with our meta-model. The idea is to provide a representation that adapts to our needs while abstracting from the specific database characteristics captured in Definition 7. Chapter 4 shows examples of application of the OpenSLEX meta-model to different data sources where the original database representation is transformed into our proposed meta model structure, and Appendix A presents a formalization of the mapping between each scenario (defined according to the definitions in Section 2.2) and the OpenSLEX meta-model.

The remainder of this section proposes a formalization of the elements in the OpenSLEX meta-model, starting from the data and continuing with the process side. As has been mentioned before, we can assume a way to classify elements in types or *classes* exists. Looking at our running example, we can distinguish between a *ticket* class and a *customer* class. This leads to the definition of data model as a way to describe the schema of our data.

Definition 16 (Data Model) *A data model is a tuple* $DM = (CL, AT, classOfAttribute, RS,$

$sourceClass, targetClass)$ *such that*

- *CL is a set of class names,*
- *AT is a set of attribute names,*
- *$classOfAttribute \in AT \to CL$ is a function that maps each attribute to a class,*
- *RS is a set of relationship names,*
- *$sourceClass \in RS \to CL$ is a function that maps each relationship to its source class,*
- *$targetClass \in RS \to CL$ is a function that maps each relationship to its target class.*

Each element belonging to a *Class* represents a unique entity, something that can be differentiated from the other elements of the same class, e.g., *Customer A* from *Customer B*. In a database setting, these elements would be table rows, and they would be uniquely identified by a unique key. Definition 9 provides a formalization of the collection of these entities in the general context of a database. In the context of OpenSLEX, we will call these unique entities *Objects*, and the collection of them an *Object collection*.

Definition 17 (Object Collection) *Let OBJ be the set of all possible objects. An object collection OC is a set of objects such that $OC \subseteq OBJ$.*

Something we know as well is that, during the execution of a process, the nature of *objects* can change over time. Modifications can be made on their attributes. Each of these represents *mutations* of an object, modifying the values of some of its attributes, e.g., modifying the address of a customer. As a result, despite being the same object, we will be looking at a different *version* of it. The notion of *object version* is, therefore, introduced to show the different stages in the life cycle of an *object*.

During the execution of a process, operations will be performed and, many times, links between elements are established. These links allow relating *tickets* to *concerts* or *customers* to *bookings*, for example. These relationships are of a structured nature and usually exist at the data model level, being defined between *classes*. Therefore, we know upfront that elements of the *ticket* class can be related somehow to elements of the *concert* class. *Relationships* is the name we use to call the definition of these links at the data model level. However, the actual instances of these *relationships* appear at the *object version* level, connecting specific versions of objects during a specific period of time. These specific connections are called *relations*.

Definition 18 (Version Collection) *Let V be some universe of values, TS a universe of timestamps and $DM = (CL, AT, classOfAttribute, RS, sourceClass, targetClass)$ a data model. A version collection is a tuple $OVC = (OV, attValue, startTimestamp, endTimestamp, REL)$ such that*

- *OV is a set of object version identifiers,*

- $\boldsymbol{attValue} \in (AT \times OV) \not\to V$ *is a function that maps a pair of object version and attribute to a value,*
- $\boldsymbol{startTimestamp} \in OV \to TS$ *is a function that maps each object version to a start timestamp,*
- $\boldsymbol{endTimestamp} \in OV \to TS$ *is a function that maps each object version to an end timestamp such that* $\forall \boldsymbol{ov} \in OV : \boldsymbol{endTimestamp}(\boldsymbol{ov}) \geq \boldsymbol{startTimestamp}(\boldsymbol{ov})$*,*
- $REL \subseteq (RS \times OV \times OV)$ *is a set of triples relating pairs of object versions through a specific relationship.*

At this point, it is time to consider the *process* side of the meta-model. The most basic piece of information we can find in a process event log is an event. Events are defined by some attributes. For convenience purposes we provide shorthands for the most typical attributes like *timestamp*, *resource*, and *life-cycle.*

Definition 19 (Event Collection) *Let V to be some universe of values and TS a universe of timestamps. An event collection is a tuple* $EC = (EV, EVAT, \boldsymbol{eventAttributeValue}, \boldsymbol{eventTimestamp}, \boldsymbol{eventLifecycle}, \boldsymbol{eventResource})$
such that

- *EV is a set of event identifiers,*
- *EVAT is a set of event attribute names,*
- $\boldsymbol{eventAttributeValue} \in \left(EV \times EVAT\right) \not\to V$ *is a function that maps a pair of an event and event attribute name to a value,*
- $\boldsymbol{eventTimestamp} \in EV \to TS$ *is a function that maps each event to a timestamp,*
- $\boldsymbol{eventLifecycle} \in EV \to \{\boldsymbol{start}, \boldsymbol{complete}, \ldots\}$ *is a function that maps each event to a value for its life-cycle attribute. If an event* $\boldsymbol{e} \in EV$ *does not have a value for the life-cycle attribute, then* $\boldsymbol{eventLifecycle}(\boldsymbol{e}) = \perp (\boldsymbol{nullvalue})$*,*
- $\boldsymbol{eventResource} \in EV \to V$ *is a function that maps each event to a value for its resource attribute. If an event* $\boldsymbol{e} \in EV$ *does not have a value for the resource attribute, then* $\boldsymbol{eventResource}(\boldsymbol{e}) = \perp (\boldsymbol{nullvalue})$*.*

When we consider events of the same activity but relating to a different life cycle, we gather them under the same *activity instance.* For example, two events that belong to the activity *make booking* could have different *life-cycle* values, being *start* the one denoting the beginning of the operation (first event) and *complete* the one denoting the finalization of the operation (second event). Therefore, both events belong to the same *activity instance.* Each of these activity instances can belong to different *cases* or *traces.* At the same time, *cases* can belong to different *logs*, which represent a complete set of *traces* on the behavior of a process.

Definition 20 (Instance Collection) *An instance collection is a tuple* $IC = (AI, CS, LG, aisOfCase, casesOfLog)$ *such that*

- AI *is a set of activity instance identifiers,*
- CS *is a set of case identifiers,*
- LG *is a set of log identifiers,*
- $aisOfCase \in CS \rightarrow \mathscr{P}(AI)$ *is a function that maps each case to a set of activity instances,*
- $casesOfLog \in LG \rightarrow \mathscr{P}(CS)$ *is a function that maps each log to a set of cases.*

The last piece of our meta-model is the *process model collection.* This part stores *process models* on an abstract level, i.e., as sets of *activities*, ignoring details about the control flow or how these activities relate between them. An *activity* can belong to different *processes* at the same time.

Definition 21 (Process Model Collection) *A process model collection is a tuple* $PMC = (PM, AC, actOfProc)$ *such that*

- PM *is a set of process identifiers,*
- AC *is a set of activity identifiers,*
- $actOfProc \in PM \rightarrow \mathscr{P}(AC)$ *is a function that maps each process to a set of activities.*

Now that we have all the pieces of our meta-model, it is still necessary to wire them together. A *connected meta-model instance* defines the logical relations between these blocks. Therefore, we see that *versions* belong to *objects* ($objectOfVersion$) and *objects* belong to a class ($classOfObject$). In the same way, *events* belong to *activity instances* ($eventAI$), *activity instances* belong to *activities* ($activityOfAI$) and can belong to different *cases* ($aisOfCase$), *cases* belong to different *logs* ($casesOfLog$), and *logs* relate to processes ($processOfLog$). *Events* and *versions* are related ($eventToOVLabel$) in a way that can be interpreted as a causal relation, i.e., when *events* happen they trigger the creation of *versions* as a result of modifications on data (the update of an attribute for instance). Another possibility is that the event represents a read access or query of the values of a *version.*

Definition 22 (Connected Meta-Model Instance) *A connected meta-model instance is a tuple* $CMI = (DM, OC, classOfObject, OVC, objectOfVersion, EC, eventToOVLabel, IC, eventAI, PMC, activityOfAI, processOfLog)$ *such that*

- $DM = (CL, AT, classOfAttribute, RS, sourceClass, targetClass)$ *is a data model,*
- OC *is an object collection,*

- $classOfObject \in OC \to CL$ *is a function that maps each object to a class,*
- $OVC = (OV, attValue, startTimestamp, endTimestamp, REL)$ *is a version collection,*
- $objectOfVersion \in OV \to OC$ *is a function that maps each object version to an object,*
- $EC = (EV, EVAT, eventAttributeValue, eventTimestamp, eventLifecycle, eventResource)$ *is an event collection,*
- $eventToOVLabel \in (EV \times OV) \not\rightarrow V$ *is a function that maps pairs of an event and an object version to a label. If* $(ev, ov) \in dom(eventToOVLabel)$, *this means that both event and object version are linked. The label itself defines the nature of such link, e.g "insert", "update", "read", "delete", etc.,*
- $IC = (AI, CS, LG, aisOfCase, casesOfLog)$ *is an instance collection,*
- $eventAI \in EV \to AI$ *is a function that maps each event to an activity instance,*
- $PMC = (PM, AC, actOfProc)$ *is a process model collection,*
- $activityOfAI \in AI \to AC$ *is a function that maps each activity instance to an activity,*
- $processOfLog \in LG \to PM$ *is a function that maps each log to a process.*

The previous definition defines a connected meta-model instance (*CMI*). However, we still need to define the validity of a *CMI*. Before doing so, we introduce some shorthands:

Definition 23 (Notation) *Let* $CMI = (DM, OC, classOfObject, OVC, objectOfVersion, EC, eventToOVLabel, IC, eventAI, PMC, activityOfAI, processOfLog)$ *be a connected meta-model instance. We define the following shorthands:*

- $E_{ai} = \{e \in EV \mid eventAI(e) = ai\}$ *is the set of events that belong to the activity instance* ai*,*
- $AI_{cs} = \{ai \in AI \mid ai \in aisOfCase(cs)\}$ *is the set of activity instances that belong to the case* cs*,*
- $OV_{obj} = \{ov \in OV \mid objectOfVersion(ov) = obj\}$ *is the set of object versions that belong to the object* obj*,*
- $OC_{cl} = \{obj \in OC \mid classOfObject(obj) = cl\}$ *is the set of objects that belong to the class* cl*,*
- $REL_{rs} = \{(rs', ov, ov') \in REL \mid rs' = rs\}$ *is the set of relations that belong to the relationship* rs*,*
- $AIAC_{ac} = \{ai \in AI \mid activityOfAI(ai) = ac\}$ *is the set of activity instances that belong to the activity* ac*.*

In order to consider a connected meta-model instance as valid, the data and connections in it must fulfill certain criteria set in the following definition:

Definition 24 (Valid Connected Meta-Model Instance) *Let* $CMI = (DM, OC, classOfObject, OVC, objectOfVersion, EC, eventToOVLabel, IC, eventAI, PMC, activityOfAI, processOfLog)$ *be a connected meta-model instance. A valid connected meta-model instance VCMI is a connected meta-model instance such that:*

- *$attValue$ is only defined for attributes of the same class of the object version, that is,* $(at, ov) \in domain(attValue) \iff classOfObject(objectOfVersion(ov)) = classOfAttribute(at)$*,*
- *none of the object versions of a same object overlap in time, that is,* $\forall obj \in OC : \forall ov_a \in OV_{obj} : \forall ov_b \in OV_{obj} : ov_a \neq ov_b \implies endTimestamp(ov_a) \leq startTimestamp(ov_b) \vee endTimestamp(ov_b) \leq startTimestamp(ov_a)$*,*
- *all the cases of a log contain only activity instances that refer to activities of the same process of the log, that is,* $\forall log \in LG : \forall cs \in casesOfLog(lg) : \forall ai \in AI_{cs} : activityOfAI(ai) \in actOfProc(processOfLog(lg))$*.*

Figure 3.4 depicts the entity-relation diagram of the meta-model as formalized above. Some elements of the meta-model have been omitted from the diagram for the sake of simplicity. A full version of the ER diagram is available online[3]. Each of the entities in the diagram, as represented by a square, corresponds to the basic *entities* of the meta-model as formalized in Definition 22. Also, these entities, together with their relations (diamond shapes), have been grouped in areas that we call *sectors* (delimited by dashed lines). These sectors are: *data models*, *objects*, *versions*, *events*, *cases*, and *process models*. These tightly related concepts provide an abbreviated representation of the meta-model. For the sake of clarity, the "sectorized" representation of the meta-model will be used in further parts of the thesis. As can be observed, the entity-relation diagram is divided into six sectors. The purpose of each of them is described below:

- **Data models**: this sector is formed by concepts needed to describe the structure of any database system. Many data models can be represented together in this sector, whose main element is the *data model* entity. For each data model, several *classes* can exist. These classes are abstractions of the more specific concept of table, which is commonly found in RDBMSs. Looking at the database provided in Section 3.2, the tables *customer* and *booking* are examples of classes. These classes contain *attributes*, which are equivalent to table columns in modern databases (e.g., *id*, *name*, *address*, etc.). The references between classes of the same data model are represented with the *relationship* entity. This last entity holds links between a source and a target class (e.g., *booking_customer_fk* which relates the source class *booking* to the target class *customer*).

[3] `https://github.com/edugonza/OpenSLEX/blob/master/doc/meta-model.png`

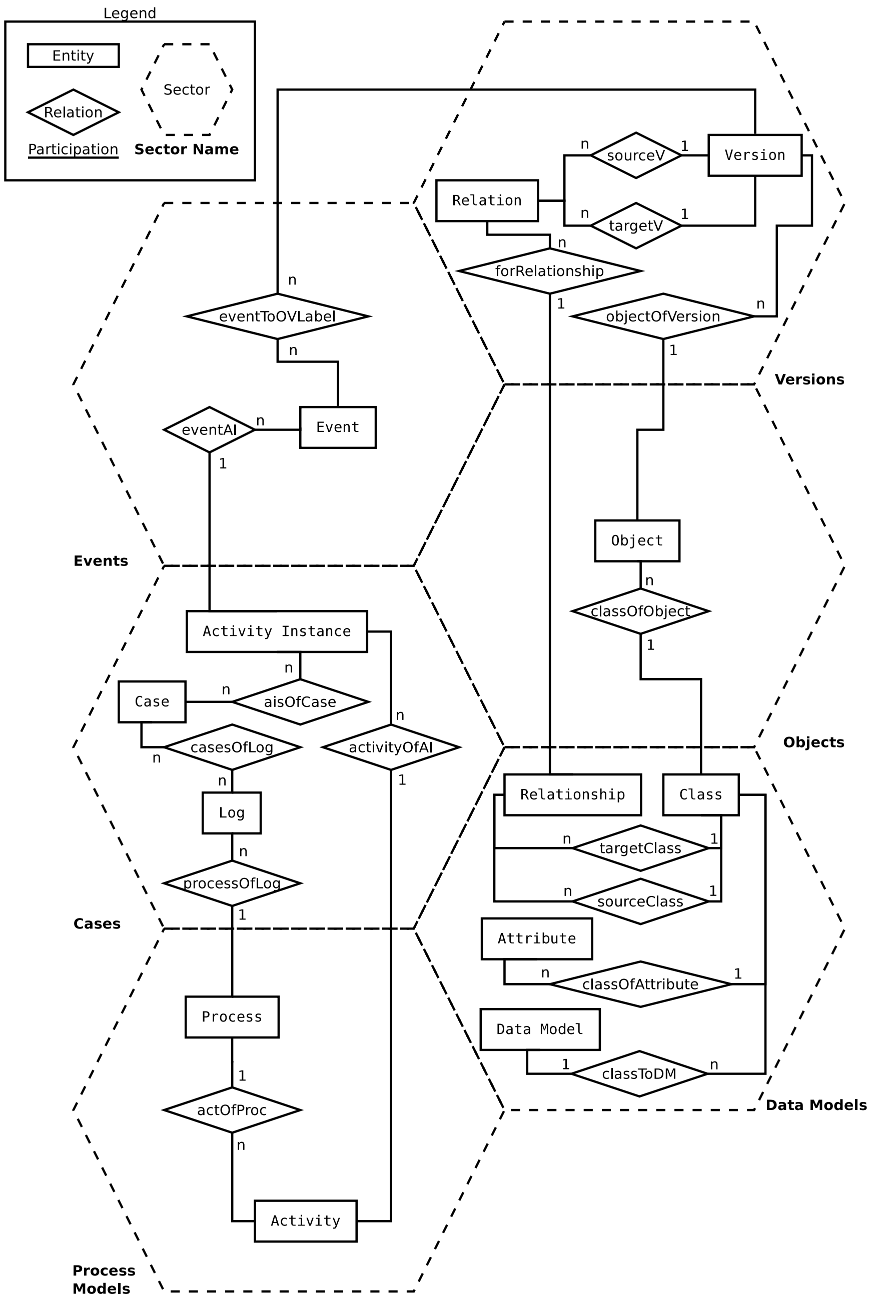


Figure 3.4: ER diagram of the OpenSLEX meta-model. The entities have been grouped into sectors, delimited by the dashed lines.

- **Objects**: the *object* entity, part of the objects sector, represents each of the unique data elements that belong to a class. An example of this can be a hypothetical customer with *customer_id* = 75. Additional details of this object are omitted, given that they belong to the next sector.

- **Versions**: for each of the unique object entities described in the previous sector, one or many *versions* can exist. A version is an instantiation of an object during a certain period of time, e.g., the customer object with id 75, existed in the database, during a certain period of time, for example from "2015-08-01 14:45:00" to "2016-09-03 12:32:00". During that period of time, the object had specific values for the attributes of the customer class that it belongs to. Therefore, there is a version of customer 75, valid between the mentioned dates, with name "John Smith", address "45, 5th Avenue", and birth date "1990-01-03". If at some point, the value of one of the attributes changed (e.g., a new address), the end timestamp of the previous version would be set to the time of the change, and a new version would be created with the updated value for that attribute, and a start timestamp equal to the end of the previous version, e.g., version_-1 = {object_id = 75, name = "John Smith", address = "45, 5th Avenue", birth_date = "1990-01-03", start_timestamp = "2015-08-01 14:45:00", end_-timestamp = "2016-09-03 12:32:00"}, and version_2 = {object_id = 75, name = "John Smith", address = "floor 103, Empire State Building", birth_date = "1990-01-03", start_timestamp = "2016-09-03 12:32:00", end_timestamp = NONE }. Note that the value of *end_timestamp* for the newly created object version (*version_2*) is NONE. That means that it is the current version for the corresponding object (*object_id* = 75). Another entity reflected in this sector is the concept of *relation*. A relation is an instantiation of a relationship, and holds a link between versions of objects that belong to the source and target classes of the relationship. For example, a version of a booking object can be related to another version of a customer object by means of a relation instance, given that a relationship (*booking_customer_fk*) exists from class *booking* to class *customer*.

- **Events**: this sector collects a set of events, obtained from any available source (database tables, redo-logs, change records, system logs, etc.). In this sector, events appear as a collection, not grouped into traces (such grouping is reflected in the next sector). In order to keep process information connected to the data side, each event can be linked to one or many object versions by means of a label (*eventToOVLabel*). This label allows specifying what kind of interaction exists between the event and the referred object version, e.g., *insert*, *update*, *delete*, *read*, etc. Events hold details such as *timestamp*, *life-cycle*, and *resource* information, apart from an arbitrary number of additional event attributes.

- **Cases and instances**: the entities present in this sector are very important from the process mining point of view. The events by themselves do not provide much information about the control flow of the underlying process, unless they are correlated and grouped into traces (or cases). First, the *activity instance*

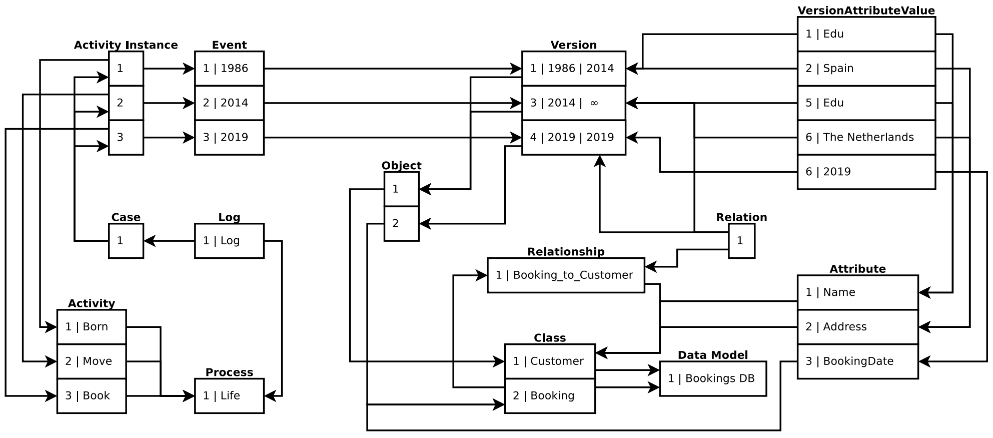


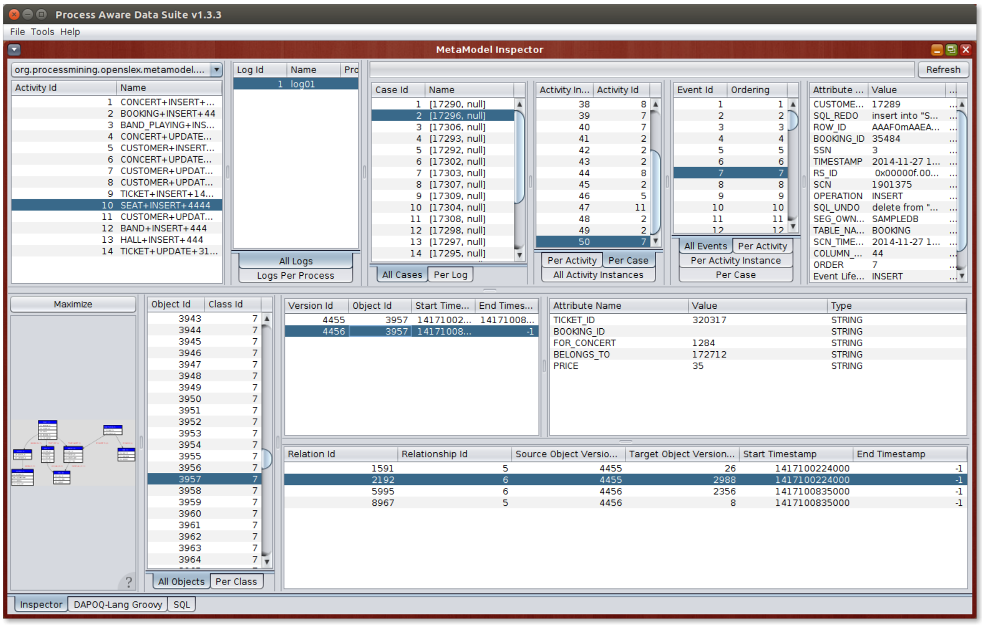
Figure 3.5: Diagram of an instance of the OpenSLEX meta-model.

entity should be explained. This entity is used to group events that refer to the same instance of a certain activity with different values for its life-cycle, e.g., the execution of the activity *book_tickets* generates one event for each phase of its life-cycle: *book_tickets+start*, and *book_tickets+complete*. Both events, referring to the same execution of an activity, are grouped into the same activity instance. Next, as in any other event log format, activity instances can be grouped in *cases*, and these cases, together, form a *log*.

- **Process models**: the last sector contains information about *processes*. Several processes can be represented in the same meta-model. Each process is related to a set of *activities*, and each of these activities can be associated with several activity instances, contained in the corresponding *cases and instances* sector.

An instantiation of this meta-model fulfills the requirements set in Section 3.3.1 in terms of storage of data and process view. Some characteristics of this meta-model that enable full compatibility with the XES standard have been omitted in this formalization for the sake of brevity.

Figure 3.5 shows an example of an instance of the OpenSLEX meta-model. For the sake of clarity the model has been simplified, but the main structure remains. We see that there is a global *data model*. All the *classes* belong to it: “Customer” and “Booking”. Also, there are three *attributes*: “Name”, “Address”, and “BookingDate”. The first two attributes belong to the class “Customer”. The third one belongs to “Booking”. There is a *relationship* connecting bookings to customers named “Booking_to_Customer”. Two *objects* exist. The first object has two *versions*. Each version of the customer object has *values* for the corresponding attributes. We see that the first customer version corresponds to a customer named “Edu” while he lived in “Spain”, from 1986 to 2014. The second version corresponds to the same customer, while he lived in “The Netherlands” from 2014 until the present. There is another object version that belongs to the second object, a booking object. The “BookingDate”



Activity Id	Name
1	CONCERT+INSERT+...
2	BOOKING+INSERT+44
3	BAND_PLAYING+INS...
4	CONCERT+UPDATE...
5	CUSTOMER+INSERT...
6	CONCERT+UPDATE...
7	CUSTOMER+UPDAT...
8	CUSTOMER+UPDAT...
9	TICKET+INSERT+14...
10	SEAT+INSERT+4444
11	CUSTOMER+UPDAT...
12	BAND+INSERT+444
13	HALL+INSERT+444
14	TICKET+UPDATE+31...

Log Id	Name	Pro
1	log01	

Case Id	Name
1	[17290, null]
2	[17296, null]
3	[17306, null]
4	[17293, null]
5	[17292, null]
6	[17302, null]
7	[17303, null]
8	[17307, null]
9	[17309, null]
10	[17304, null]
11	[17308, null]
12	[17298, null]
13	[17297, null]
14	[17295, null]

Activity In...	Activity Id
38	8
39	7
40	7
41	2
42	2
43	2
44	8
45	2
46	5
47	11
48	2
49	2
50	7

Event Id	Ordering
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12

Attribute ...	Value	...
CUSTOME...	17289	...
SQL_REDO	insert into "S...	...
ROW_ID	AAAFOmAAEA...	...
BOOKING_ID	35484	...
SSN	3	...
TIMESTAMP	2014-11-27 1...	...
RS_ID	0x00000f.00...	...
SCN	1901375	...
OPERATION	INSERT	...
SQL_UNDO	delete from "...	...
SEG_OWN...	SAMPLEDB	...
TABLE_NA...	BOOKING	...
SCN_TIME...	2014-11-27 1...	...
COLUMN_...	44	...
ORDER	7	...
Event Life...	INSERT	...

Object Id	Class Id
3943	7
3944	7
3945	7
3946	7
3947	7
3948	7
3949	7
3950	7
3951	7
3952	7
3953	7
3954	7
3955	7
3956	7
3957	7
3958	7
3959	7
3960	7
3961	7
3962	7
3963	7
3964	7

Version Id	Object Id	Start Time...	End Times...
4455	3957	14171002...	14171008...
4456	3957	14171008...	-1

Attribute Name	Value	Type
TICKET_ID	320317	STRING
BOOKING_ID		STRING
FOR_CONCERT	1284	STRING
BELONGS_TO	172712	STRING
PRICE	35	STRING

Relation Id	Relationship Id	Source Object Versio...	Target Object Version...	Start Timestamp	End Timestamp
1591	5	4455	26	1417100224000	-1
2192	6	4455	2988	1417100224000	-1
5995	6	4456	2356	1417100835000	-1
8967	5	4456	8	1417100835000	-1

Figure 3.6: Screenshot of the meta-model inspector tool

value of this version is “2019”. There is a *relation* (an instance of the relationship “Booking_to_Customer”), that connects the second object version of customer *1* to the first object version of booking *1*. On the left side of the figure, we see that three *events* exist. The first event, related to the first version of customer *1*, is linked to the *activity* “Born”, and happened in 1986. The second event, linked to the activity “Move”, happened in 2014 and is related to the second version of the same customer. Finally, the third event is linked to the activity “Book”, and is linked to the first version of booking *1*. Each event belongs to its own *activity instance*. All activity instances belong to one *case*. This case belongs to a *log* of the *process* “Life”.

The meta-model previously formalized has been implemented. This was required in order to provide tools that assist in the exploration of the information contained within the populated meta-model. More details on this implementation are explained in the following section.

3.4 Implementation

The Open SQL Log Exchange (OpenSLEX[4]) library, based on the meta-model proposed in this thesis, has been implemented in Java. This library provides an interface to insert data in an instantiation of this meta-model and to access it in a similar way to how XES Logs are managed by the IEEE XES Standard [1]. The OpenSLEX

[4]`https://github.com/edugonza/openslex`

implementation relies on SQL technology. Specifically, the populated meta-model is stored in a SQLite[5] file. This provides some advantages, like an SQL query engine, a standardized format, as well as storage in self-contained single data files that benefits its exchange and portability. Figure 3.4 shows an ER diagram of the internal structure of the meta-model. However, it represents a simplified version to make it more understandable and easy to visualize. The complete class diagram of the meta-model can be accessed in the OpenSLEX's website. In addition to the library mentioned earlier, an inspector has been included in the Process-Aware Data Suite (PADAS)[6] tool. This inspector, depicted in Figure 3.6, allows exploring the content of OpenSLEX files by means of a GUI in an exploratory fashion, which lets the user dig into the data and apply some basic filters on each element of the structure. The tool presents a series of blocks that contain the *activities*, *logs*, *cases*, *activity instances*, *events*, *event attribute values*, *data model*, *objects*, *object versions*, *object version attribute values* and *relations* entities in the meta-model. Some of the lists in the inspector (*logs*, *cases*, *activity instances*, *events* and *objects*) have tabs that allow one to filter the content they show. For instance, if the tab *"Per Activity"* in the *cases* list is clicked, only cases that contain events of such activity will be shown. In the same way, if the tab *"Per Case"* in the *events* list is clicked, only events contained in the selected case will be displayed. An additional block in the tool displays the attributes of the selected event.

The goal of providing these tools is to assist in the task of populating the proposed meta-model, in order to query it in a later step. Because the meta-model structure and the data inserted into it are stored in a SQLite file, it is possible to execute SQL queries in a straightforward way. In particular, the whole process of extracting, transforming, and querying data has been implemented in a RapidProM[7] workflow. RapidProM is an extension to the well-known data mining workflow tool RapidMiner[8] that adds process mining plugins from ProM. For our task, an OpenSLEX meta-model population operator has been implemented in a development branch[9] of RapidProM. This operator, together with the data handling operators of RapidMiner (including database connectors), lets us extract, transform, and query our populated meta-model automatically in a single execution of the workflow. More details on this workflow and the extraction and transformation steps are provided in Section 4.2, showing how the technique can be applied in different real-life environments. A workflow was designed for each of the evaluation environments with the purpose of extracting and transforming the content of the corresponding databases to fit the structure of the OpenSLEX meta-model. Each of these workflows, also referred to as adapters, are extensible using the collection of operators available in RapidMiner, RapidProM, and other plugins. They can be modified to fit new environments. Moreover, Section 4.3 demonstrates how querying of the resulting OpenSLEX populated meta-model can be standardized for all the proposed environments.

[5]`http://www.sqlite.org/`

[6]`https://github.com/edugonza/PADAS`

[7]`http://rapidprom.org/`

[8]`http://rapidminer.com/`

[9]`https://github.com/rapidprom/rapidprom-source/tree/egonzalez`

The provided implementation of the OpenSLEX meta-model is a general and extensible platform for data extraction and transformation. It provides the means to abstract raw data to high-level concepts. These concepts, (events, cases, objects, etc.) are easy to deal with for business analysts. On the other hand, the current implementation presents certain limitations. A general method does not exist at the moment to perform this extraction and transformation independently of the raw data structure. This means that an ad-hoc adapter needs to be designed for different data architectures (SAP data schema, Oracle redo-logs, in-table versioning, etc.) in order to properly extract and interpret events and object versions. However, we believe that the three adapters provided with the implementation should suffice in most cases and can serve as a template for other environments. It is important to note that, despite the need to design an extraction adapter for each type of environment, it provides a notable advantage with respect to writing ad-hoc queries for event log extraction. This is due to the fact that, once the extraction and transformation to OpenSLEX are performed, automated methods (implemented as operators in the RapidProM platform) become available for generating multiple event logs from different perspectives in a single dataset. This saves time and effort. Also, this approach is less prone to errors than designing ad-hoc queries for each specific event log perspective.

3.5 Related Work

Despite the efforts made by the community, we can identify a few areas in which the problem of modeling, collecting, and analyzing process execution data remains a challenge. Business process management and workflow management are areas in which the community has focused on providing models to describe their functioning and make the analysis of their behavior possible. Papers like [98, 130] propose meta-models to provide structure to audit trails on workflows. However, they focus mainly on the workflow or process perspective and leave out the database side.

Process mining techniques require event data as input. In order to improve the exchange of event data between systems and tools, it was necessary to define a standard way to store event logs. In [113], the authors provide a meta-model to define event logs, which would evolve later in the IEEE XES Standard format [1]. This format represents a great achievement from the standardization point of view. Also, it allows exchanging logs and developing mining techniques, assuming a common representation of the data. However, the current definition of XES is not able to effectively and efficiently store the data perspective of our meta-model. From our point of view, the XES format is, in fact, a target format and not a source of information. We aim at generating from a richer source different views on data in XES format to enable process mining.

Artifact-centric approaches provide a point of view on event logs that gives more relevance to the data entities in business management systems. Data artifacts are identified within business process execution data, in order to discover how changes and updates affect their life-cycle [92]. Techniques like the one proposed in [72] are able to combine the individual life-cycles of different artifacts to show the interrelation of the different steps. However, a limitation of these techniques is the difficulty to interpret correctly the resulting models, in which clear execution semantics are not always available. The application of other techniques, like conformance and performance analysis, to the resulting models has not been solved yet.

In the field of conceptual modeling and formal ontologies, we find several studies that explore different aspects of data modeling in business scenarios. In [47, 48] *relationship reification* is discussed from a ontological point of view. An ongological analysis of relations and relationships is presented, and a general theory of reification and truthmaking is proposed. Our meta-model, OpenSLEX, closely relates to such concepts of relation and relationship, since relationships must exist to give meaning and truth to relations found between data elements. The work in [49] explores the ontological nature of events and object-like entities (*endurants*) in business models. Endurants are entities that can qualitatively change in certain respects while maintaining their identity. The connection of this work with our meta-model is very interesting since the concept of object in OpenSLEX matches the description of an endurant, and its evolution through time is described by means of object versions and events. Finally, [46] discusses the temporal characteristics of future and ongoing events. It proposes a theory by which events shall not be considered as instantaneous, i.e., *frozen in time*, but rather changing elements. In our meta-model, we adopt a more traditional concept of events that happen instantaneously at a certain moment in time. However, we support long lasting occurrences by means of *activity instances*, which can group several events indicating changes in the life-cycle during the execution of a certain activity.

Data warehousing is an area in which there is a great interest to solve the problem of gathering business process information from different sources. Work has been done on this aspect [28, 89, 127–129], proposing ways to centralize the storage of heterogeneous data, with the purpose of enabling process monitoring and workflow analysis. However, some of these approaches, besides being process-oriented, do not allow for the application of process mining analysis. This is due to the lack of event data, where measures of process performance are stored instead. On the other hand, some approaches allow us to store more detailed information [87], but they force an ad-hoc structure only relevant for the process at hand. The main distinction between our approach and existing work on data warehousing for process mining is the generality and standardization of our meta-model, which is independent of the specific process, while combining both data and process perspectives.

Process cubes [107] are techniques, closely related to data warehousing, that aim at combining aspects of multidimensional databases (OLAP cubes) with event logs, in order to enable the application of process mining techniques. Related work has been presented in [9, 117–119]. These approaches allow one to slice, dice, roll up, and drill down event data using predefined categories and dimensions. It represents a great improvement for the analysis of multidimensional data with respect to traditional filtering. However, these techniques still rely on event data as their only source of information. Also, they require any additional data to be part of each individual event. Therefore, given that it is impossible to efficiently represent in an event log all the aspects we cover in our meta-model, process cubes as they are right now, do not represent an alternative for our purposes.

SAP Process Observer is a component of the wide spectrum of SAP utilities. It makes it possible to monitor the behavior of some SAP Business Suite processes (e.g., order to cash). This component monitors Business Object (BO) events and creates logs correlating them. The tool provides insights such as deviation detection, real-time exception handling, service level agreement tracking, etc. These event logs can be used as well to perform other kinds of analytics. One of the main advantages of this solution is the fact that it enables real-time monitoring, without the need to deal with raw data. However, it needs to be configured specifying the activities that must be monitored. The kind of logs that it generates is not different from the event logs provided by other tools or techniques, in the sense that they use the same structure, having events grouped in sets of traces, and data only represented as plain attributes of events, traces, and logs. Also, it lacks genericity, being only applicable in SAP environments to monitor Business Suite processes.

Nowadays, several commercial process mining tools are available in the market. These tools import either XES event logs (e.g., Disco[10]) or event tables (e.g., Celonis[11]), supporting attributes at the event, trace, and log levels. Most of them provide advanced filtering features based on these attributes, as is the case for Disco. Additionally, Celonis has two interesting features. First, it implements OLAP capabilities, making it possible to perform some operations like slicing and dicing event data. Second, Celonis provides its own language to express formulas. These formulas are used to compute statistics and key performance indicators (KPIs), as well as to express complex conditions for OLAP operations. All these features are powerful tools to work on data. However, they are restricted by the flat structure of their input event log format. It is not possible to keep complex data relations in such a restricted representation. These complex relations are needed to perform advanced data querying, cf. query GQ in Section 4.3.1, where relevant event data must be selected based on the evolution of database objects of a specific class.

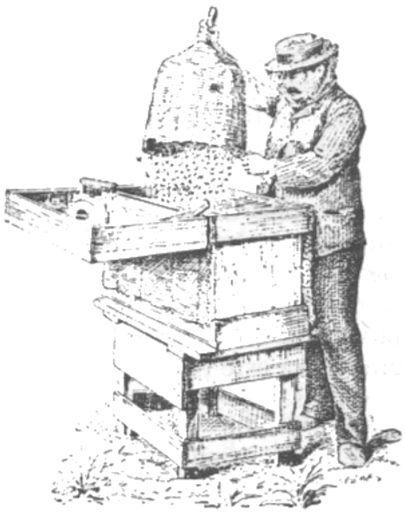
In general, the main weakness of most of the approaches discussed resides in the way they force the representation of complex systems by means of a flat event log. The data perspective is missing, which only allows one to add attributes at the event, trace or log level. More recent works try to improve the situation, analyzing data dependencies [80] in business models with the purpose of improving them or even observing object-state changes to improve their analysis [51]. However, none of the existing approaches provides a generic and standard way of gathering, classifying, storing, and connecting process and data perspectives on information systems, especially when dealing with databases where the concept of structured process can be fuzzy or nonexistent.

[10]`https://fluxicon.com/disco/`

[11]`https://www.celonis.com/en/`

3.6 Chapter Summary

In this chapter, a meta-model has been proposed that provides a way to capture a more descriptive image of the reality of business processes. This meta-model aligns data and process perspectives and enables the application of existing process mining techniques. At the same time, it unleashes a new way to query data and historical information. This is possible thanks to the combination of data and process perspectives in a rich format. The data are transformed in order to be ready for multi-perspective analysis of information systems, while allowing one to use existing process mining techniques that still require a flat event log as an input. The next chapter presents the application of the OpenSLEX meta-model for the data extraction and querying on three examples of real-life database environments.



Beekeeper introducing a swarm into a hive from the top.

"Cours complet d'apiculture", Georges de Layens and Gaston Bonnier, 1897

4 OpenSLEX in Practice: Data Extraction and Querying

In the previous chapter, we proposed a meta-model for process mining on databases. This data model allows us to capture process (processes, logs, events, etc.) and data aspects (data models, objects, object versions, etc.) in a single structure, enabling multi perspective log building and standardized event data querying regardless of the source of data. This chapter evaluates the applicability of this meta-model to three different real-life environments, showing details of the extraction, transformation, and load process. Also, we show how query templates can be applied to answer a similar question for three very different datasets.

4.1 Introduction

Data extraction and preparation are among the first steps to take in any business intelligence or data analysis project (steps 2 and 3 in Figure 1.1). It is a very time-consuming task, that can take up to 80% of the total project duration. This is due to the fact that the original sources of data come in a great variety, differing in structure depending on the nature of the application or process under study. The standardization of this phase represents a challenge, given that a lot of domain knowledge is usually required in order to carry it out. It is because of this that most of the work is done by hand, in an ad-hoc fashion, requiring a lot of iterations in order to obtain the proper data in the right form.

In process mining, the situation is not much different. Studies have been carried out, focusing on SAP [55,97,100] or ERP systems in general [34,125]. Also, efforts have been made to achieve a certain degree of generalization with the tool XESame [116], which assists in the task of defining mappings between database fields on the one side, and events, traces and logs on the other. However, these solutions, which we refer to as part of the classical or traditional approach, are tightly coupled to the specific IT system or data schema they were designed to analyze. Moreover, they do not support the extraction of event data from systems that are not process-aware or do not explicitly record historical information.

Other techniques exist that try to leverage on the existence of alternative sources of data. A very promising approach is redo-log process mining [44]. This technique takes advantage of the redo-log mechanism that exists in most modern relational database management systems (RDBMSs), to extract events that represent data changes, and be able to build event logs even on systems that are not process-aware. In some other environments, a functionality similar to redo-logs is implemented at the application level. SAP is an example of such an environment, which records data changes performed on certain tables. These tables are tracked for data modifications, keeping a record in what is called *change log* tables. Sometimes, none of the previous situations apply. In that case, events must be extracted in an ad-hoc manner.

The remainder of this chapter is structured as follows: Section 4.2 evaluates the application of ETL on three different environments, where the data are transformed to conform with the meta-model proposed in Chapter 3. The results of the evaluation and the querying of the output of our approach are analyzed in Section 4.3. Finally, Section 4.4 presents the conclusions.

4.2 Evaluation in Real-life Environments

The development of the meta-model presented in Chapter 3 has been partly motivated by the need of a general way to capture the information contained in different systems combining the data and process views. Such systems, usually backed up by a database, use very different ways to internally store their data. Therefore, in order to extract these data, it is necessary to define a translation mechanism tailored to the wide variety of such environments. Because of this, *the evaluation aims at demonstrating the feasibility of transforming information from different environments to the proposed meta-model*. Specifically, three real-life source environments are analyzed:

1. *Database redo-logs*: files generated by the RDBMS in order to maintain the consistency of the database in case of failure or rollback operations. The data were obtained from real redo-logs generated by a running Oracle database instance. A synthetic process was designed and run in order to insert and modify data in the database and trigger the generation of redo-logs. Because of the clarity of the data model, this environment inspired the running example of Section 5.2.

2. *In-table version storage*: Application-specific schema to store new versions of objects as a new row in each table. The data of this analysis were obtained from

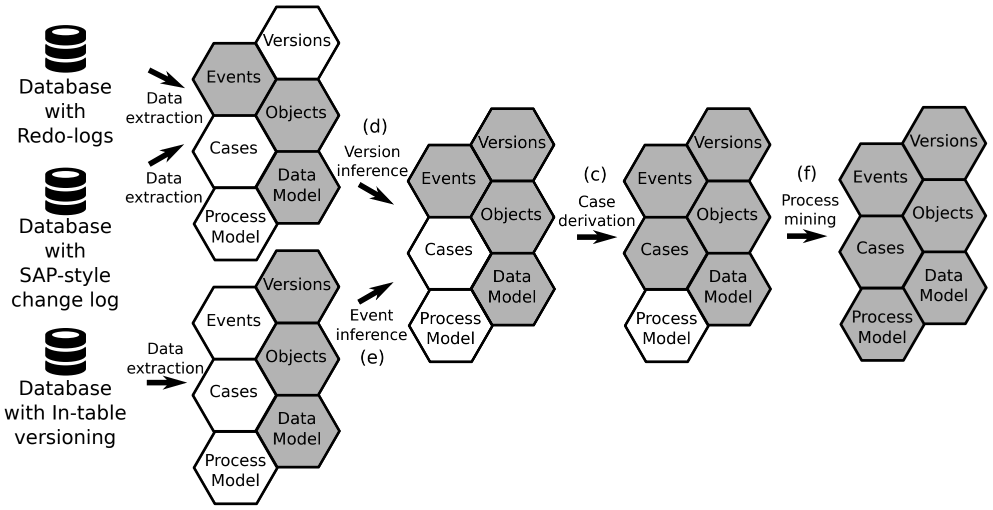


Figure 4.1: Meta-model completion in the three evaluated environments

a real-life instance of a Dutch financial organization.

3. *Change tables*: changes in tables are recorded in a "redo-log" style as a separate table, the way it is done in SAP systems. For this analysis, real SAP data, generated by an external entity, were used. Such data are often used for training purposes by the third party organization.

The benefit of transforming data to a common representation is that it allows for decoupling the application of techniques for the analysis from the sources of data. In addition, a centralized representation allows linking data from different sources. However, the source of data may be incomplete in itself. In some real-life scenarios, explicitly defined events might be missing. In other scenarios, there is no record of previous object versions. Something common is the lack of a clear case notion, which causes the absence of process instances. In all these cases, it is necessary to apply some automated inference techniques to derive the missing information and create a complete and fully integrated view. Figure 4.1 shows these environments. It also shows which sectors of our meta-model can be populated right away, only extracting what is available in the database. Next, following a series of automated steps like *version inference* and *event inference*, *case derivation*, and *process discovery*, all the sectors can be populated with inferred or derived data.

The first part of this evaluation (Section 4.2.1) presents the different scenarios that we can find when transforming data. Each of these scenarios starts from data that correspond to different sectors of the meta-model. Next, we show how to derive the missing sectors from the given starting point. Sections 4.2.2, 4.2.3, and 4.2.4 analyze the three real-life common environments mentioned before. We will demonstrate that data extraction is possible and that the meta-model can be populated from these different sources. Additional formalizations describing the mapping between these three

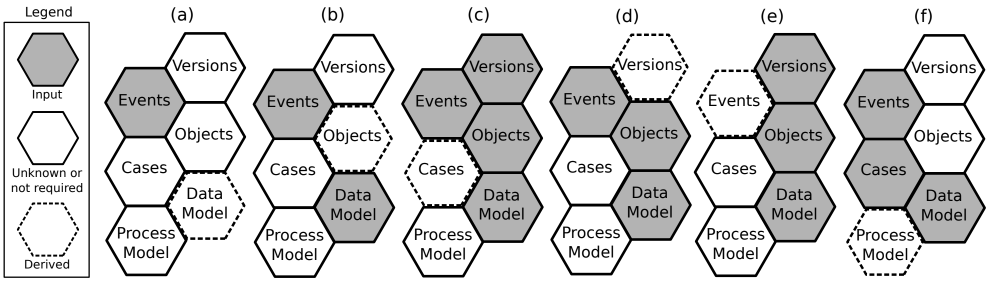


Figure 4.2: Input scenarios to complete meta-model sectors

real-life environments and the OpenSLEX meta-model can be found in Appendix A. Section 4.2.5 presents a method to merge data from two different systems into a single meta-model structure, providing an end-to-end process view. Section 4.3 demonstrates that, starting from the three resulting populated meta-models, it is possible to standardize the process mining analysis and perform it automatically, adapting only a few parameters specific to each dataset. In addition to that, an example of the corresponding resulting populated meta-model for each of the environments is shown.

4.2.1 Meta-Model Completion Scenarios

Experience teaches us that it is not common to find an environment that explicitly provides the information to fill every *sector* of our meta-model. This means that additional steps need to be taken to evolve from an incomplete meta-model to a complete one. Figure 4.2 presents several scenarios in which, starting from a certain input (gray sectors), it is possible to infer the content of other elements (dashed sectors). Depending on the starting point that we face, we must start inferring the missing elements consecutively in the right order, which will lead us, in the end, to a completely populated meta-model:

a) *Schema discovery*: One of the most basic elements we require in our meta-model to be able to infer other elements is the *events sector*. Starting from this input and applying schema, primary key, and foreign key discovery techniques [101, 126], it is possible to obtain a data model describing the structure of the original data.

b) *Object identification*: If the events and a data model are known, we can infer the objects that these events represent. It is necessary to know the attributes of each class that identify the objects (primary keys). Finding the unique values for such attributes in the events corresponding to each class results in the list of unique objects of the populated meta-model.

c) *Case derivation*: Another possible scenario is the one in which we derive cases from the combination of events and a data model. The event splitting technique described in [44], which uses the transitive relations between events defined by the data model, allows generating different sets of cases, or event logs. This requires, similar to scenario *b*, to match the attributes of each event to the attributes of each class. Next, we must use the foreign key relations to correlate events between them. Selecting the desired combination of relationships between events will tell us how to group these events in cases to form a log. A more general approach was proposed in [39] (Chapter 5), where we proposed a definition of case notion and a method to generate multiple event logs as different perspectives on the same data.

d) *Version inference*: The events of each object can be processed to infer the *object versions* as results of the execution of each event. Events must contain the values of the attributes of the object they relate to at a certain point in time or, at least, the values of the attributes that were affected (modified) by the event. Next, ordering the events by (descending) timestamp and applying them to the last known value of each attribute allows us to reconstruct the versions of each object.

e) *Event inference*: The inverse of scenario *d* is the one in which events are inferred from object versions. By looking at the attributes that differ between consecutive versions it is possible to create the corresponding event for the modification.

f) *Process mining analysis*: Finally, a set of cases, or an event log, can be used to apply different kinds of process mining techniques such as process discovery, performance analysis, conformance analysis, prediction, model repair, etc.

Figure 4.1 provides an overview of the extraction and transformation process applied to each environment. We see that two alternative paths exist in order to obtain a populated meta-model instance, depending on the data source. The first path (top left of Figure 4.1) starts with the basic *extraction* step, in which data model, objects and explicitly defined events are extracted and stored in an OpenSLEX instance. Next, *version inference* (scenario *c*) is applied to obtain object versions from the extracted objects and events. The second path (bottom left of Figure 4.1) extracts data model, objects and object versions from the original data source. Next, *event inference* is applied (scenario *e*), in order to obtain event data from object versions. At that point, the data from all the environments have been transformed into a common representation. If our purpose is to apply existing process mining techniques (scenario *f*), first we need to build event logs that can be analyzed. This can be achieved by means of the *case derivation* step. In this step, we need to correlate events into cases to build event logs. There are many ways in which events can be grouped, each of them providing a different perspective on the same data. Chapter 5 proposes a definition of case notion in the context of OpenSLEX, and provides insights on how to obtain alternative perspectives on the event data. For the sake of simplicity, in this chapter we build one single event log per environment, which provides a specific perspective to be analyzed.

The following three sections consider real-life environments that can act as a source for event data. These environments are used to illustrate the scenarios in Figure 4.2

Table 4.1: Fragment of a redo-log: each row corresponds to the occurrence of an event

#	Time + Op + Table	Redo	Undo
1	2016-11-27 15:57:08.0 + INSERT + CUSTOMER	`insert into "SAMPLEDB". "CUSTOMER" ("ID", "NAME", "ADDRESS", "BIRTH_DATE") values ('17299', 'Name1', 'Address1', TO_DATE( '01-AUG-06', 'DD-MON-RR'));`	`delete from "SAMPLEDB". "CUSTOMER" where "ID" = '17299' and "NAME" = 'Name1' and "ADDRESS" = 'Address1' and "BIRTH_DATE" = TO_DATE( '01-AUG-06', 'DD-MON-RR') and ROWID = '1';`
2	2016-11-27 16:07:02.0 + UPDATE + CUSTOMER	`update "SAMPLEDB". "CUSTOMER" set "NAME" = 'Name2' where "NAME" = 'Name1' and ROWID = '1';`	`update "SAMPLEDB". "CUSTOMER" set "NAME" = 'Name1' where "NAME" = 'Name2' and ROWID = '1';`
3	2016-11-27 16:07:16.0 + INSERT + BOOKING	`insert into "SAMPLEDB". "BOOKING" ("ID", "CUSTOMER_ID") values ('36846', '17299');`	`delete from "SAMPLEDB". "BOOKING" where "ID" = '36846' and "CUSTOMER_ID" = '17299' and ROWID = '2';`
4	2016-11-27 16:07:16.0 + UPDATE + TICKET	`update "SAMPLEDB". "TICKET" set "BOOKING_ID" = '36846' where "BOOKING_ID" IS NULL and ROWID = '3';`	`update "SAMPLEDB". "TICKET" set "BOOKING_ID" = NULL where "BOOKING_ID" = '36846' and ROWID = '3';`
5	2016-11-27 16:07:17.0 + INSERT + BOOKING	`insert into "SAMPLEDB". "BOOKING" ("ID", "CUSTOMER_ID") values ('36876', '17299');`	`delete from "SAMPLEDB". "BOOKING" where "ID" = '36876' and "CUSTOMER_ID" = '17299' and ROWID = '4';`
6	2016-11-27 16:07:17.0 + TICKET + UPDATE	`update "SAMPLEDB". "TICKET" set "ID" = '36876' where "BOOKING_ID" IS NULL and ROWID = '5';`	`update "SAMPLEDB". "TICKET" set "ID" = NULL where "BOOKING_ID" = '36876' and ROWID = '5';`

and demonstrate that the complete meta-model structure can be derived for each of them. In each scenario, the goal is to create an integrated view of data and process, even when event logs are not directly available.

4.2.2 Database Redo-Logs

The first environment focuses on database *redo-logs*, a mechanism present in many DBMSs to guarantee consistency, as well as providing additional features such as rollback, point-in-time recovery, etc. Redo-logs have already been considered in our earlier work [44, 108] as a source of event data for process mining. Its feasibility has been validated in [38]. This environment corresponds to the scenario depicted in Figure 4.2.d, where data model, objects, and events are available, but object versions need to be inferred. Table 4.1 shows an example fragment of a redo-log obtained from an Oracle DBMS. After processing the redo-log records, explained in [44], they are transformed into events. Table 4.2 shows the derived values for attributes of these events according to the changes observed in the redo-log. For instance, we see that the first row of Table 4.2 corresponds to the processed redo-log record observed in the first row in Table 4.1. It corresponds to a *Customer* insertion. Therefore, none of the values for each attribute existed before the event was executed (fourth column). The second column holds the values for each attribute right after the event occurred.

Table 4.2: Fragment of a redo-log: each row corresponds to the occurrence of an event

#	**Attribute name**	**Value after event**	**Value before event**
1	Customer:id	17299	-
	Customer:name	Name1	-
	Customer:address	Address1	-
	Customer:birth_date	01-AUG-06	-
	RowID	=	1
2	Customer:id	=	{17299}
	Customer:name	Name2	Name1
	Customer:address	=	{Address1}
	Customer:birth_date	=	{01-AUG-06}
	RowID	=	1
3	Booking:id	36846	-
	Booking:customer_id	17299	-
	RowID	=	2
4	Ticket:booking_id	36846	NULL
	Ticket:id	=	(317132)
	Ticket:belongs_to	=	(172935)
	Ticket:for_concert	=	(1277)
	RowID	=	3
5	Booking:id	36876	-
	Booking:customer_id	17299	-
	RowID	=	4
6	Ticket:booking_id	36876	NULL
	Ticket:id	=	(317435)
	Ticket:belongs_to	=	(173238)
	Ticket:for_concert	=	(1277)
	RowID	=	5

The rest of the events are interpreted in the same way.

Figure 4.1 shows a general overview of how the meta-model sectors are completed according to the starting input data and the steps taken to derive the missing sectors. In this case, the analysis of database redo-logs allows one to obtain a set of events, together with the objects they belong to and the data model of the database. These elements alone are not sufficient to perform process mining without the existence of an event log (Figure 4.2.c). In addition, the versions of the objects of the database need to be inferred from the events as well (Figure 4.2.d). Therefore, the starting point is a meta-model in which the only populated sectors are *Events*, *Objects* and *Data Model*. We need to infer the remaining ones: *Versions*, *Cases*, and *Process Models*.

Fortunately, we developed a technique to build logs using different perspectives (Trace ID Patterns), as presented in [44]. The existence or definition of a data model is required for this technique to work. Figure 4.1 shows a diagram of the data transformation performed by the technique and how it fits in the proposed meta-model structure. A more formal description of the mapping between redo-logs and our meta-model can be consulted in Appendix A.2. The data model obtained from the source database is stored in an OpenSLEX file. This data model, together with the

Table 4.3: Redo-log dataset transformation to populate the OpenSLEX meta-model

Entity	Format	# Input Els.	# Output Els.	Aut/Manual	Derivation
Data Model	SQL	1	1	aut	explicit
Class	SQL	8	8	aut	explicit
Attribute	SQL	27	27	aut	explicit
Relationship	SQL	8	8	aut	explicit
Object	-	0	6740	aut	inferred
Version	-	0	8424	aut	inferred
Relation	-	0	13384	aut	inferred
Event	Redo-Log	8512	8512	aut	explicit
Activity Instance	-	0	8512	aut	inferred
Case	-	0	21	aut	inferred
Log	-	0	1	aut	inferred
Activity	-	0	14	aut	inferred
Process	-	0	1	aut	inferred

extracted events, allows us to generate both cases (Figure 4.2.c) and object versions (Figure 4.2.d). Next, process discovery completes the meta-model with a process (Figure 4.2.f). Once the meta-model structure is populated with data, we can make queries on it taking advantage of the established connections between all the entities and apply process mining to do the analysis.

Database Redo-Logs: Transformation

In the previous section, we have described the nature of redo-logs and how they can be extracted and transformed into event collections first, and used to populate the meta-model afterward. The goal of this section is to sketch the content of this resulting populated meta-model and, in further sections, the kind of analysis that can be done on it.

Table 4.3 represents quantitatively the input and output data for our meta-model population process for each of the entities introduced in Section 3.3. These entities match the ones depicted in Figure 3.4. Moreover, this table provides qualitative properties on the nature of the transformation for each entity. In this case, we observe that all the data are obtained from the original database through SQL (except for the redo-logs, which require some specific-to-the-job tools from Oracle). The input entities (*data model*, *class*, *attribute*, *relationship*, and *event*) are extracted automatically (fifth column), directly transformed given that they are explicitly defined in the source data (sixth column), and maintaining their quantity (third vs. fourth columns). However, the remaining entities of our meta-model need to be inferred, i.e., derived from the original entities. In this case, this derivation can be done automatically by our tool, PADAS[1], which implements the technique for redo-log extraction and event log building described in [44]. The data model that describes this dataset is depicted in Figure 4.3. It is clearly inspired by the running example provided in Section 3.2.

[1]`https://github.com/edugonza/padas`

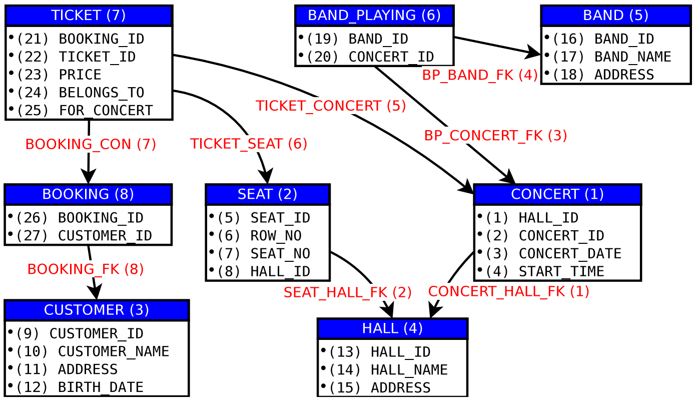


Figure 4.3: Data model of the redo-log dataset as obtained from the populated OpenSLEX file

Database Redo-Logs: Adapter Implementation

In order to assist in the task of extracting and processing redo-logs, the tool PADAS (Section 3.4) has been developed. This tool is able to:

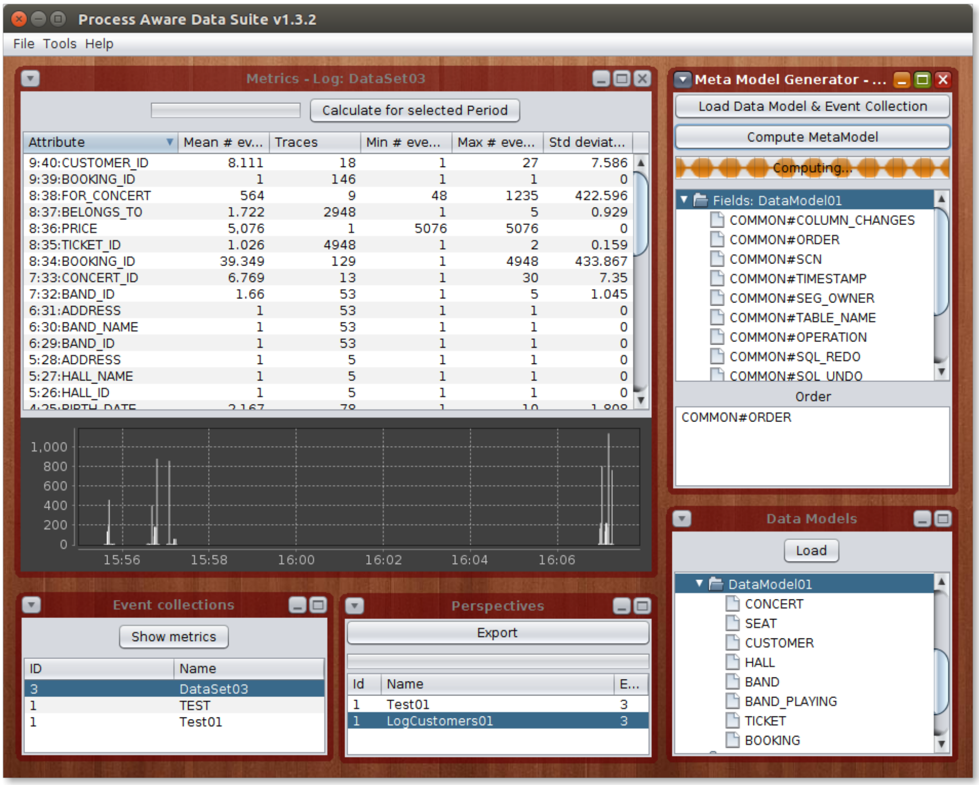
1. Connect to an existing Oracle RDBMS,
2. Load a collection of redo-logs,
3. Extract their content and fill the missing data from a database snapshot,
4. Obtain the data model automatically,
5. Export the resulting collection of events to an intermediate format.

The rest of the processing can be done offline from the Oracle database. The tool allows the user to select case notions from the data model and build logs for a specific view. As can be observed in Figure 4.4, once the three main sectors have been extracted, i.e., data model, events, and cases, the population of the meta-model can begin. This is an automatic process that does not require any further user interaction.

4.2.3 In-Table Versioning

It is not always possible to get redo-logs from databases. Sometimes, they are disabled or not supported by the DBMS. Also, we simply may not be able to obtain credentials to access them. Whatever the reason, it is common to face the situation in which events are not explicitly stored. This seriously limits the analysis that can be performed on the data. The challenge in such a setting is to obtain, somehow, an event log to complete our data.

It may be so that in a certain environment, despite lacking events, versioning of objects is kept in the database, i.e., it is possible to retrieve the old value for



Attribute	Mean # ev...	Traces	Min # eve...	Max # eve...	Std deviat...
9:40:CUSTOMER_ID	8.111	18	1	27	7.586
9:39:BOOKING_ID	1	146	1	1	0
8:38:FOR_CONCERT	564	9	48	1235	422.596
8:37:BELONGS_TO	1.722	2948	1	5	0.929
8:36:PRICE	5,076	1	5076	5076	0
8:35:TICKET_ID	1.026	4948	1	2	0.159
8:34:BOOKING_ID	39.349	129	1	4948	433.867
7:33:CONCERT_ID	6.769	13	1	30	7.35
7:32:BAND_ID	1.66	53	1	5	1.045
6:31:ADDRESS	1	53	1	1	0
6:30:BAND_NAME	1	53	1	1	0
6:29:BAND_ID	1	53	1	1	0
5:28:ADDRESS	1	5	1	1	0
5:27:HALL_NAME	1	5	1	1	0
5:26:HALL_ID	1	5	1	1	0
4:25:BIRTH_DATE	2.167	78	1	10	1.808

Fields: DataModel01
- COMMON#COLUMN_CHANGES
- COMMON#ORDER
- COMMON#SCN
- COMMON#TIMESTAMP
- COMMON#SEG_OWNER
- COMMON#TABLE_NAME
- COMMON#OPERATION
- COMMON#SQL_REDO
- COMMON#SQL_UNDO

Order: COMMON#ORDER

DataModel01
- CONCERT
- SEAT
- CUSTOMER
- HALL
- BAND
- BAND_PLAYING
- TICKET
- BOOKING

ID	Name
3	DataSet03
1	TEST
1	Test01

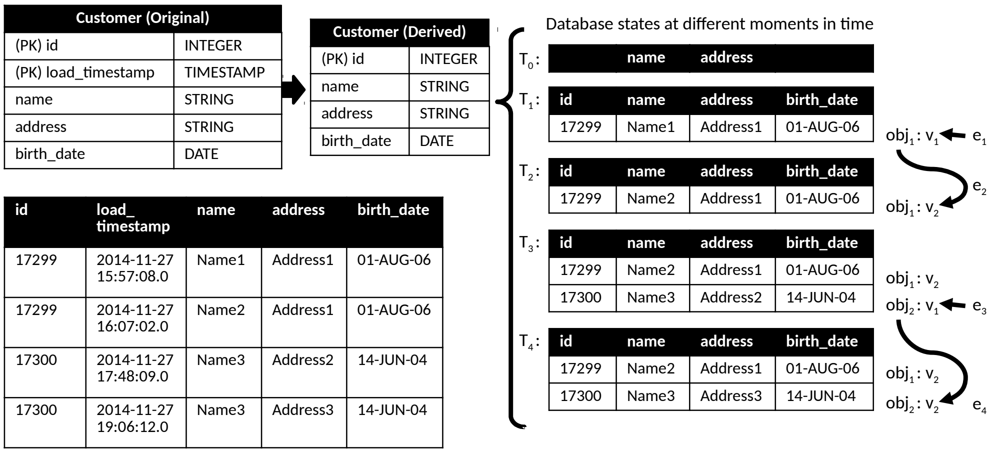
Id	Name	E...
1	Test01	3
1	LogCustomers01	3

Figure 4.4: PADAS tool settings to convert a redo-log dataset into a populated meta-model

any attribute of an object at a certain point in time. This is achieved by means of duplication of the modified versions of rows. This environment corresponds to the scenario depicted in Figure 4.2.e, where data model, objects, and object versions are available, but events need to be inferred. The table at the bottom left corner of Figure 4.5 shows an example of in-table versioning of objects. We see that the primary key of *Customer* is formed by the fields *id* and *load_timestamp*. Each row represents a version of an object and every new reference to the same *id* at a later *load_timestamp* represents an update. Therefore, if we order rows (ascending) by *id* and *load_timestamp*, we get sets of versions for each object. The first one (with older *load_timestamp*) represents an insertion, the rest are updates on the values. A more formal description of the mapping from In-table versioning environments to our meta-model can be found in Section A.3.

In-Table Versioning: Transformation

Looking at Figure 4.5 it is clear that, when ordering by timestamps, the versions in the original set (bottom left), we can reconstruct the different states of the database



Customer (Original)	
(PK) id	INTEGER
(PK) load_timestamp	TIMESTAMP
name	STRING
address	STRING
birth_date	DATE

Customer (Derived)	
(PK) id	INTEGER
name	STRING
address	STRING
birth_date	DATE

id	load_ timestamp	name	address	birth_date
17299	2014-11-27 15:57:08.0	Name1	Address1	01-AUG-06
17299	2014-11-27 16:07:02.0	Name2	Address1	01-AUG-06
17300	2014-11-27 17:48:09.0	Name3	Address2	14-JUN-04
17300	2014-11-27 19:06:12.0	Name3	Address3	14-JUN-04

Database states at different moments in time

T_0:

	name	address	

T_1:

id	name	address	birth_date
17299	Name1	Address1	01-AUG-06

T_2:

id	name	address	birth_date
17299	Name2	Address1	01-AUG-06

T_3:

id	name	address	birth_date
17299	Name2	Address1	01-AUG-06
17300	Name3	Address2	14-JUN-04

T_4:

id	name	address	birth_date
17299	Name2	Address1	01-AUG-06
17300	Name3	Address3	14-JUN-04

Figure 4.5: Example of in-table versioning and its transformation into objects and versions

(right). Each new row in the original table represents a change in the state of the database. Performing this process for all the tables allows for inferring the events in a setting where they were not explicitly stored. Figure 4.1 shows that, thanks to the meta-model proposed, it is possible to derive events starting from a data model, a set of objects, and their versions as input (Figure 4.2.e). The next step is to obtain cases from the events and data model applying the techniques from [39, 44] (described in Chapter 5) to split event collections into cases selecting an appropriate *case notion* (Figure 4.2.c). Finally, process discovery will allow us to obtain a process model to complete the meta-model structure (scenario *f*). A more formal description of the mapping between In-table versioning sources and our meta-model can be found in Appendix A.3.

As a result of the whole procedure, we have a meta-model completely filled with data (original and derived) that enables any kind of analysis available nowadays in the process mining field. Moreover, it allows for extended analysis by combining the data and process perspectives. Table 4.4 shows the input/output of the transformation process, together with details of its automation and derivation. We see that in this environment the input data corresponds to the following entities of the meta-model: *data model*, *class*, *attribute* and *version*. These elements are automatically transformed from the input data, where they are explicitly defined. However, the rest of the elements needs to be either manually obtained from domain knowledge (*dom. know.*), e.g. relationships between classes, or automatically inferred from the rest of the elements. This is so because, given the configuration of the database, there was no way to query the relationships between tables. After all, they were not explicitly defined in the DBMS but managed at the application level. Therefore, these relationships needed to be specified by hand after interviewing domain experts on the process at hand.

Table 4.4: In-table versioning dataset transformation to populate the OpenSLEX meta-model

Entity	Format	# Input Els.	# Output Els.	Aut/Manual	Derivation
Data Model	CSV	1	1	aut	explicit
Class	CSV	8	8	aut	explicit
Attribute	CSV	65	65	aut	explicit
Relationship	-	0	15	manual	dom. know.
Object	-	0	162287	aut	inferred
Version	CSV	277107	277094	aut	explicit
Relation	-	0	274359	aut	inferred
Event	-	0	277094	aut	inferred
Activity Instance	-	0	267236	aut	inferred
Case	-	0	9858	aut	inferred
Log	-	0	1	aut	inferred
Activity	-	0	62	aut	inferred
Process	-	0	1	aut	inferred

The resulting data model is depicted in Figure 4.6. It is based on the database of a financial organization, focusing on the claim and service processing for a specific financial product. As can be observed, this dataset links information corresponding to customers (*CUSTOMER (3)*), their products (*Product (8)* and *Service_Provition (6)*), incoming and outgoing phone calls (*Call_IN (2)* and *Call_OUT (7)*), letters being sent to customers (*Letter_OUT (4)*), monthly statements from customers (*MONTHLY_STATEMENT (1)*), and customer details change forms (*User_-Details_Change_Form_IN (5)*). In the coming sections, we will show how to obtain a view on this dataset by exploiting the structure of the meta-model.

In-Table Versioning: Adapter Implementation

The task of transforming the data from an environment that presents an In-table versioning structure into our meta-model can be automated by means of an adapter. In this case, we make use of the RapidMiner[2] platform together with the RapidProM[3] extension to implement a workflow that automates this transformation. Specifically, the operator needed for this task can be found in one of the development branches of the RapidProM repository[4]. This adapter, as depicted in Figure 4.7, takes as input the set of CSV files that contains the unprocessed dump of the tables to be analyzed directly from the source database. Also, if we get direct access to such a database, the data can be queried directly, skipping the transformation to CSV files.

4.2.4 Change Table

The last environment we will consider for our feasibility study is related to widespread ERP systems such as SAP. These systems provide a huge amount of functionalities

[2]`http://www.rapidminer.com`
[3]`http://www.rapidprom.org/`
[4]`https://github.com/rapidprom/rapidprom-source/tree/egonzalez`

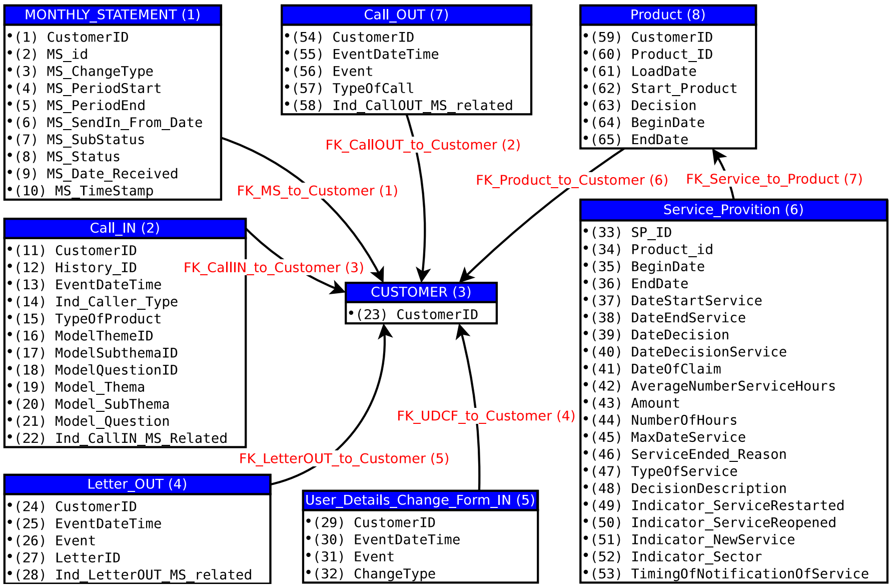


Figure 4.6: Data model of the in-table versioning dataset as obtained from the populated meta-model

to companies by means of configurable modules. They can run on various platforms and rely on databases to store all their information. However, in order to make them as flexible as possible, their implementation tries to be independent of the specific storage technology running underneath. We can observe SAP systems running on MS SQL, Oracle, and other technologies. However, they generally do not make intensive use of the features that the database vendor provides. Therefore, data relations are often not defined in the database schema, but managed at the application level. This makes the work of the analyst who would be interested in obtaining event logs rather complicated. Fortunately, SAP implements its own redo-log-like mechanism to store changes in data. This represents a valid source of data for our purposes. In this setting, we lack event logs, object versions, a complete data model, and processes. Without some of these elements, performing any kind of process mining analysis becomes very complicated. For instance, the lack of an event log does not allow for the discovery of a process and, without it, performance or conformance analyses are not possible. To overcome this problem, we need to infer the lacking elements from the available information in the SAP database.

First, it must be noted that, despite the absence of an explicitly defined data model, SAP uses a consistent naming system for their tables and columns. Also, there is lots of documentation available that describes the data model of the whole SAP table landscape. Therefore, this environment corresponds to the scenario depicted in Figure 4.2.d, where data model, objects, and events are available, but object versions need to be inferred. To extract the events we need to process the change log. This

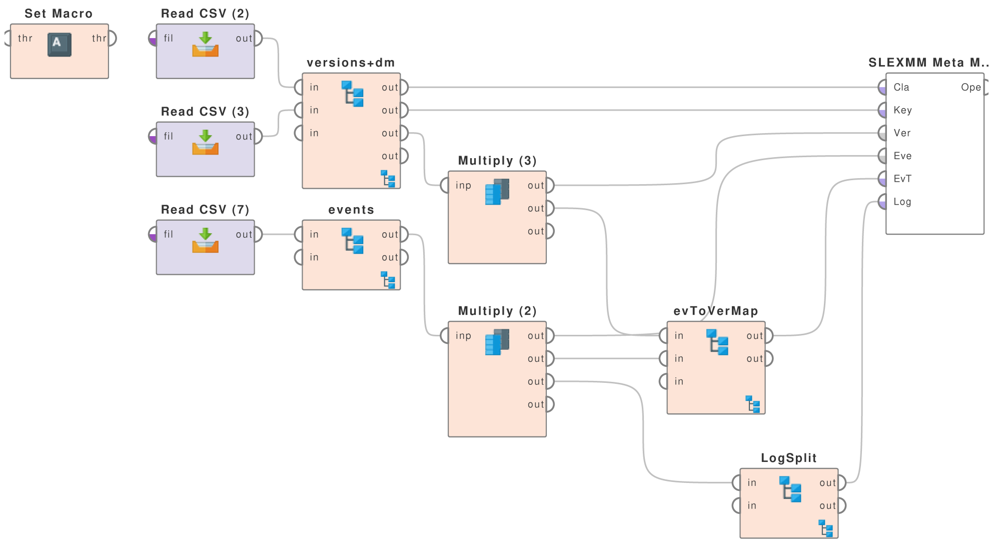
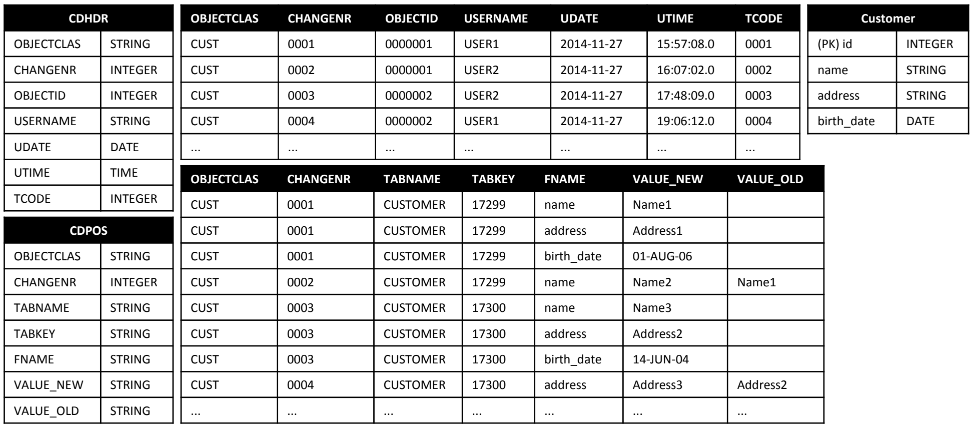


Figure 4.7: RapidMiner workflow to convert an in-table versioning dataset in order to populate the OpenSLEX meta-model

SAP-style change log, as can be observed in Figure 4.8, is based on two change tables: *CDHDR* and *CDPOS*. The first table, *CDHDR*, stores one entry per change performed on the data with a unique change id (*CHANGENR*,*OBJECTCLAS*,*OBJECTID*) and other additional details. The second table, *CDPOS*, stores one entry per field changed. Several fields in a data object can be changed at the same time and will share the same change id. For each field changed, the table name is recorded (*TABNAME*) together with the field name (*FNAME*), the key of the row affected by the change (*TABKEY*), and the old and new values of the field (*VALUE_OLD*, *VALUE_NEW*). A more formal description of the mapping between SAP change tables and our meta-model can be found in the Appendix A.4. This structure is very similar to the one used for redo-logs. However, one of the differences is that changes on different fields of the same record are stored in different rows of the *CDPOS* table, while in the redo logs they are grouped in a single operation.

Change Table: Transformation

As can be seen in Figure 4.1, after processing the change-log and providing a SAP data model, we are in a situation in which the events, objects, and data model are known. Therefore, we can infer the versions of each object (d), split the events over cases (c), and finally, discover a process model (f). With all these ingredients it becomes possible to perform any process mining analysis and answer complex questions combining process and data perspectives.



CDHDR	
OBJECTCLAS	STRING
CHANGENR	INTEGER
OBJECTID	INTEGER
USERNAME	STRING
UDATE	DATE
UTIME	TIME
TCODE	INTEGER

OBJECTCLAS	CHANGENR	OBJECTID	USERNAME	UDATE	UTIME	TCODE
CUST	0001	0000001	USER1	2014-11-27	15:57:08.0	0001
CUST	0002	0000001	USER2	2014-11-27	16:07:02.0	0002
CUST	0003	0000002	USER2	2014-11-27	17:48:09.0	0003
CUST	0004	0000002	USER1	2014-11-27	19:06:12.0	0004
...	...	...	...	...	...	...

Customer	
(PK) id	INTEGER
name	STRING
address	STRING
birth_date	DATE

CDPOS	
OBJECTCLAS	STRING
CHANGENR	INTEGER
TABNAME	STRING
TABKEY	STRING
FNAME	STRING
VALUE_NEW	STRING
VALUE_OLD	STRING

OBJECTCLAS	CHANGENR	TABNAME	TABKEY	FNAME	VALUE_NEW	VALUE_OLD
CUST	0001	CUSTOMER	17299	name	Name1	
CUST	0001	CUSTOMER	17299	address	Address1	
CUST	0001	CUSTOMER	17299	birth_date	01-AUG-06	
CUST	0002	CUSTOMER	17299	name	Name2	Name1
CUST	0003	CUSTOMER	17300	name	Name3	
CUST	0003	CUSTOMER	17300	address	Address2	
CUST	0003	CUSTOMER	17300	birth_date	14-JUN-04	
CUST	0004	CUSTOMER	17300	address	Address3	Address2
...	...	...	...	...	...	...

Figure 4.8: Example of SAP change tables CDHDR and CDPOS

Table 4.5: SAP dataset transformation to populate the OpenSLEX meta-model

Entity	Format	# Input Els.	# Output Els.	Aut/Manual	Derivation
Data Model	SQL	1	1	aut	explicit
Class	SQL	87	87	aut	explicit
Attribute	SQL	4305	4305	aut	explicit
Relationship	-	0	296	aut	dom. know.
Object	SQL	7340011	7339985	aut	explicit
Version	-	0	7340650	aut	inferred
Relation	-	0	7086	aut	inferred
Event	SQL	26106	26106	aut	explicit
Activity Instance	-	0	5577	aut	inferred
Case	-	0	22	aut	inferred
Log	-	0	1	aut	inferred
Activity	-	0	172	aut	inferred
Process	-	0	1	aut	inferred

Table 4.5 shows that, in order to populate our meta-model, what we obtain from SAP are the following entities: *data model*, *class*, *attribute*, *object*, and *event*. All these elements are explicitly defined and can be automatically transformed. However, the rest needs further processing to be inferred from the input data. Only the relationships cannot be inferred, but can be obtained automatically from domain knowledge. The difference between in-table versioning and SAP is that, in the case of SAP, we can query the online documentation that specifies the connections between different tables of their data model. A script[5] has been developed which, from a set of table names, finds and obtains the relationships and stores them in a CSV file.

To get an idea of the complexity of the data model of this dataset, Figure 4.9 shows a full picture of it. As can be noticed, it is a very complex structure. However,

[5]https://www.win.tue.nl/~egonzale/createkeysfile-sh/

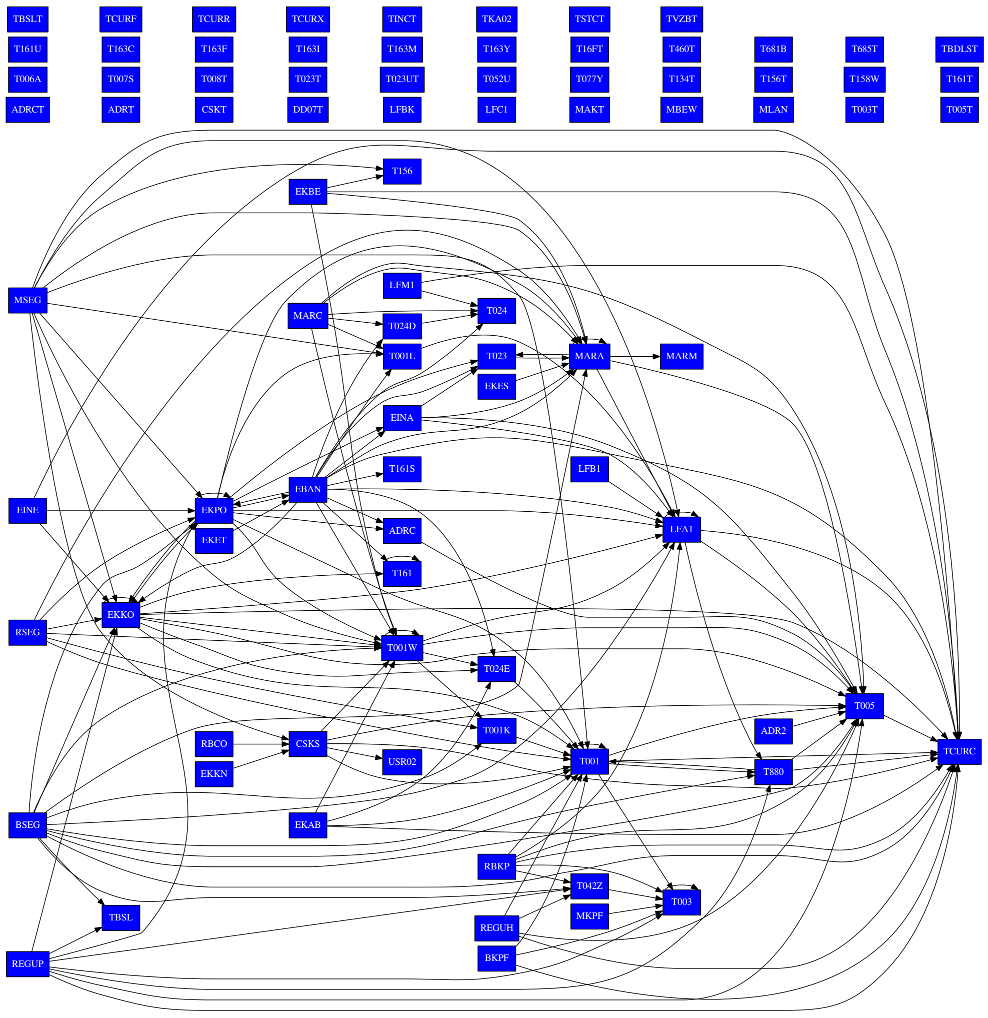


Figure 4.9: General view of the data model of the SAP dataset as obtained from the populated meta-model. Due to the size of the data model, the attributes or the tables have been omitted from this graph.

the tool allows one to zoom in and explore areas of interest. Figure 4.10 shows a zoomed-in area of the data model, where the *EKPO* table points to the *EKKO* table.

Change Table: Adapter Implementation

As has been mentioned before, SAP systems often use relational databases to store the documents and information they manage. Despite the wide variety of database systems used, the extraction of the relevant information is always possible, as long

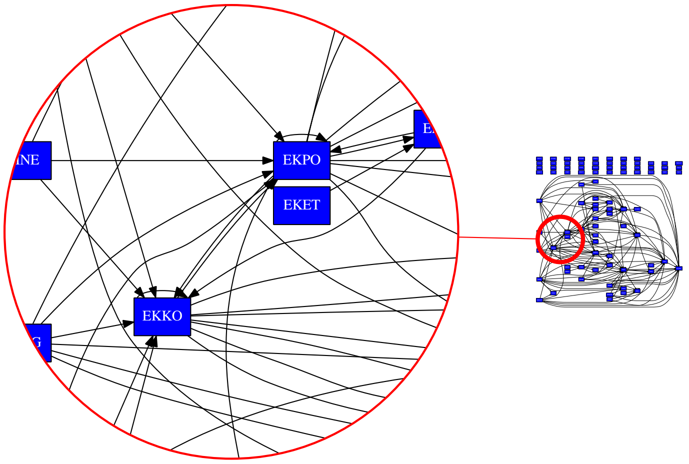


Figure 4.10: Detail of the data model of the SAP dataset as obtained from the populated meta-model

as a JDBC Driver exists. This allows one to connect to the source database directly from RapidMiner, as shown in the workflow in Figure 4.11. In this specific case, the database was SAP Adaptive Server Enterprise (ASE), originally known as Sybase SQL Server. The process is as simple as providing the list of table names we are interested in. The workflow will loop through them (Figure 4.12) and extract their content into CSV files that will be processed in a further step by a different workflow.

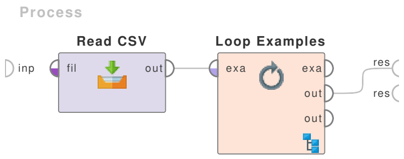


Figure 4.11: RapidMiner workflow to connect to an SAP database and extract the content of all the tables

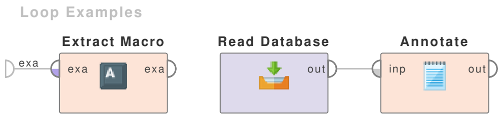


Figure 4.12: RapidMiner subprocess to extract each table when connecting to a SAP database

Once we have the CSV files of the SAP tables, we need to process them. To do so, the RapidProM workflow in Figure 4.13 has been developed, which loops through these files (including the SAP change tables) to build our meta-model. The top branch corresponds to the processing of the data schema. The branch below incorporates the information about the foreign keys, which are used to define the relationships at the leve of the data schema, and to establish relations between object versions. The third branch loads information about the objects, i.e., unique identifiers of each data object from the source database. The two branches at the bottom process the change log tables from SAP in order to derive object versions. The rightmost block takes as an input the source data in a standardized format, and outputs an OpenSLEX file corresponding to a valid meta-model instance.

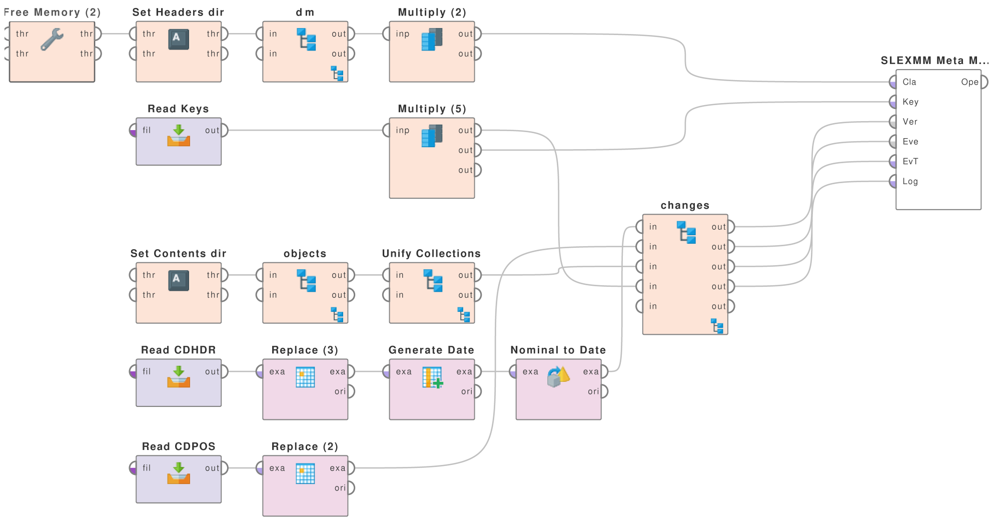


Figure 4.13: RapidMiner workflow to populate the meta-model based on SAP dataset.

4.2.5 Merging Data Sources

The three previous environments have been considered in isolation, showing the ETL process on individual systems. On the other hand, as depicted in Figure 3.1, it often happens that the landscape of a company is constructed by several systems such as ERP systems, CRM systems, BPM systems, and so on, which operate separately. These systems store data related to different aspects of the operations within a company. It is possible that, in order to perform our analysis, we need to make use of data from several of these sources combined, e.g., to link user requests made through a website on the one hand with internal request handling managed by an internal SAP system on the other. In such a case, we need to merge the data coming from these independent systems into a single structure. In order to achieve this goal, we require, at the very least, one common connecting concept to be shared by these systems.

Figure 4.14 shows an example of two systems being merged in a single data model. On the left-hand side of the figure, the data model described in Section 4.2.3 is shown as an abstract group of tables from which one of them, the customer table, has been

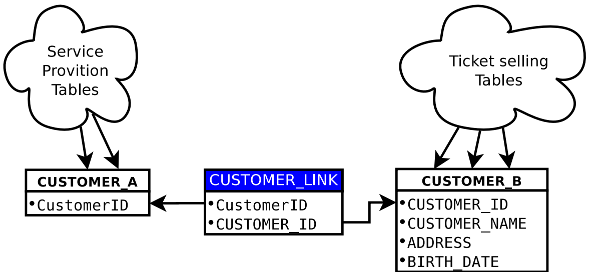


Figure 4.14: The link between different data models through tables representing a common concept (*CUSTOMER_A* and *CUSTOMER_B*) by means of a link table (*CUSTOMER_LINK*)

selected. On the right-hand side, the ticket selling platform described in Section 4.2.2 is represented, from which the customer table has been selected as well. Both customer tables (*CUSTOMER_A* and *CUSTOMER_B*) have a customer identification field. Both could share the same ids to represent the same concept. However, that is not needed to be correlated. The merging process requires the creation of an additional linking table (*CUSTOMER_LINK*) which holds the relation between customer ids for both separate data models.

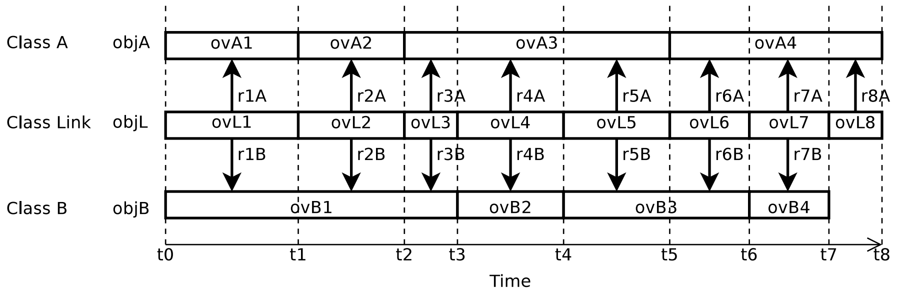


Figure 4.15: Merging method to map versions of objects belonging to classes of different data models (class A and class B) through a linking class (class Link)

To make the combination of these data models fit the structure of our meta-model, it is necessary to make the connection between objects at the object version level. This can be done automatically as long as we know (a) which customer id from table A is related to which customer id from table B, and (b) during which period of time this relation existed. It may be that the correlation is permanent, for example when we map customer ids of two systems which always represent the same entity (e.g. social security numbers on one system with national identification numbers on the other). In such a case, the connecting object will always relate to versions of the same two connected objects. This situation is represented in Figure 4.15, where an object of class A (top line) is connected to an object of class B (bottom line) by means of a linking object (middle line). For each pair of coexisting object versions of *objA* and *objB*, a new version of *objL* must exist to relate them. This method has the benefit of providing great flexibility. It allows us to establish relations between objects that change through time. An example of this is the case in which the connection between two tables represents employer-to-employee relations. An employer can be related to many employees and employees can change employers at any time. Therefore, the mapping between employer and employee objects will not be permanent over time, but rather evolve and change. Such a case is supported by the proposed mapping method, as presented in Figure 4.16, by means of new object versions for the linking object.

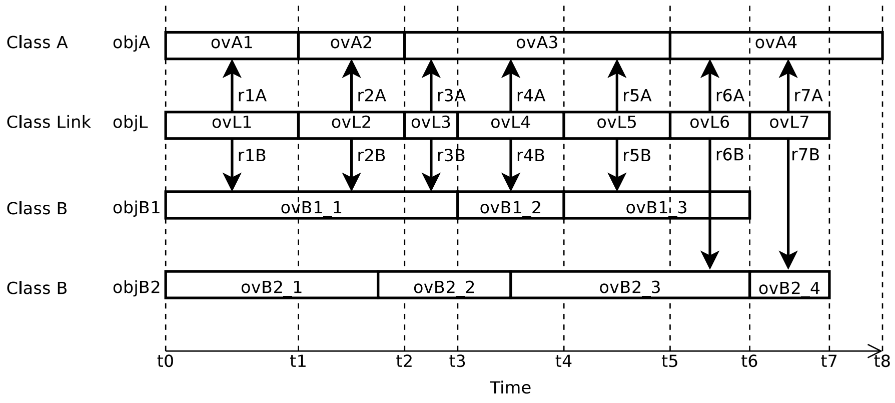
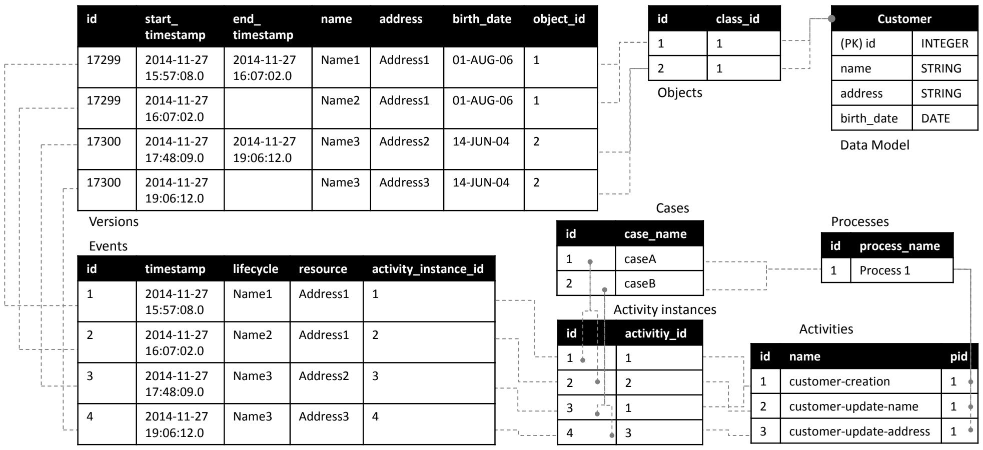


Figure 4.16: Merging method to map versions of multiple objects belonging to classes of different data models (class A and class B) through a linking class (class Link). In this case, the related objects change through time

As demonstrated with this example, the proposed meta-model is able to merge information from different systems into a single structure, enabling the analysis of process and data in a holistic way, beyond the boundaries of IT systems infrastructure. Throughout this section, we have seen how data extraction and transformation into the proposed meta-model can be performed when dealing with environments of very different nature. It is important to note that, despite its apparent complexity, all the steps previously mentioned are carried out automatically by the provided implementations of the adapters. These adapters can be modified and extended to add support for new environments.

4.3 Analysis of the Resulting Populated Meta-Model

The main advantage of transforming all our source information into the proposed meta-model structure is that, regardless of the origin of data, we can pose questions in a standard way. Let us consider the three environments described in previous sections: redo-logs, in-table versions, and change tables. In the examples, we chose to illustrate the transformation it is evident that they belong to very different processes and businesses. The redo-log example corresponds to a concert ticket selling portal.



Versions

id	start_ timestamp	end_ timestamp	name	address	birth_date	object_id
17299	2014-11-27 15:57:08.0	2014-11-27 16:07:02.0	Name1	Address1	01-AUG-06	1
17299	2014-11-27 16:07:02.0		Name2	Address1	01-AUG-06	1
17300	2014-11-27 17:48:09.0	2014-11-27 19:06:12.0	Name3	Address2	14-JUN-04	2
17300	2014-11-27 19:06:12.0		Name3	Address3	14-JUN-04	2

Objects

id	class_id
1	1
2	1

Data Model

Customer	
(PK) id	INTEGER
name	STRING
address	STRING
birth_date	DATE

Events

id	timestamp	lifecycle	resource	activity_instance_id
1	2014-11-27 15:57:08.0	Name1	Address1	1
2	2014-11-27 16:07:02.0	Name2	Address1	2
3	2014-11-27 17:48:09.0	Name3	Address2	3
4	2014-11-27 19:06:12.0	Name3	Address3	4

Cases

id	case_name
1	caseA
2	caseB

Processes

id	process_name
1	Process 1

Activity instances

id	activitiy_id
1	1
2	2
3	1
4	3

Activities

id	name	pid
1	customer-creation	1
2	customer-update-name	1
3	customer-update-address	1

Figure 4.17: Fragment of resulting populated meta-model

The second example, based on in-table versioning, corresponds to the claim and service processing for a financial product within an organization. The third example, based on a SAP system, represents the procurement process of a fictitious company. Due to the diversity of data formats and designs of the systems, in a normal situation these three environments would require very different approaches in order to be queried and analyzed. However, we claim that our meta-model provides the standardization layer required to tackle these systems in a similar manner. In this section we propose a way to standardize the querying process, using a similar template to answer different questions on each dataset.

4.3.1 Standardized Querying

Our goal is to query the data from different sources in a standardized fashion. To do so, as demonstrated in Section 5.6, we extracted the raw input data from the databases of each system. Next, our automated workflows were used to transform these data and obtaining, as a result, a structure compliant with the proposed meta-model. In this section, we assume that a fully populated meta-model (result on the right in Figure 4.1) is available for each of the three environments under study. Also, we assume that the case derivation step (Figure 4.2.c) has been applied in order to generate, at least, one set of cases that capture the perspective needed to answer to business question of interest. As has been stated in Section 4.2.1, many different case notions could be considered in order to generate alternative perspectives based on the same event data. Chapter 5 will elaborate on this by proposing a definition of case notion based on our meta-model in order to support multi-perspective analysis.

Given the populated meta-model instances obtained from the previously described environments, we hope to make plausible that, given a specific business question, it is possible to make a SQL query that answers this question, regardless of the original format of the data. We demonstrate our claim with the following example. Let us assume we want to find the group of cases in our logs that comply with a specific rule or represent a part of the behavior we want to analyze. As an example, for the redo log dataset we may want to define the following Business Question (BQ1):

BQ1: In which cases was the address of a customer updated?

Doing this on the original data would require to go through the redo-log, looking for events on the table *CUSTOMER*, linking them to the specific *customer_id*, correlating the other events of the same customer, grouping them in traces and building the log. This is not an easy task, especially if such a complicated query needs to be built specifically for each new business question that comes to mind. Let us consider another example, now for the in-table versioning dataset (BQ2):

BQ2: In which cases was a service request resolved?

To answer this, we need to go through the table *product* and check in which row the value of the field *decision* changed respect to the previous one, that has the same *customer_id*. The next step would be to correlate events from any other table through the same *customer_id* and build the log again. This requires a sequence of non-trivial and time-consuming steps. As a final example, we could think of the following question to inspect on the basis of the SAP dataset (BQ3):

BQ3: In which cases was the quantity of a purchase requisition order modified?

This time we would need to go to the *EKPO*[6] table, which contains information about purchasing document items. However, unlike the in-table versioning case, the previous values of the items are not in the original table anymore. To find historical values for each field, we need to go through the CDHDR and CDPOS tables to find a row that corresponds to the modification of the field *MENGE* (purchase order quantity). We must look at the value of the field *FNAME*, which should be equal to the string *"MENGE"*, as well as the value of the field *TABNAME*, which should be equal to the string *"EKPO"*. Next, that needs to be matched with the correct Purchase Order (sharing the same *TABKEY*). Finally, we should correlate event (from the *CDHDR* and *CDPOS* tables) that affected the same purchase order and group them in cases to form a log. By doing so, we can see the changes in context. As the complexity of data increases, the process to query it becomes more complex as well.

To summarize, if we want to ask three different questions in three different environments, then each question needs to be answered in a specific way. However, we claim that we can derive a standard way to answer questions of the same nature on different datasets, as long as the data have been previously normalized into our meta-model. If we look at the three proposed business questions (BQ1, BQ2 and BQ3), all of them have something in common: they rely on identifying cases in which

[6] `http://www.sapdatasheet.org/abap/tabl/EKPO.html`

a value was modified for a specific field. We can even filter the cases to obtain only the ones that cover a specific period of time. This means that we can generalize these questions into one common general question (GQ):

> **GQ:** In which cases (a) there was an event that happened between time T1 and T2 (b) that performed a modification in a version of class C (c) in which the value of field F changed from X to Y?

This is a question we can answer on the basis of our meta-model, regardless of the dataset it represents. All we need to do is to specify the values of each parameter (T1, T2, C, X, and Y) according to the data model at hand. We translated the general question GQ into the SQL query in Listing 7.20.

Query 4.1: Standard query executed on the three populated meta-models

```sql
SELECT C.id as "T:concept:name",
  E.timestamp as "E:time:timestamp",
  AC.name as "E:concept:name",
  E.resource as "E:org:resource",
  E.lifecycle as "E:lifecycle:transition",
  E.ordering
FROM
  event as E,
  "case" as C,
  activity_instance as AI,
  activity_instance_to_case as AITC,
  activity as AC
WHERE
  C.id = AITC.case_id AND
  AITC.activity_instance_id = AI.id AND
  E.activity_instance_id = AI.id AND
  AI.activity_id = AC.id AND
  C.id IN
  (
    SELECT C.id
    FROM
      "case" as C,
      class as CL,
      object as O,
      object_version as OV,
      object_version as OVP,
      event as E,
      activity_instance as AI,
      activity_instance_to_case as AITC,
      event_to_object_version as ETOV,
      attribute_name as AT,
      attribute_value as AV,
      attribute_value as AVP
    WHERE
      E.activity_instance_id = AI.id AND
      AITC.activity_instance_id = AI.id AND
      AITC.case_id = C.id AND
      ETOV.event_id = E.id AND
      ETOV.object_version_id = OV.id AND
      OV.object_id = O.id AND
      O.class_id = CL.id AND
      CL.name = "%{CLASS_NAME}" AND
      E.timestamp > "%{TS_LOWER_BOUND}" AND
      E.timestamp < "%{TS_UPPER_BOUND}" AND
      AT.name = "%{ATTRIBUTE}" AND
      AV.attribute_name_id = AT.id AND
      AV.object_version_id = OV.id AND
      AV.value LIKE "%{NEW_VALUE}" AND
      AVP.attribute_name_id = AT.id AND
      AVP.object_version_id = OVP.id AND
      AVP.value LIKE "%{OLD_VALUE}" AND
      OVP.id IN
      (
        SELECT OVP.id
        FROM object_version as OVP
        WHERE
          OVP.start_timestamp < OV.start_timestamp AND
          OVP.object_id = OV.object_id
        ORDER BY OVP.start_timestamp DESC LIMIT 1
      )
  )
ORDER BY C.id, E.ordering;
```

This query is standard and independent of the dataset thanks to the use of macros (RapidMiner macros) denoted by %{*MACRO_NAME*}. We just need to instantiate their values according to the dataset to process. Table 4.6 shows the values for each macro to be replaced in each case. Notice that the timestamps (*TS_LOWER_-BOUND* and *TS_UPPER_BOUND*) are expressed in milliseconds and the string % in *NEW_VALUE* and *OLD_VALUE* will match any value.

Table 4.6: Parameters to query the three different populated meta-models with the same query

Variable	Value-RL	Value-ITV	Value-SAP
CLASS_NAME	CUSTOMER	Product	EKPO
TS_LOWER_BOUND	527292000000	527292000000	527292000000
TS_UPPER_BOUND	1480531444303	1480531444303	1480531444303
ATTRIBUTE	ADDRESS	Decision	MENGE
NEW_VALUE	%	Toekenning	%
OLD_VALUE	%	Nog geen beslissing	%

This query can be easily executed in the RapidMiner environment by using the *Query Database* operator. Figure 4.18 shows the workflow executed to set the macro values and make the query for each dataset. Some details of the logs obtained for each of the three datasets can be observed in Table 4.7.

Table 4.7: Query results for the three different populated meta-models

Properties	Redo-Log	In-table Ver.	SAP
# Cases	17	3177	1
# Events	301	69410	12
# Event classes	5	50	11

It is important to notice that this technique does not exclude the classical way to analyze logs. On the contrary, it enables the use of all the existing process mining techniques. A proof of it is the workflow in Figure 4.18, which executes the subprocess in Figure 4.19 for each of the resulting logs. As can be observed, this process mining task transforms the log table into an XES log. Next, a process tree is discovered using

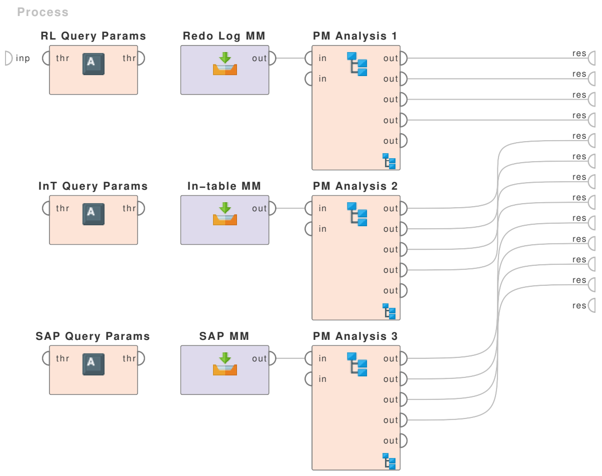


Figure 4.18: Analysis workflow to process the three resulting populated meta-models

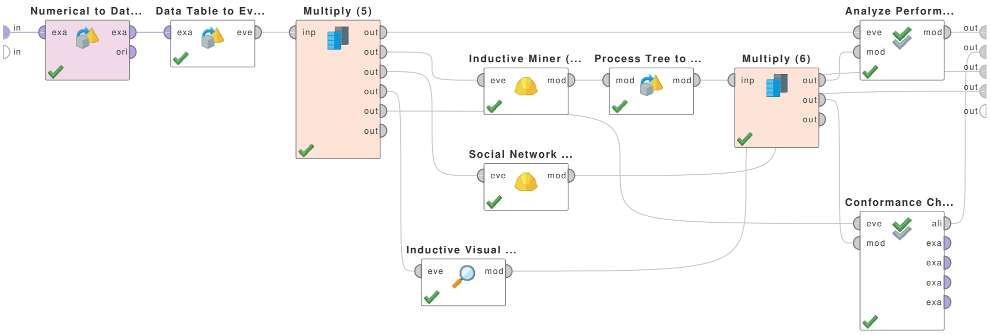


Figure 4.19: Process mining analysis subprocess in the analysis workflow of the three populated meta-models. It shows that queries on the populated meta-models can be converted into event logs to be analyzed by a range of process mining algorithms.

the Inductive Miner, which transforms it into a Petri Net. After that, a performance analysis is done on the discovered model and the log, together with a conformance check. In addition to that, a more interactive view of the process is obtained with the Inductive Visual Miner. A social network (assuming the resource field is available) can be discovered using the Social Network Miner. All these analyses are performed automatically and with exactly the same parameters for each resulting log. It is completely independent of the source. Just by modifying the values of the macros of our query, we can standardize the process of analyzing sub-logs for value modification cases.

4.3.2 Process Mining Results

In this section, we continue with the study in order to determine the viability of this technique. We show the results of the automated analysis performed on the three different datasets by means of the same query and same workflow.

The Redo-Log Environment: Ticket Selling Process

In the case of the ticket selling process, we are interested in the traces that modified the address of a customer. First, the discovered model is shown in Figure 4.20. We see five main activities in this model: customer insertions, booking insertions, and three different kinds of customer updates. One can notice that the activity *CUSTOMER+UPDATE+1411*, which corresponds to updates in the address field, is the most frequent of the three modifications. The activity *BOOKING+INSERT+44* is the most frequent with a total of 141 executions. One can also see that, as expected, a customer record can only be modified when it has been inserted before. The dashed lines in the model represent deviations from the discovered model, which usually represent infrequent cases that were filtered out during the mining phase.

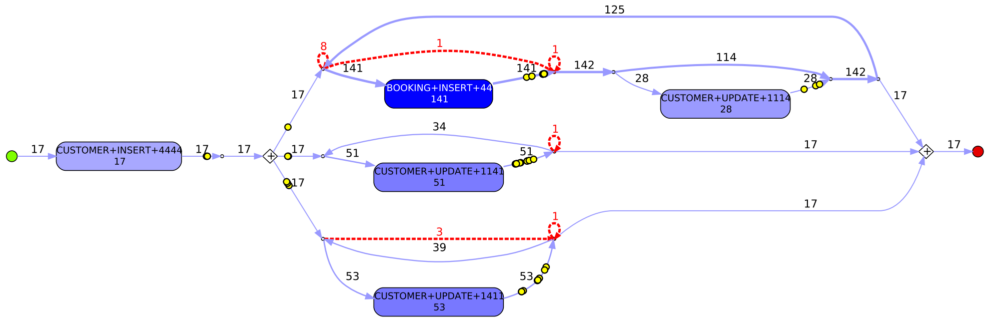


Figure 4.20: Discovered model and deviations for the redo-log dataset

Another technique that can be used during the execution of the automated workflow is performance analysis. The log is replayed on the discovered model such that time differences can be computed between consecutive events. Next, this information is displayed on top of the original model to visualize bottlenecks and performance issues. In this case, Figure 4.21 shows that the most time-consuming activity is one of the customer updates. However, this information can be misleading since these modifications were atomic, while data about start and complete events is missing. To have a better view on performance metrics and obtain accurate task duration, the life-cycle attribute should be properly assigned to pair events in activity instances. This is done automatically by the transformation adapter. However, in this case, there are no activities to be paired.

To finalize the analysis of this process, Figure 4.22 shows the result of checking the conformance of the log respect to the process. Here we see that, framed in red, activities *CUSTOMER+UPDATE+1411* and *BOOKING+INSERT+44* show move

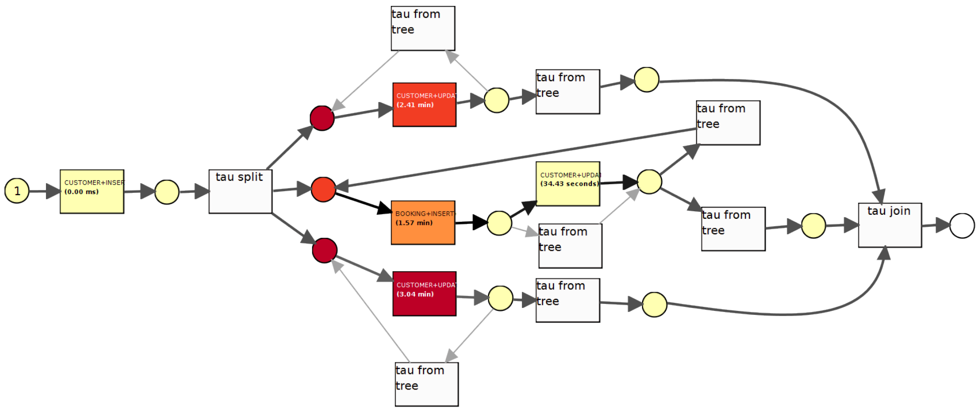


Figure 4.21: Performance analysis of the model for the redo-log dataset

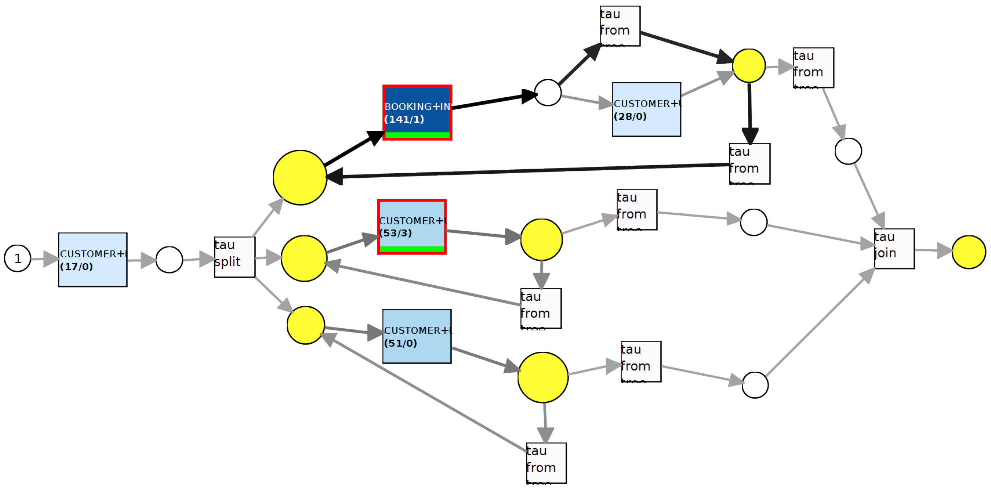


Figure 4.22: Conformance analysis of the model for the redo-log dataset

on log deviations. However, as the green bar at the bottom and the number in parentheses show, these deviations occur in a very small share of the total cases. The rest of the activities are always executed synchronously in accordance with the model.

The In-Table Versioning Environment: Claim Management Process

The second environment corresponds to the claim management process of a financial organization. The result of our query represents the cases in which the value of the *Decision* field of product service claim changed from *Undecided* (*Nog geen beslissing*) to *Granted* (*Toekenning*). Figure 4.23 shows the model as obtained from the Inductive Visual Miner. It can be observed that the process starts with a *Monthly Statement*

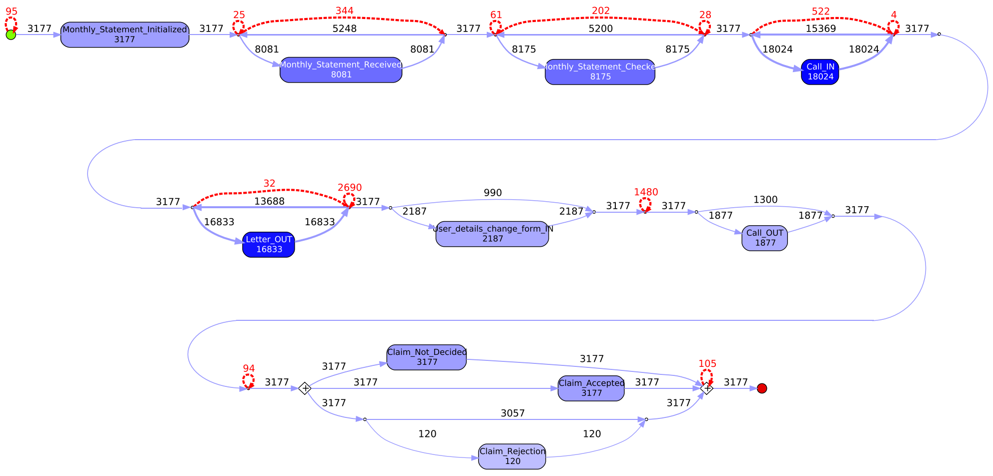


Figure 4.23: Discovered model & deviations for the in-table versioning dataset

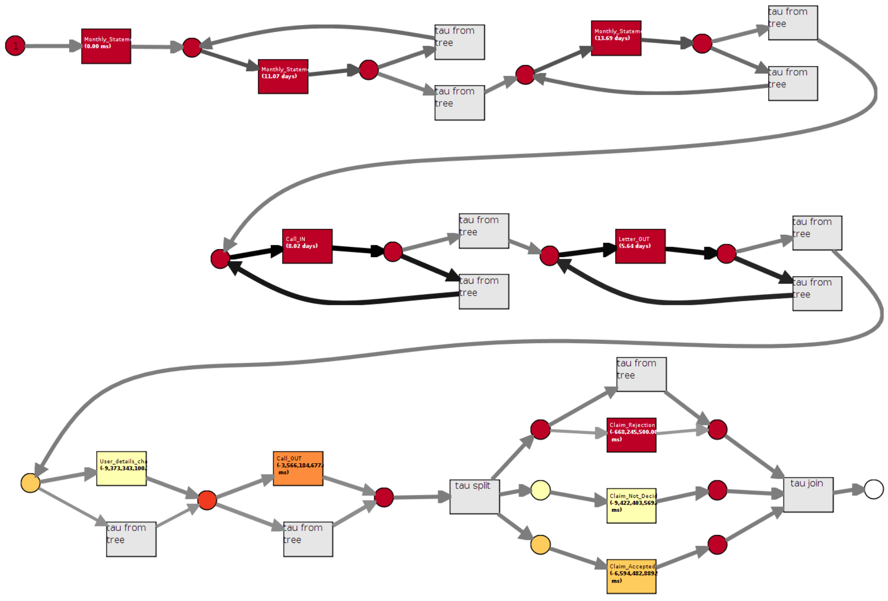


Figure 4.24: Performance analysis of the model for the in-table versioning dataset

being initialized. Then, for most of the following steps three possibilities exist: the activity is executed, the activity is re-executed (in a loop), or the activity is skipped. The frequencies in the arcs demonstrate that, in most of the cases, each subsequent activity happens at least once. These activities consist of the *Reception* and *Checking* of *Monthly Statements*, *Incoming calls*, *Outgoing letters*, *Changes in user details*, and *Outgoing calls*. Finally, the claim is resolved and three outcomes are possible: *Not decided*, *Accepted*, or *Rejected*.

When looking at the performance view in Figure 4.24, we notice that most of the places of the net are colored in red. This means that the waiting time in these places is higher than the rest. This makes sense since for some automatic activities the waiting time between tasks will be very low, i.e. in the order of milliseconds or seconds, and those places will set the lower bound for the performance scale. On the other hand, for most of the human-driven activities, the times will be much longer, i.e. in the order of days. With respect to activities, we see some variability, observing some very costly activities especially at the beginning. These activities are mainly the ones related to processing of *Monthly Statements*, *Incoming calls*, and *Outgoing letters*.

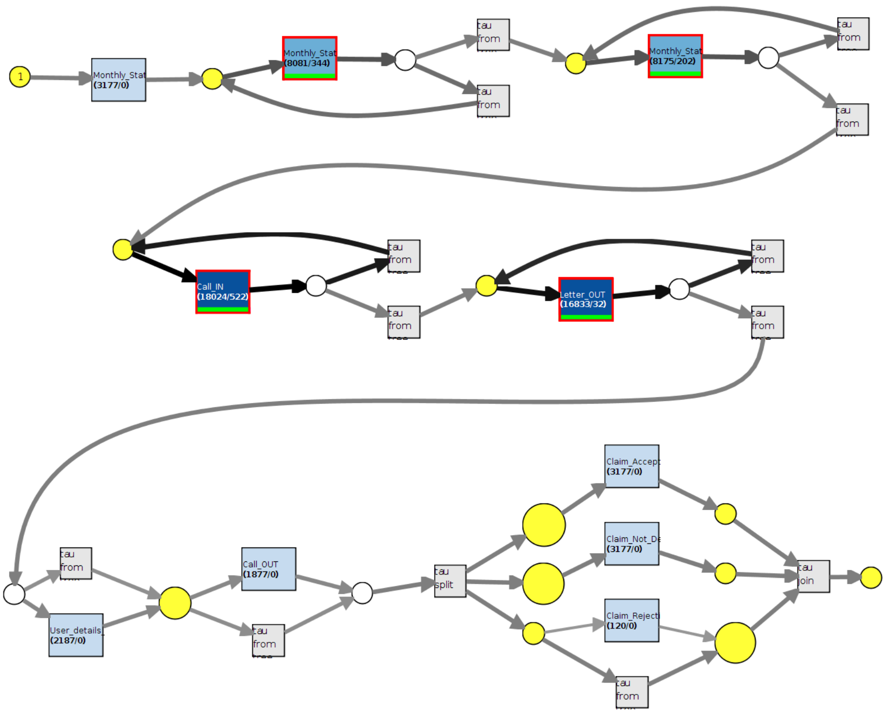


Figure 4.25: Conformance analysis of the model for the In-table Versioning dataset

Figure 4.25 shows some conformance results. Here we can observe that most of the activities happen without significant deviations. However, some model moves are observed. This happens at a very low rate, mainly because of the infrequent skips explained previously on Figure 4.23.

The SAP Environment: Procurement Process

Finally, we focus on the Procurement process within an SAP environment. We are particularly interested in the cases in which the quantity of a purchase order was modified. After querying the populated meta-model and obtaining the relevant cases, we used the Inductive Visual Miner to get a model of the process. Figure 4.26 shows a very structured process, which is not strange given the few cases that fulfilled our criteria.

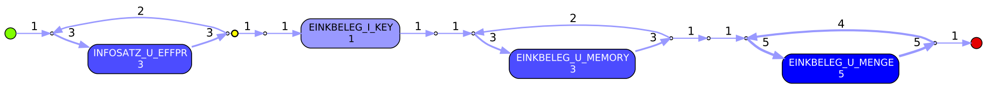


Figure 4.26: Discovered model & deviations for the SAP dataset

It seems clear that some activities are executed repeatedly because of the loops. The process always starts with an update in the value of the *Effective Price in Purchasing Info Record* (*INFOSATZ_U_EFFPR*). This is followed by updates in the *Purchasing Document* (*EINKBELEG* object class), specifically in the *Purchase order not yet complete* field (*MEMORY*). Only then, the *Purchase Order Quantity* (*MENGE*) field is updated.

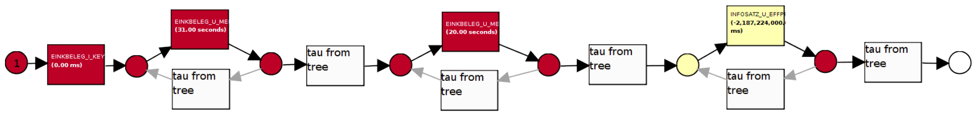

EINKBELEG_I_KEY (0.00 ms)
EINKBELEG_U_ME (31.00 seconds)
EINKBELEG_U_ME (20.00 seconds)
INFOSATZ_U_EFFP (-2,187,224,000 ms)
tau from tree

Figure 4.27: Performance analysis of the model for the SAP dataset

According to Figure 4.27, the performance seems evenly distributed, except for the update in *Purchase Order Quantities*, which seems to take a shorter time than the rest. From a conformance point of view (Figure 4.28), we see that the process does not show deviations. This was to be expected, given the size of the log and the existence of only one case with this particular behavior.

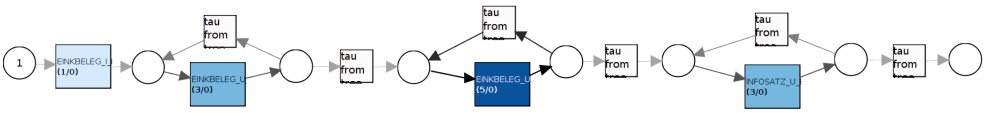


Figure 4.28: Conformance analysis of the model for the SAP dataset

Something interesting to note in this dataset with respect to the other environments is the existence of *resource* information. This means that we know who per-

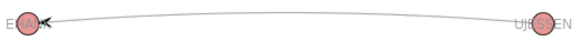
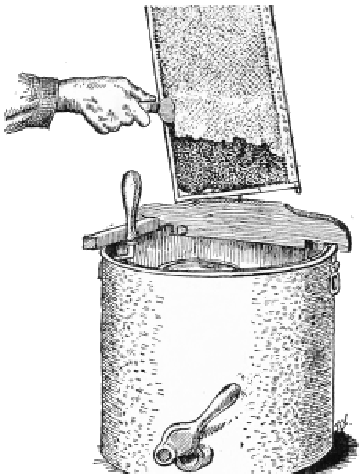


Figure 4.29: Social network for the SAP dataset

formed a change in the documents in SAP. This can be exploited to analyze the data discovering the social network behind the interactions. In this case, Figure 4.29 represents the discovered social network, showing the handout of work between resources *UJBSSEN* and *EHANI*.

4.4 Chapter Summary

In this chapter, we have demonstrated the applicability of the meta-model described in Chapter 3, with a special focus on the standardization of the analysis. Several real-life environments have been analyzed, for which we provided formal descriptions of their mapping on our meta-model. Moreover, we provided an implementation of all our techniques, which makes it possible to transform the data to populate our meta-model, capturing the data and process perspectives of the system under analysis. This allows for querying it in a standard way, obtaining the relevant cases for the business question at hand, in order to proceed with the analysis. Finally, a demonstration of the analysis is made, covering all the phases of the process, starting from the data extraction and continuing with data transformation, querying, process discovery, conformance, and performance analysis. The applicability of this ETL technique to such different environments provides a common ground to separate data extraction and analysis as different problems. In this way, an interface is generated that is much richer and powerful than the existing standards. To summarize, the OpenSLEX meta-model and the proposed ETL technique provide a standardization layer to simplify and generalize the analysis phase.



Capping melter. This also shows the proper method of removing cappings.

"Beekeeping: a discussion of the life of the honeybee and of the production of honey", Everett Franklin Phillips, 1923

5 Case Notion Discovery and Recommendation

In Chapter 3 we proposed a meta-model for process mining on databases. The applicability of this meta-model in practice on different environments has been demonstrated in Chapter 4. Chapter 5 builds on top of the previous work, assuming that events have been extracted from the source system, and provides the tools to build event logs for process mining analysis. First, we provide a formalization of the concept of a case notion. Next, we present a method for building event logs based on a defined case notion. Also, the concept of event log "interestingness" is discussed, together with a method to predict it. Finally, the technique is evaluated.

5.1 Introduction

Obtaining event logs is not a trivial matter. This is due to the fact that data come in many forms, while a lot of manual work and domain knowledge is needed to obtain meaningful event logs from it. The principal idea behind log building is to correlate events in such a way that they can be grouped into traces to form event logs. Classical approaches would use a common attribute to correlate events. This is a valid method in scenarios where the data schema has a star shape [45] (Figure 5.1.a): there is a central table, and the rest of the tables are directly related to it, with at least one column in common, which can be used as a case notion. However, we consider the scenario in which some pairs of events may not have any attribute in common. This is the case for a snowflake schema [45] (Figure 5.1.b), which resembles the shape of a star schema, with the difference that, at the nodes, we find tables that only hold

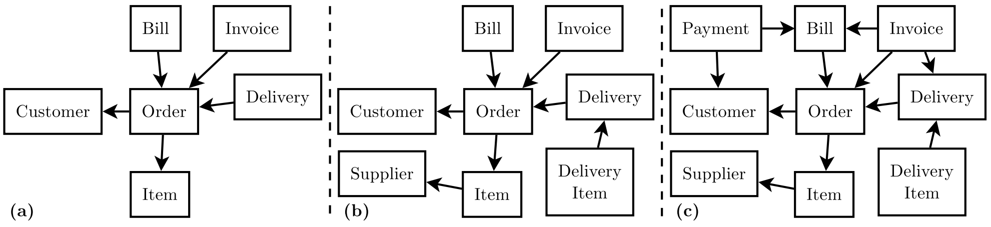


Figure 5.1: Example of database schema types: (a) star, (b) snowflake, and (c) arbitrary. The edges represent relationships, i.e., a foreign key in a table (source or the edge) pointing to a unique key in another table (target of the edge).

a transitive relation with the central table. In practice, we often find databases of which its schema presents a higher complexity than a star or snowflake structure (Figure 5.1.c). In that case, there are many combinations in which events can be grouped. These combinations cannot be arbitrary, but must obey some criteria with a business meaning, e.g., group the *invoice* and *delivery* events by means of the *invoice_id* field present in the former ones. Also, more complex combinations can be defined when transitive relations are considered for the grouping, e.g., group the *invoice*, *delivery*, and *bill* events according to the field *invoice_id* in delivery events and the field *delivery_id* in the bill events. Each of these examples captures what we will refer to as a *case notion*, i.e., a way to look at event data from a specific perspective.

When dealing with vast datasets from complex databases, the existence of many potential case notions is evident. Enterprise Resource Planning (SAP, Oracle EBS, Dolibarr), Hospital Information Systems (ChipSoft, GE Centricity, AGFA Integrated Care), and Customer Relationship Management (Salesforce, MS Dynamics, SugarCRM) are examples of systems powered by large databases where multi-perspective analysis can be performed. According to different case notions, many different event logs can be built. *The research problem we tackle in this chapter is how to choose the right perspective on the data, which is a crucial step in order to obtain relevant insights.* It is common practice to perform this selection by hand-written queries, usually by an analyst with the right domain knowledge about the system and process under study. However, when facing complex data schemas, writing such queries can become a very complicated task, especially when many tables are involved.

A naive way to tackle the exploration of complex databases is to generate all the possible case notions as combinations of tables. This can lead to many event log candidates, even for a small database. The combinatorial problem is aggravated in more complex scenarios, i.e., with hundreds of tables involved. Given a weakly connected[1] data schema of 90 tables, there exist 4 005 combinations of pairs of tables[2]. If we consider combinations of 3 tables instead, the number increases to 117 480,

[1]Weakly connected graph: a directed graph such that, after replacing all of its directed edges with undirected ones, it produces a connected graph. A connected graph is one such that, for any pair of nodes (a, b), there is a path from a to b.

[2]For a set of n elements (n tables), the number of k-combinations (combinations of k tables) is $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

even before considering the many different paths that could connect the tables in each combination. In such cases, the automated building of logs for all possible table combinations may still be possible, but has proven to be computationally very expensive: In the hypothetical case that building an event log would take 4 seconds on average, building the event logs for a data schema with 90 tables and 10 000 possible case notions would take approximately 11 hours. Even if we spend the time to compute all of them, we still need to inspect 10 000 event logs to find out which perspective is both meaningful and interesting.

A way to mitigate the combinatorial explosion is to reduce the case notion search space as much as possible. Identifying the most interesting event logs would help to prioritize the most promising views on the data for its analysis. The challenge of identifying the most promising views is related to the log quality problem. The *log quality problem* is concerned with identifying the properties that make an event log more suitable to be analyzed, i.e. the characteristics that increase the probability of obtaining valuable insights from the analysis of such an event log. The choices made during the log building process have an effect on the log quality [56]. Also, in the literature metrics are available to assess structural log properties [50], which may be important to assess log quality.

The main contributions described in this chapter are: (a) formally defining complex case notions to adopt different perspectives on event data; (b) automatically generating candidate case notions on a dataset; (c) assessing the quality of the resulting event logs; (d) automatically predicting an event log's quality before it is built; (e) sorting the case notions according to their relative quality from the analysis point of view. This drastically reduces the computational cost avoiding the generation of uninteresting event logs. In order to achieve these goals, data must be extracted from the original system and transformed to fit into a certain structure. This structure should be able to capture both the process and the data sides of the system under study. The techniques proposed in this paper have been implemented in a framework and evaluated with respect to related ranking algorithms. The approach yields promising results in terms of performance and accuracy on the computation of event log rankings.

To enable the application of process mining and the techniques proposed in this chapter, we need access to the database of the system under study. This information should be extracted and transformed to fit into a specific data structure. An appropriate structure has been previously defined in Chapter 3 as the OpenSLEX meta-model.

From now on, any reference to input or extracted data will assume to be in the form of a valid connected meta-model as described in Definition 24 (page 39). As we have seen, according to our meta-model description, *events* can be linked to *object versions*, which are related to each other by means of *relations*. These *relations* are instances of data model *relationships*. In database environments, this would be the equivalent of using foreign keys to relate table rows and knowing which events relate to each row. For the purpose of this chapter, we assume that pairwise correlations between events, by means of related object versions, are readily available in the input meta-model. This means that, prior to the extraction, we know the data schema, i.e., primary and foreign keys, and how events are stored in each table, e.g., which columns contain the timestamp and activity name of each event. The first precondition (knowing the data schema) is fair to assume in most real-life environment. Given the lack of automated approaches in the literature that tackle the challenge of event data discovery, the second precondition (knowing the events) requires having the right domain knowledge in order to extract events. The presented meta-model formalization sets the ground for the definition of *case notion* and *log* that will be presented in the coming sections.

5.2 Running Example

Extracting data from an information system's database is a complex task. Very often, we lack the domain knowledge needed to identify business objects and meaningful case notions. Also, understanding complex data schemas can be challenging when the number of tables is beyond what can be plotted and explored intuitively. Consider for example the SAP ERP system. This widespread ERP system is often a target for process mining analysis, as it is used in a multitude of organizations, and contains a huge amount of functionalities by means of configurable modules. SAP can run on different database technologies, but its instances always maintain a common data model which is well-known for its complexity. SAP represents a prime example because it is a widely used system. Nevertheless, the approach is highly generic and can be applied in different environments, e.g., alternative ERP tools such as Oracle EBS, HIS solutions such as ChipSoft, and CRM systems like Salesforce.

Figure 5.2 depicts the data model of a sample SAP dataset. This dataset, belonging to SAP IDES (Internet Demonstration and Evaluation System), is an instance of a fictitious organization. It contains more than 7M data objects of 87 different classes and more than 26k events corresponding to changes for a subset of the objects present in the database. In the diagram, classes are represented by squares, while edges show the relationships between classes, i.e., a foreign key in one table (edge source) pointing to a unique key in another table (edge target). Table names in SAP are codified in such a way that it is not easy to identify what these classes mean without further documentation. Also, most of the relevant classes are connected to many others. This makes it very difficult to plot the graph in such a way that clusters of classes can be easily identified.

Figure 5.3 shows in detail a small portion of the graph, where we observe that the *EKKO* class is linked, among others, to the *EKPO* class. Also, the *EBAN* class is connected to both. Additionally, the class *EKET* is linked to *EBAN*. According to the official documentation, both *EKKO* (header table) and *EKPO* (item table) refer to purchasing documents. The *EBAN* class contains information about purchase requisition and the *EKET* class contains scheduling agreement schedule lines. This could very well be a valid case notion, if we use the connection between the four tables to correlate the corresponding events in traces. However, there are many ways in which this correlation could be constructed. One-to-many relationships can exist between classes, which leads to the well-known problems of *data divergence* (several events of the same type are related to a single case) and *data convergence* (one event is

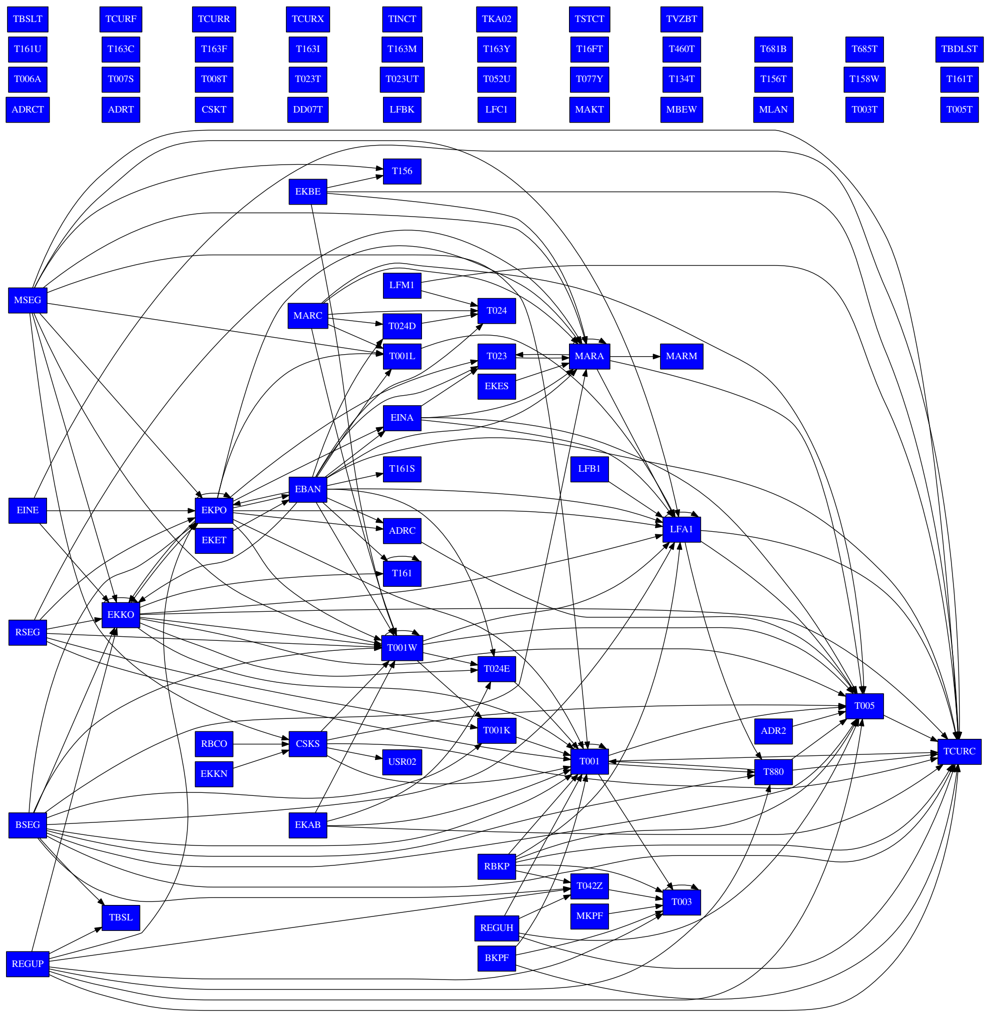


Figure 5.2: General view of the data model of the SAP dataset (the table attributes have been omitted).

related to multiple cases), as described in [72]. This means that the combination of a subset of classes can yield several, different event logs, depending on the choices made to correlate the events. Should all the purchase items be grouped in the same purchase requisition trace? Should one trace per purchase item exist? Would that mean that the same purchase requisition events would be duplicated in different traces? The fact that these choices exist makes the process of log building a non-trivial task. Section 5.3 provides a definition of case notion and presents a framework to build event logs effectively, taking into account the aforementioned choices in a formal manner.

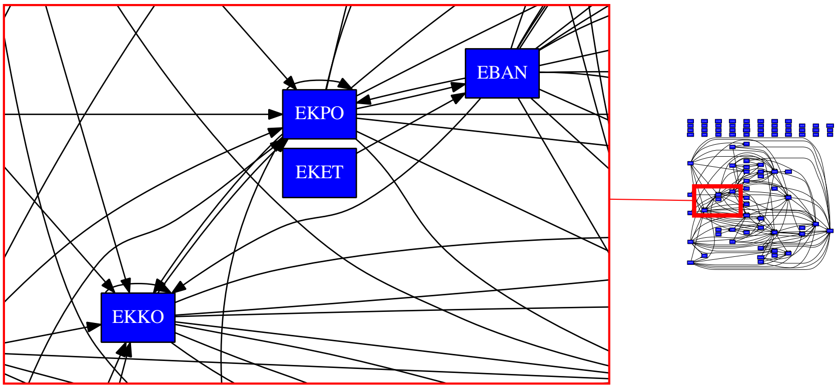


Figure 5.3: Detail of the data model of the SAP dataset. *EKKO* and *EKPO* tables refer to purchase documents, while *EBAN* contains information about purchase requisitions.

5.3 Case Notions and Log Building

The focus of this section is on defining what a case notion is, in order to build logs from event data. Relying on the meta-model structure to correlate events gives us the freedom to apply our log building technique to data coming from different environments, where SAP is just an example. As long as the existing data elements can be matched to the class, object and event abstractions, event correlation will be possible. Therefore, our log building technique will be feasible. The fact that this kind of data and correlations can be obtained in real-life environments has been previously demonstrated in [42]. Our approach defines case notions based on the data model of the dataset (classes and relationships) and projects the data onto it (objects, object versions, and events) to build traces with correlated events.

5.3.1 Defining Case Notions

We define a case notion (Definition 25) as an annotated rooted tree in which there is always a root node (root class of the case notion). There can be a set of additional regular class nodes, together with some converging class nodes, as children of the root node or other nodes of the subtrees. The root node is the main class of the case notion and triggers the creation of a new case identifier for each object that belongs to it (e.g. a case identifier for a purchase order). Regular nodes will force the creation of a new case identifier when several of its objects relate to one root or regular object (e.g. several deliveries of the same order will result in one case identifier for each delivery). Converging nodes are the ones that allow one case identifier to refer to objects of that same class (e.g., several delivery items linked to the same delivery will be grouped in under the same case identifier).

Definition 25 (Case Notion) *Let us assume a data model* $DM = (CL, AT,$

$classOfAttribute, RS, sourceClass, targetClass)$. *We define a case notion as a tuple* $CN = (C, root, children, CONV, IDC, rsEdge)$ *such that:*
- $C \subseteq CL$ *is the set of classes involved in the case notion,*
- $root \in C$ *is the root class in the case notion tree,*
- $children \in C \rightarrow \mathscr{P}(C)$ *is a function returning the children of a class in the case notion tree,*
- $CONV \subseteq C$ *is the set of classes of the case notion for which convergence is applied. If a class* c *belongs to* $CONV$*, all the members of the subtree of* c *must belong to this set, i.e.,* $\forall c \in CONV : children(c) \subseteq CONV$*,*
- $IDC = C \setminus CONV$ *is the set of identifying classes that will be used to uniquely identify cases of this case notion,*
- $rsEdge \in (C \times C) \rightarrow RS$ *is a function returning the relationship of the edge between two classes in the tree such that,* $\forall c \in C : \forall c' \in children(c) : \exists rs \in RS : \{c, c'\} = \{sourceClass(rs), targetClass(rs)\} \wedge rsEdge(c, c') = rs$*.*

Table 5.1: Sample object, version and event identifiers for the classes involved in the case notion.

Class	ObjectID	VersionID	EventID	RelationID
EKET	a1	av1	ae1	bv1
EKET	a1	av2	ae2	bv2
EKET	a2	av3	ae3	bv3
EBAN	b1	bv1	be1	-
EBAN	b1	bv2	be2	-
EBAN	b2	bv3	be3	-
EKKO	c1	cv1	ce1	bv2
EKKO	c2	cv2	ce2	bv2
EKKO	c3	cv3	ce3	bv3
EKPO	d1	dv1	de1	cv1
EKPO	d2	dv2	de2	cv1
EKPO	d3	dv3	de3	cv2
EKPO	d4	dv4	de4	cv3

Figure 5.4 shows an example of a case notion combining classes *EBAN*, *EKET*, *EKKO*, and *EKPO*. The class *EBAN* is the root of the case notion. The class *EKET* is a regular child of the root node, while the child node *EKKO* is a converging class. By inheritance, the node *EKPO* is a converging class as well, given that it belongs to a subtree of the converging class *EKKO*. Therefore, Figure 5.4 is the graphical representation of the case notion cn for which $C = \{EBAN, EKET, EKKO, EKPO\}$, $root = EBAN$, $CONV = \{EKKO, EKPO\}$, $IDC = \{EBAN, EKET\}$, $children \in C \rightarrow \mathscr{P}(C)$ such that $children(EBAN) = \{EKET, EKKO\}, children(EKKO) = \{EKPO\}, children(EKPO) = \emptyset$, and $children(EKET) = \emptyset$, and $rsEdge \in (C \times C) \rightarrow RS$ such that $rsEdge(EKET, EBAN) = fk\_eket\_to\_eban$[3], $rsEdge(EKKO, EBAN) =$

[3] *fk_** stands for *"foreign key"*, e.g., *fk_eket_to_eban* represents a foreign key from table *EKET* to table *EBAN*.

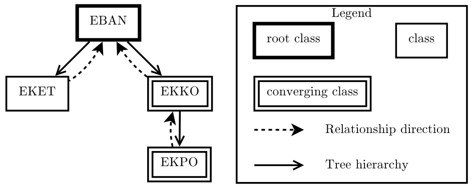


Figure 5.4: Sample of a case notion, represented as an annotated rooted tree.

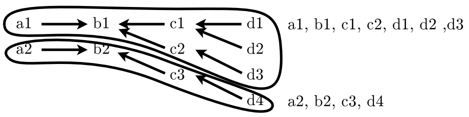


Figure 5.5: Links between objects of classes EKET (*a1, a2*), EBAN (*b1, b2*), EKKO (c1, c2, c3), and EKPO (d1, d2, d3, d4). The objects have been grouped in two sets, corresponding to the case identifiers computed for the case notion of Figure 5.4.

fk_ekko_to_eban, and $rsEdge(EKPO, EKKO) = fk\_ekpo\_to\_ekko$. According to this case notion, each trace will contain events belonging only to one *EBAN* object, only one *EKET* object, but to any *EKKO* or *EKPO* objects that hold a relation with the *EBAN* object represented by the trace. This is due to the fact that *EKKO* and *EKPO* are defined as converging classes in our case notion. The log building process is described in greater detail below.

5.3.2 Building a Log

The process of building an event log can be seen as the projection of a dataset on a certain case notion. First, a set of case identifiers will be constructed, which will determine the objects that will be correlated per trace. Definition 26 describes in more detail how this set of case identifiers is generated. Figure 5.5 will be used in this section as an example to illustrate the method.

Definition 26 (Case Identifiers) *Let us assume a valid connected meta-model instance CMI and a case notion* $CN = (C, root, children, CONV, IDC, rsEdge)$. *We define CI as the maximal set*[4] *of case identifiers such that, each case identifier* $ci \in CI$ *is a set of objects* $ci = \{o \in OC \mid classOfObject(o) \in C\}$ *and the following properties apply:*

- $\forall o \in ci : classOfObject(o) \in IDC \Rightarrow (\exists o' \in ci : classOfObject(o') = classOfObject(o) \Rightarrow o' = o)$, *i.e., cannot exist two objects per identifying class in each case identifier,*
- $\exists o \in ci : classOfObject(o) = root$, *i.e., one object of the case identifier belongs to the root,*
- $R \subseteq (ci \times ci) = \{(o, o') | \exists (rs, ov, ov') \in REL : c = classOfObject(o) \land c' = classOfObject(o') \land objectOfVersion(ov) = o \land objectOfVersion(ov') = o' \land rs = rsEdge(c, c') \land sourceClass(rs) = c \land targetClass(rs) = c'\}$, *i.e., R is a relation between two objects of the case identifier such that both objects have at least one link in the original data for a relationship*

[4] A is a maximal set for property P if: (a) A satisfies property P and (b) $\forall B \supseteq A$ satisfying property P: $B = A$.

considered in the case notion. To improve readability, we can say that $oRo' \iff (o,o') \in R$*,*
- $|ci| > 1 \Rightarrow \forall (o,o') \in (ci \times ci) : oR^+o'$*, i.e., as long as the case identifier contains more than one object, any pair of objects must belong to the transitive closure*[5] *of the relation* R*, i.e., directly or transitively related through objects of the case identifier.*

Let us consider the sample dataset in Table 5.1. It corresponds to the tables *EBAN*, *EKET*, *EKKO*, and *EKPO*. In total there are 11 objects ($\{a1, a2, b1, b2, c1, c2, c3, d1, d2, d3, d4\}$), 13 object versions ($\{av1, av2, av3, bv1, bv2, bv3, cv1, cv2, cv3, dv1, dv2, dv3, dv4\}$), and 13 events ($\{ae1, ae2, ae3, be1, be2, be3, ce1, ce2, ce3, de1, de2, de3, de4\}$). Additionally, there are 10 relations between object versions ($\{av1 \rightarrow bv1, av2 \rightarrow bv2, av3 \rightarrow bv3, cv1 \rightarrow bv2, cv2 \rightarrow bv2, cv3 \rightarrow bv3, dv1 \rightarrow cv1, dv2 \rightarrow cv1, dv3 \rightarrow cv2, dv4 \rightarrow cv3\}$).

The first step to build the event log corresponding to the case notion in Figure 5.4 is to build the set of case identifiers. First, we have to find the maximal set of case identifiers that comply with the constrains set by the case notion at hand, i.e. (a) all the objects must belong to the classes in the case notion, (b) at least one object per case identifier must belong to the root class of the case notion, (c) two objects of the same case identifier cannot belong to the same identifying class of the case notion, and (d) all the objects in the same case identifier must be related, either directly or transitively, by means of the relationships specified in the case notion.

Going back to our example, we will construct the set of case identifiers by observing the relations between objects (Figure 5.5). Knowing that $\{b1, b2\}$ are the objects belonging to the *EBAN* class and that *EBAN* is the root class of the case notion, we know that exactly one of these objects must be in each of the resulting traces. That means we will generate, at least, two traces. Objects $\{a1, a2\}$ belong to the class *EKET*, which is the other identifying class of the case notion. Only one of these objects is allowed per trace. In this case, each one of them is related to a different *EBAN* object. Because *EKET* and *EBAN* are the only identifying classes of the case notion, we can combine their objects already to create a (non-maximal) set of case identifiers $CI' = \{ci1', ci2'\}$:

$$ci1' = \{a1, b1\}$$
$$ci2' = \{a2, b2\}$$

The next class to look at in the case notion hierarchy is *EKKO*. There are three objects ($\{c1, c2, c3\}$) belonging to this class. Two of them ($\{c1, c2\}$) are related to the *EBAN* object $b1$. Given that it is a converging class, we can put them in the same case identifier, in this case $ci1'$. The other object ($c3$) is related to the *EBAN* object $b2$. Therefore, it will be inserted in the case identifier $ci2'$. We proceed analogously with the *EKPO* objects $\{d1, d2, d3, d4\}$, given that *EKPO* is a converging class in our case notion as well. Finally, the maximal case identifiers set $CI = \{ci1, ci2\}$ is:

$$ci1 = \{a1, b1, c1, c2, d1, d2, d3\}$$
$$ci2 = \{a2, b2, c3, d4\}$$

[5] R^+ is the transitive closure of a binary relation R on a set X if it is the smallest transitive relation on X containing R.

Once the case identifiers have been generated, it is possible to build the log in its final form. First, we introduce some useful notation in Definition 27.

Definition 27 (Shorthands I) *Given a valid connected meta-model instance CMI, a case notion $CN = (C, root, children, CONV, IDC, rsEdge)$ and a maximal set of case identifiers CI, we define the following shorthands:*

- $Act_o = \{act \in AC \mid \exists (e, ov) \in dom(eventToOVLabel) : objectOfVersion(ov) = o \land activityOfAI(eventAI(e)) = act\}$, *i.e., the set of activities of the activity instances related to an object through its versions and events,*
- $ActC_c = \{act \in AC \mid \exists (e, ov) \in dom(eventToOVLable) : objectOfVersion(ov) = o \land activityOfAI(eventAI(e)) = act \land classOfObject(o) = c\}$, *i.e., the set of activities related to a class through its activity instances, events, versions and objects,*
- $O_c = \{o \in OC \mid classOfObject(o) = c\}$, *i.e., the set of objects of a certain class* $c \in C$,
- $EvO_o = \{e \in EV \mid \exists (e, ov) \in dom(eventToOVLabel) : objectOfVersion(ov) = o\}$, *i.e., the set of events of a certain object* $o \in OC$,
- $EvC_c = \{e \in EV \mid \exists (e, ov) \in dom(eventToOVLabel) : classOfObject(objectOfVersion(ov)) = c\}$, *i.e., set of events of a certain class* $c \in C$,
- $E_{ai} = \{e \in EV \mid ai \in AI \land eventAI(e) = ai\}$, *i.e., set of events of a certain activity instance* $ai \in AI$.

In order to build the final log, we will map a set of activity instances to each object and group them per case identifier to form traces. According to the definition of the OpenSLEX meta-model, an activity instance is a set of events that belong to the same activity and case, e.g., correlated events with different life-cycle of the same activity (*start* and *complete* events). In our example, for the sake of clarity, we assume that each activity instance is a singleton with a single event. In fact, we will represent traces as a set of events. Definition 28 provides a formal description of a log and how to build it from a maximal set of case identifiers.

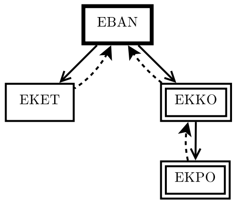
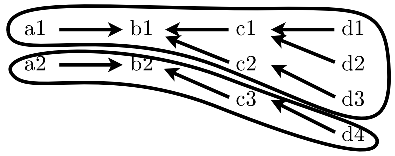
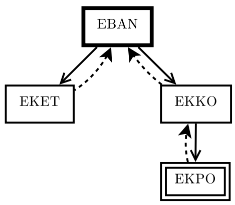
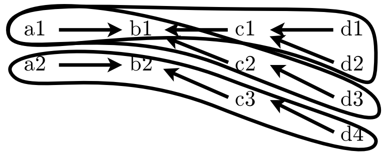
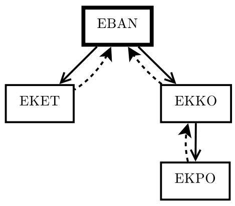
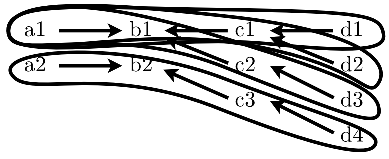
Definition 28 (Log) *Given a valid connected meta-model instance CMI, a case notion $CN = (C, root, children, CONV, IDC, rsEdge)$ and a maximal set of case identifiers CI, we define a log $l \in CI \rightarrow \mathscr{P}(AI)$ as a deterministic mapping between the set of case identifiers and the powerset of activity instances, such that each of the activity instances in the mapped set is linked to at least one object of the case identifier, i.e., for all $ci \in CI : l(ci) = \{ai \in AI \mid \exists e \in EV : ai = eventAI(e) \land \exists ov \in OV : (e, ov) \in dom(eventToOVLabel) \land objectOfVersion(ov) \in ci\}$.*

Assuming that, in our example, each activity instance is represented by a single event, we can build the final log l as the following mapping:

$$\begin{aligned} &CI \rightarrow \mathscr{P}(AI) \\ l : \ &ci1 = \{ae1, ae2, be1, be2, ce1, ce2, de1, de2, de3\} \\ &ci2 = \{ae3, be3, ce3, de4\} \end{aligned}$$

Of course, different variations of case notions will lead to different event logs, given that the grouping rules will change. Table 5.2 shows three different case notions, as well as the corresponding case identifiers and final traces. The first row (a) is based on the case notion in Figure 5.4, representing the same example we have just analyzed. Case notions (b) and (c) are variations of the case notion (a). In (b), the *EKKO* class has been promoted to be an identifying class. This provokes the generation of an additional case identifier, since objects $\{c1, c2\}$ cannot coexist in the case identifier anymore. In (c), also the *EKPO* class has been transformed into an identifying class. This triggers the creation of another case identifier, since the objects $\{d1, d2, d3, d4\}$ cannot belong to the same case identifier either. These examples show the impact of converging and identifying classes in the output of the log building process.

Table 5.2: Case identifiers and final traces built from the sample dataset, according to each of the three case notions.

ID	Case Notion	Case Identifiers & Traces
a		 a1, b1, c1, c2, d1, d2 ,d3 a2, b2, c3, d4 Trace 1: $\{ae1, ae2, be1, be2, ce1, ce2, de1, de2, de3\}$ Trace 2: $\{ae3, be3, ce3, de4\}$
b		 a1, b1, c1, d1, d2 a1, b1, c2, d3 a2, b2, c3, d4 Trace 1: $\{ae1, ae2, be1, be2, ce1, de1, de2\}$ Trace 2: $\{ae1, ae2, be1, be2, ce2, de3\}$ Trace 3: $\{ae3, be3, ce3, de4\}$
c		 a1, b1, c1, d1 a1, b1, c1, d2 a1, b1, c2, d3 a2, b2, c3, d4 Trace 1: $\{ae1, ae2, be1, be2, ce1, de1\}$ Trace 2: $\{ae1, ae2, be1, be2, ce1, de2\}$ Trace 3: $\{ae1, ae2, be1, be2, ce2, de3\}$ Trace 4: $\{ae3, be3, ce3, de4\}$

These definitions make it possible to create specialized logs that capture behavior from different perspectives. However, the combinatorial explosion problem makes it practically impossible to explore all the case notions for large and complex data models. This means that we must focus our efforts on the most interesting perspectives to obtain insights without being overwhelmed by excessive amounts of information. The following section proposes a set of metrics to assess the "interestingness" of a case notion, based on measurable quality features of the resulting event log.

5.4 Log Quality: Is my Log Interesting?

The log quality problem concerns the identification of characteristics that make event logs interesting to be analyzed. This problem is not new to the field. Some authors have studied how the choices made during the log building process can affect the log quality [56] and have developed procedures to minimize the negative impact. Other authors have tried to define metrics to assess different log properties from the structural point of view [50]. In this work, we aim at assessing the quality of an event log in an automated way. For that purpose, we adopt some metrics from [50], that will give us an idea of the structural and data properties that a log should possess in order to be an interesting candidate. In the scope of our meta-model and the logs we are able to build, we need to adapt these concepts to be able to compute them. Considering a valid connected meta-model instance *CMI*, a case notion *CN*, a set of case identifiers CI, and a log l, we define the following three equations to match the structure of our meta-model.
Support (SP) (Equation 5.1): number of traces present in an event log:

$$SP(l) = |dom(l)| = |CI| \tag{5.1}$$

Level of detail (LoD) (Equation 5.2): average number of unique activities per trace:

$$LoD(l) = \frac{\sum\limits_{ci \in CI} |\bigcup\limits_{ai \in l(ci)} activityOfAI(ai)|}{SP(l)} = \frac{\sum\limits_{ci \in CI} |\bigcup\limits_{o \in ci} Act_o|}{|CI|} \tag{5.2}$$

Average number of events (AE) (Equation 5.3): average number of events per trace:

$$AE(l) = \frac{\sum\limits_{ci \in CI} |\bigcup\limits_{ai \in l(ci)} E_{ai}|}{SP(l)} = \frac{\sum\limits_{ci \in CI} |\bigcup\limits_{o \in ci} EvO_o|}{|CI|} \tag{5.3}$$

When analyzing processes, intuitively, it is preferred to have event logs with as many cases as possible, i.e., higher support (Equation 5.1), but not too many activities per case, i.e., a reasonable level of detail (Equation 5.2). The reason for this is that the complexity of the resulting model, and therefore its interpretation, is closely related to the number of activities it needs to represent. However, too few activities result in very simple models that do not capture any interesting patterns we want to observe. Also, we try to avoid cases with extremely long sequences of events, i.e., large average number of events per trace (Equation 5.3), because of the difficulty to interpret the models obtained when trying to depict the behavior. However, too short sequences of events will be meaningless if they represent incomplete cases.

Therefore, while we would like to maximize the support value (5.1), i.e., give priority to logs with a higher number of traces, we cannot say the same for the level of detail (5.2) and average number of events per case (5.3). These last two metrics will find their optimality within a range of acceptable values, which will depend on the domain of the process and taste of the user, among other factors. Given the differences between the pursued optimal values for each of the metrics, the need for a scoring function becomes evident. It is required to be able to effectively compare

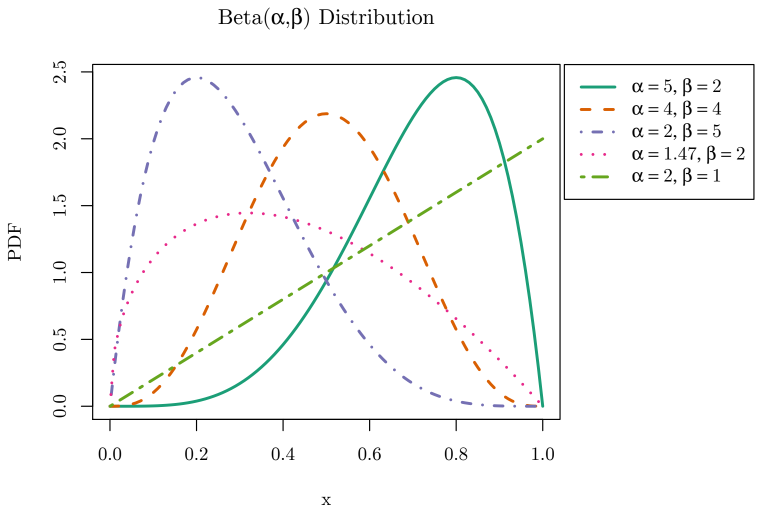


Figure 5.6: Sample of beta distribution curves for different values of the α and β parameters.

log metrics. A candidate is the beta distribution. Note that the choice of the scoring function is not restricted by the approach and could be replaced by any distribution more appropriate to the setting of application.

The beta distribution is defined on the interval $[0,1]$ and has two shape parameters, α and β. The values of these two parameters determine the shape of the curve, its mean, mode, variance, etc. Also, the skewness of the distribution can be controlled by choosing the right combination of parameters (See Figure 5.6). This allows one to define a range of values for which the *probability density function* (PDF) of the beta distribution (Equation 5.4) will return higher scores as they approximate to the mode.

$$Beta_{PDF}(x;\alpha,\beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}, \text{ where } B(\alpha,\beta) \text{ is the Euler beta function.} \tag{5.4}$$

The input values will get a lower score as they get farther from the mode. One advantage of this distribution is that it is possible to define a mode value different from the mean, i.e., to shape an asymmetric distribution. Figure 5.6 shows examples of beta distributions for different values of α and β.

The parameters α and β can be estimated based on the mode and approximate inflection points of the desired PDF [90]. We show an example considering only the mode. If we are interested in event logs with a *level of detail* close to 7, we need to estimate the values of α and β to obtain a PDF with mode 7. First, we scale the value. If the minimum and maximum values for LoD are 1 and 20, then the scaled mode is 0.32. Assuming that we are after a unimodal PDF and $\alpha,\beta > 1$, we use Equation 5.5

to compute the mode:

$$mode = \frac{\alpha - 1}{\alpha + \beta - 2} \quad \text{for } \alpha, \beta > 1 \tag{5.5}$$

Given the desired $\boldsymbol{mode}$, we can fix the value of one of the shape parameters and estimate the other one using Equation 5.5:

$$est(mode) = \begin{cases} \beta = 2, \alpha = \frac{1}{1-mode}, & \text{if } mode < 0.5 \Rightarrow \text{positively skewed} \\ \alpha = 2, \beta = \frac{1-4mode}{mode}, & \text{if } mode > 0.5 \Rightarrow \text{negatively skewed} \\ \alpha, \beta = 2, & \text{if } mode = 0.5 \Rightarrow \text{symmetric} \end{cases} \tag{5.6}$$

Therefore, for the mode 0.32, the PDF is positively skewed. Using Equation 5.6 we evaluate $\boldsymbol{est}(0.32)$ to obtain the values $\beta = 2$ and $\alpha = 1/(1-0.32) = 1.47$. The resulting PDF can be observed in Figure 5.6 (dotted curve). This is a basic yet effective method to set the shape parameters of the beta function using domain knowledge, i.e., the optimal value that we desire to score higher. Once the parameters α and β have been selected, we can compute the scores of the previous log metrics by means of the following score function:

$$score(f, x_i, X, \alpha, \beta) = Beta_{PDF}(scaled(f, x_i, X); \alpha, \beta) \tag{5.7}$$

Here, f is a function to compute the metric to be scored (e.g., *SP*, *LoD* or *AE*), x_i is the input of function f (e.g., a log l), X is the set of elements with respect to which we must scale the value of $f(x_i)$ (e.g., a set of logs L), α and β are the parameters of the beta probability distribution function, and $\boldsymbol{scaled}(f, x_i, X)$ is a rescaling function such that:

$$scaled(f, x_i, X) = \frac{f(x_i) - \min\limits_{x_j \in X}\{f(x_j)\}}{\max\limits_{x_j \in X}\{f(x_j)\} - \min\limits_{x_j \in X}\{f(x_j)\}} \tag{5.8}$$

With the $\boldsymbol{score}$ function in Equation 5.7, first we perform feature scaling (Equation 5.8). Next, we apply the beta distribution function (Equation 5.4) with the corresponding α and β parameters. With respect to the **support** of the log, the score will be the result of scaling the support feature $(SP(l))$ with respect to the set of possible logs L and applying the beta probability distribution function. As the purpose, in this case, is to give a higher score to higher support values, we will set the parameters α_{SP} and β_{SP} such that the probability distribution function resembles an ascending line (e.g., $\alpha = 2$ and $\beta = 1$ in Figure 5.6):

$$s_{sp}(l, L) = score(SP, l, L, \alpha_{SP}, \beta_{SP}) \tag{5.9}$$

To score the **level of detail**, we let the parameters α_{LoD} and β_{LoD} to be tuned according to the preference of the user:

$$s_{lod}(l, L) = score(LoD, l, L, \alpha_{LoD}, \beta_{LoD}) \tag{5.10}$$

The score of the **average number of events** per case is computed in the same way, using the appropriate values for the parameters α_{AE} and β_{AE}:

$$s_{ae}(l, L) = score(AE, l, L, \alpha_{AE}, \beta_{AE}) \tag{5.11}$$

The "interestingness" of a log l with respect to all the logs L can be defined by the combination of the score values for each of the previous metrics. In order to combine the scores for each log metric, a global scoring function $gsf \in L \times \mathscr{P}(L) \Rightarrow \mathbb{R}$ can be used, which takes a log l and a set of logs L, and returns the score of l with respect to L. The approach does not depend on the choice of this function, and it can be replaced by any custom one. For the purpose of demonstrating the feasibility of this approach, we define the global scoring (or "log interestingness") function as the weighted average of the three previous scores. The weights (w_{sp}, w_{sp}, w_{sp}) and the parameters of the beta distribution (α_{SP}, β_{SP}, α_{LoD}, β_{LoD}, α_{AE}, β_{AE}) can be adjusted by the user to balance the features according to their interest.

$$gsf(l, L) = w_{sp} \cdot s_{sp}(l, L) + w_{lod} \cdot s_{lod}(l, L) + w_{ae} \cdot s_{ae}(l, L) \tag{5.12}$$

The log "interestingness" scoring function (Equation 5.12) proposed in this section aims at giving an indication of how likely is that a log will be of interest, with respect to the other candidates, given a set of parameters. Table 5.3 shows the top 8 discovered case notions of the sample SAP dataset, according to the computed score. We see that the tables involved in the purchase requisition process represent a relevant case notion candidate for this specific dataset. The main contribution until now is not the specific scoring function, but the framework that enables the assessment and its configuration. However, this framework still requires a log to be generated in order to be subjected to evaluation. Taking into account that the final goal is to assess log "interestingness" at a large scale automatically, we need better ways to score case notions before the corresponding logs are built. The following section explores this idea, proposing a method to predict log interestingness.

5.5 Predicting Log Interestingness

If an event log is completely created from an extracted dataset, then it is straightforward to assess the actual "interestingness". However, as explained before, for large databases it is infeasible to compute all candidates. In order to mitigate this problem and save computation time, we aim at approximating the value of the metrics considered in Section 5.4 for a certain case notion, before the log is computed. For this purpose, we define bounds for the log metrics, given a certain case notion. The goal is to restrict the range of uncertainty and improve the prediction accuracy. In fact, at the end of this section, the bounds will be used to define a custom predictor for each of the log metrics. First, we try to set bounds to the support of a log. From Equation 5.1 we see that the support of a log is equal to the domain of the mapping, i.e., the number of case identifiers of the log. Definition 26 shows that the number of case identifiers depends on the combination of objects belonging to the identifying

Table 5.3: Top 8 discovered case notions, sorted by score with parameters ($\alpha_{SP} = 2$, $\beta_{SP} = 1$, $\alpha_{LoD} = 4.16$, $\beta_{LoD} = 1$, $\alpha_{AE} = 1.28$, $\beta_{AE} = 1.53$, $w_{sp} = 0.3$, $w_{lod} = 0.3$, and $w_{ae} = 0.3$). The α and β parameters have been estimated based on desired min, max, and mode values for the corresponding beta distribution ($LoD_{min} = 2$, $LoD_{max} = 10$, $LoD_{mode} = 4$, $AE_{min} = 4$, $AE_{max} = 30$, and $AE_{mode} = 8$). The values for SP, LoD, and AE have been scaled.

	Root	Tables	SP'	LoD'	AE'	Score
1	EBAN	EKPO, EINE, EBAN, EKKO, LFA1	0.54	1.00	0.60	1.90
2	EINE	EKPO, EINE, EBAN, EKKO, LFA1	0.70	0.95	0.65	1.79
3	EBAN	EKPO, EINE, EBAN, MARA	0.28	1	0.69	1.73
4	EKPO	EKPO, EINE, EBAN, EKKO, LFA1	0.80	0.87	0.63	1.60
5	EKKO	EKPO, EINE, EBAN, EKKO, LFA1	0.55	0.88	0.47	1.53
6	EINE	EKPO, EINE, EBAN, EKKO	0.70	0.85	0.56	1.52
7	EBAN	EKPO, EINE, EBAN, EKKO	0.54	0.87	0.48	1.51
8	EINE	EKPO, EINE, EBAN, MARA	0.45	0.89	0.71	1.44

classes of the case notion (*IDC*). Given that every case identifier must contain one object of the root class, that only one object of the root class is allowed per case identifier, and that the set of case identifiers is a maximal set, we can conclude that the set of case identifiers will contain at least one case identifier per object in the root class:

Bound 1 (Lower Bound for the Support of a Case Notion) *Given a valid connected meta-model instance CMI, a case notion* $CN = (C, root, children, CONV, IDC, rsEdge)$*, a maximal set of case identifiers CI, and the corresponding log l we see that* $\forall ci \in CI : \exists o \in ci : classOfObject(o) = root \iff \forall o \in O_{root} : \exists ci \in CI : o \in ci \Rightarrow |CI| \geq |O_{root}|$*. Therefore, we conclude that:* $SP(l) \geq \lfloor SP(CN) \rfloor = |O_{root}|$

For a case identifier to be transformed into an actual trace, at least an event must exist for the root object involved in it. For the sake of simplicity, Bound 1 assumes that at least one event exists for every object in the root class. This has been taken into account in the implementation, considering only objects of the root class that contain at least one event.

Each of the case identifiers is a combination of objects. Also, exactly one object of the root class and no more than one object of each identifying class (classes in *IDC*) can exist per case identifier. This leads to the following upper bound for support:

Bound 2 (Upper Bound for the Support of a Case Notion) *Given a valid connected meta-model instance CMI, a case notion* $CN = (C, root, children, CONV, IDC, rsEdge)$*, a maximal set of case identifiers CI, and the corresponding log l, we define a maximal set* CI' *for which the following properties hold:*

a) $\forall ci \in CI' : \forall o \in ci : classOfObject(o) \in IDC \Rightarrow \exists o' \in ci : classOfObject(o) = classOfObject(o') \iff o = o'$*, i.e., only one object per class belongs to the case identifier,* *b)* $\forall ci \in CI' : \exists o \in ci : classOfObject(o) = root$*, i.e., one object of the root class must always belong to the case identifier.*

This implies that CI' *contains all the possible combinations of one or zero objects of each class in* IDC*, except for the* $root$ *class, that must always be represented by an object in the case identifier. That means that* $|CI'| = |O_{root}| \cdot \prod_{c \in \{C \setminus root\}} (|O_c| + 1)$*. Given that* CI' *is less restrictive than* CI*, we know that* $CI' \supseteq CI \Rightarrow |CI'| \geq |CI|$*. Therefore:* $SP(l) \leq \lceil SP(CN) \rceil = |O_{root}| \cdot \prod_{c \in \{C \setminus root\}} (|O_c| + 1)$

Following the same logic to set a lower bound for support, we know that all the objects that belong to the root class will be involved in at least one case identifier. However, the number of traces is still unknown if the log has not been built and we can only consider it as the maximum possible, i.e., the upper bound of the support. Therefore, a lower bound for the level of detail will be given by the sum of the unique activities per object of the root class divided by the maximum number of case identifiers. If we consider that the additional case identifiers (beyond the number of objects of the root class) will, at least, add a unique number of activities equal to the minimum number of activities per object of the root class, we can get a better lower bound as described below:

Bound 3 (Lower Bound for the LoD of a Case Notion) *Given a valid connected meta-model instance* CMI*, a case notion* $CN = (C, root, children, CONV, IDC, rsEdge)$*, a maximal set of case identifiers* CI*, and the corresponding log* l*, we see that* $\forall ci \in CI : \exists o \in ci : classOfObject(o) = root \iff \forall o \in O_{root} : \exists ci \in CI : o \in ci \Rightarrow \forall ci \in CI : \bigcup_{o \in ci} Act_o \supseteq \bigcup_{o \in (ci \cap O_{root})} Act_o$*. Additionally, we know that* $\sum_{ci \in CI} |\bigcup_{o \in (ci \cap O_{root})} Act_o| \geq (\sum_{o \in O_{root}} |Act_o|) + (|CI| - |O_{root}|) \cdot \min_{o \in O_{root}} \{|Act_o|\}$*. Therefore:*

$$LoD(l) \geq \lfloor LoD(CN) \rfloor = \frac{(\sum_{o \in O_{root}} |Act_o|) + (\lceil SP(CN) \rceil - |O_{root}|) \cdot \min_{o \in O_{root}} \{|Act_o|\}}{\lceil SP(CN) \rceil}$$

A lower bound for LoD is given by the lower bound of the sum of the unique activities per case, divided by the upper bound on the number of cases. We know that, at least, one case will exist per object belonging to the root class. That is why the sum of the unique activities per object of the root class is added on the top part of the formula. Also, because these objects could be involved in more than one case, to a maximum of $\lceil SP(CN) \rceil$ cases, we add the minimum number of unique activities they could have and multiply it by the maximum number of additional case identifiers. This will always be a lower bound given that the number of activities we add at the upper part for the additional case identifiers will always be equal or lower than the average. Not adding these extra case identifiers would still result in a lower bound, but an extremely low one since the divisor is usually an overestimation for the number of possible case identifiers.

With respect to the upper bound for the level of detail, we need to consider the most extreme situation. This is caused by a case identifier that contains one object per identifying class and one or more objects per converging class, such that, for each object, the events related to it represent all the possible activities. For this case identifier, the number of unique activities will be the sum of the number of unique activities per class involved. However, there is a way to restrict this bound. If we count the number of unique activities for the events of each object, and find the maximum per class, the upper bound will be given by the sum of the maximum number of unique activities per object for all the identifying classes, plus the total of unique activities per converting class involved in the case notion:

Bound 4 (Upper Bound for the LoD of a Case Notion) *Given a valid connected meta-model instance CMI, a case notion* $CN = (C, \boldsymbol{root}, \boldsymbol{children}, CONV, IDC, \boldsymbol{rsEdge})$*, a maximal set of case identifiers CI, and the corresponding log l, we know that,* $\forall c \in C : \forall o \in O_c : |Act_o| \leq \max_{o' \in O_c}\{|Act_{o'}|\}$*. This implies that,* $\forall ci \in CI : |\bigcup_{o \in ci} Act_o| \leq \sum_{c \in IDC} \max_{o \in O_c}\{|Act_o|\} + \sum_{c\ inCONV} |ActC_c|$*. Therefore:*

$$LoD(l) \leq \lceil LoD(CN) \rceil = \frac{|CI| \cdot (\sum_{c \in IDC} \max_{o \in O_c}\{|Act_o|\} + \sum_{c\ inCONV} |ActC_c|)}{|CI|} =$$
$$\sum_{c \in IDC} \max_{o \in O_c}\{|Act_o|\} + \sum_{c\ inCONV} |ActC_c|$$

The same reasoning used to obtain a lower bound for the level of detail can be applied in the case of the average number of events per trace. Only that, in this case, instead of counting the number of unique activities, we count the number of events per object:

Bound 5 (Lower Bound for the AE of a Case Notion) *Given a valid connected meta-model instance CMI, a case notion* $CN = (C, \boldsymbol{root}, \boldsymbol{children}, CONV, IDC, \boldsymbol{rsEdge})$*, a maximal set of case identifiers CI, and the corresponding log l, we see that* $\forall \boldsymbol{ci} \in CI : \exists \boldsymbol{o} \in \boldsymbol{ci} : \boldsymbol{classOfObject}(o) = \boldsymbol{root} \iff \forall \boldsymbol{o} \in O_{root} : \exists ci \in CI : \boldsymbol{o} \in \boldsymbol{ci} \Rightarrow \forall \boldsymbol{ci} \in CI : \bigcup_{o \in ci} EvO_o \supseteq \bigcup_{o \in (ci \cap O_{root})} EvO_o$*. Additionally, we know that* $\sum_{ci \in CI} |\bigcup_{o \in (ci \cap O_{root})} EvO_o| \geq (\sum_{o \in O_{root}} |EvO_o|) + (|CI| - |O_{root}|) \cdot \min_{o \in O_{root}}\{|EvO_o|\}$*. Therefore:*

$$AE(l) \geq \lfloor AE(CN) \rfloor = \frac{(\sum_{o \in O_{root}} |EvO_o|) + (\lceil SP(CN) \rceil - |O_{root}|) \cdot \min_{o \in O_{root}}\{|EvO_o|\}}{\lceil SP(CN) \rceil}$$

A lower bound for AE is given by the lower bound of the sum of the events per case, divided by the upper bound on the number of cases. At least one case will exist per object of the root class. Therefore, we consider the sum of the number of events per object. These objects could be involved in more than one case, to a maximum of $\lceil SP(CN) \rceil$ cases. So, we add the minimum number of events they could have, multiplied by the maximum number of additional case identifiers. This is a lower bound given that the number of events added at the upper part for the additional case identifiers is equal or lower than the average. Not adding these extra case identifiers would still result in a lower bound, but an extremely low one since the divisor is usually an overestimation on the number of possible case identifiers.

To define an upper bound for AE, we use an approach similar to the one used to compute an upper bound for LoD. We need to consider the most extreme case, the case in which the maximum number of events per object (for the identifying classes) could be included in the final trace. However, if the case notion has converging classes, the most extreme case is the one in which all the objects of such classes are contained in the case identifier, therefore all the events belonging to the converging classes would be inserted in the trace:

Bound 6 (Upper Bound for the AE of a Case Notion) *Given a valid connected meta-model instance CMI, a case notion* $CN = (C, root, children, CONV, IDC, rsEdge)$, *a maximal set of case identifiers CI, and the corresponding log l, we know that,* $\forall c \in C : \forall o \in O_c : |EvO_o| \leq \max_{o' \in O_c}\{|EvO_{o'}|\}$. *This implies that,* $\forall ci \in CI : |\bigcup_{o \in ci} EvO_o| \leq \sum_{c \in IDC} \max_{o' \in O_c}\{|EvO_{o'}|\} + \sum_{c \in CONV} |EvC_c|$. *Therefore:*

$$AE(l) \leq \lceil AE(CN) \rceil = \frac{|CI| \cdot (\sum_{c \in IDC} \max_{o' \in O_c}\{|EvO_{o'}|\} + \sum_{c \in CONV} |EvC_c|)}{|CI|} =$$
$$\sum_{c \in IDC} \max_{o' \in O_c}\{|EvO_{o'}|\} + \sum_{c \in CONV} |EvC_c|$$

These bounds define the limits for our prediction. For each metric ($SP(l)$, $LoD(l)$ and $AE(l)$), either the lower or upper bound could be a prediction. However, a better heuristic can be designed. We defined equations to predict the values as the weighted average of the corresponding bounds (Equations 5.13, 5.14, and 5.15). Given a valid connected meta-model instance CMI and a case notion CN, our prediction for each metric is given by the following heuristics:

$$\widehat{SP}(CN) = w_{lbsp} \cdot \lfloor SP(CN) \rfloor + w_{ubsp} \cdot \lceil SP(CN) \rceil \quad (5.13)$$

$$\widehat{LoD}(CN) = w_{lblod} \cdot \lfloor LoD(CN) \rfloor + w_{ublod} \cdot \lceil LoD(CN) \rceil \quad (5.14)$$

$$\widehat{AE}(CN) = w_{lbae} \cdot \lfloor AE(CN) \rfloor + w_{ubae} \cdot \lceil AE(CN) \rceil \quad (5.15)$$

From these equations, we see that in order to calculate the heuristics for each metric, we need to collect some features. These features (Table 5.4) can be easily computed once for each class $c \in CL$ in the dataset and be reused for every case notion CN we want to assess.

Finally, in order to score the predicted values of each metric, the scoring function previously used (Equation 5.7) must be individually applied. The input parameters are two: a case notion CN, and a set of case notions CNS to compare to. Equations 5.16, 5.17 and 5.18 provide the scores for the predicted metrics given a case notion CN and a set of case notions CNS.

$$\widehat{s_{sp}}(CN, CNS) = score(\widehat{SP}, CN, CNS, \alpha_{SP}, \beta_{SP}) \quad (5.16)$$

Table 5.4: Features used to compute upper and lower bounds for each log metric.

	Feature	Description
1	$MaxEvO_c = \max\limits_{o \in O_c}\{\lvert EvO_o \rvert\}$	Maximum # of events per object of a class c
2	$MaxAct_c = \max\limits_{o \in O_c}\{\lvert Act_o \rvert\}$	Maximum # of activities per object of a class c
3	$MinEvO_c = \min\limits_{o \in O_c}\{\lvert EvO_o \rvert\}$	Minimum # of events per object of a class c
4	$MinAct_c = \min\limits_{o \in O_c}\{\lvert Act_o \rvert\}$	Minimum # of activities per object of a class c
5	$\lvert EvC_c \rvert$	# of events per class c
6	$\lvert ActC_c \rvert$	# of unique activities per class c
7	$SumEvO_c = \sum\limits_{o \in O_c} \lvert EvO_o \rvert$	Total # of events per object for a class c
8	$SumAct_c = \sum\limits_{o \in O_c} \lvert Act_o \rvert$	Total # of unique activities per object for a class c
9	$\lvert O_c \rvert$	# of objects of a class c

$$\widehat{s_{lod}}(CN, CNS) = score(\widehat{LoD}, CN, CNS, \alpha_{LoD}, \beta_{LoD}) \tag{5.17}$$

$$\widehat{s_{ae}}(CN, CNS) = score(\widehat{AE}, CN, CNS, \alpha_{AE}, \beta_{AE}) \tag{5.18}$$

Next, a global scoring function is defined to combine the three of them. We will call this function the *predicted global scoring function*, $pgsf \in CNS \times \mathscr{P}(CNS) \rightarrow \mathbb{R}$ and it is the weighted average of the scores of each of the three predicted values:

$$pgsf(CN, CNS) = w_{sp} \cdot \widehat{s_{sp}}(CN, CNS) + w_{lod} \cdot \widehat{s_{lod}}(CN, CNS) + w_{ae} \cdot \widehat{s_{ae}}(CN, CNS) \tag{5.19}$$

This function represents our *custom predictor* for log “interestingness”. The accuracy of the predictor will be evaluated in Section 5.6, where it will be compared to alternative techniques.

5.6 Evaluation

So far, we proposed a set of metrics to assess the “interestingness” of an event log once it has been constructed. Also, we provided predictors for these metrics based on (a) the characteristics of the case notion being considered and (b) features of the dataset under study. The aim of this section is twofold:

1. To find out how good our predictors are at estimating the value of each log characteristic.

2. To evaluate the quality of the rankings of case notions, based on their potential “interestingness” according to certain log metrics, using our custom predictor and compare them to existing learning to rank algorithms.

Table 5.5: Details about the SAP dataset used during the evaluation.

Tables	87	**Case Notions**	1.0622×10^4
Objects	7.3400×10^6	**Non empty logs**	5.1800×10^3
Versions	7.3407×10^6	**Total log building time**	13h 57m
Events	2.6106×10^4	**Average log building time**	4.7s
		Features computation time	2m

The evaluation was carried out on an SAP sample dataset (Table 5.5). It contains the data model, objects, object versions, and events of 87 SAP tables. The following steps were executed using the open source software package *eddytools*[6]. First, a set of candidate case notions was generated. Each one of the tables in the data model was taken as the root node of a potential case notion. Next, for each of them, all the possible simple paths following outgoing arcs were computed, yielding a result of 10 622 case notion candidates. For each of the candidates, the corresponding event log was generated and the metrics presented in Section 5.4 were computed. This set of logs and metrics represent the ground truth. Given that we want to predict the metrics in the ground truth set, we need to measure the features that our predictors require. The following section describes these features.

5.6.1 Features for Log Quality Prediction

Section 5.5 presented our predictors for each of the log characteristics. These predictors estimate the values of the *support* (*SP*, Equation 5.13), *level of detail* (*LoD*, Equation 5.14), and *average number of events per trace* (AE, Equation 5.15) of a log, given the corresponding case notion and a set of features. This subsection describes the features used during the evaluation which are (a) the lower and upper bounds of each log property as listed in Section 5.5 and (b) additional features used to improve the accuracy of the regressors we will compare to.

Given a valid connected meta-model instance *CMI* (i.e., a dataset stored in the OpenSLEX format containing events, objects, versions, and a data model) and a specific case notion *CN*, we can measure the features enumerated in Table 5.6. The log associated to such case notion *does not* need to be built in order to compute these features. Actually, many of the features are the result of an aggregation function over a class property. Once the class properties have been computed, the complexity of calculating these case notion metrics is linear with respect to the number of classes involved.

5.6.2 Evaluation of Predictors' Accuracy

In Section 5.5, upper and lower bounds were given for each log property given a case notion (CN). These bounds have been used to estimate the value of such log properties by means of three predictors (one per log property), before the log is actually built.

[6] `https://github.com/edugonza/eddytools`

Table 5.6: Features used to predict log “interestingness”.

	Feature	Description
1	$\lfloor SP(CN) \rfloor$	Lower bound for the support
2	$\lceil SP(CN) \rceil$	Upper bound for the support
3	$\lfloor LoD(CN) \rfloor$	Lower bound for the level of detail
4	$\lceil LoD(CN) \rceil$	Upper bound for the level of detail
5	$\lfloor AE(CN) \rfloor$	Lower bound for average number of events per trace
6	$\lceil AE(CN) \rceil$	Upper bound for average number of events per trace
7	$\lvert C \rvert$	Number of classes in the case notion
8	$\lvert E(CN) \rvert$	Total number of events of all the classes in the case notion
9	$IR(CN)$	Average number of events per object

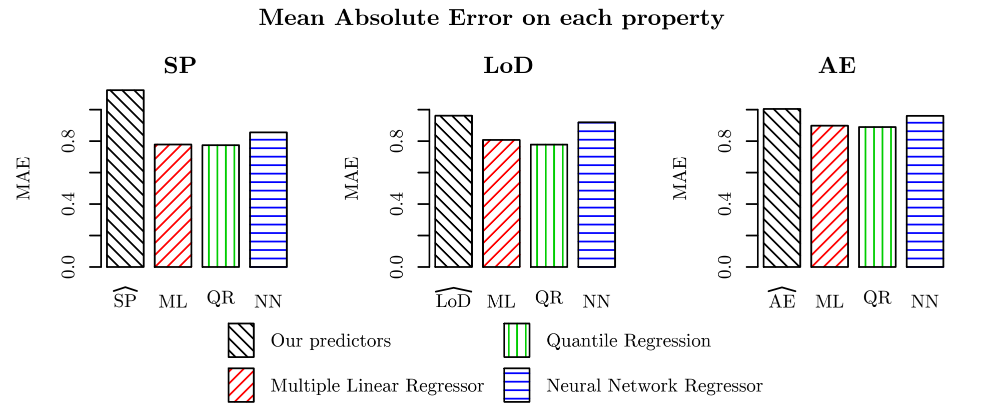


Figure 5.7: Comparison of mean absolute error (MAE) for the predictors on the three normalized log properties.

Now it is time to evaluate the accuracy of these predictors. We compared the predicted value for each log property (*SP*, *LoD*, and *AE*) with the actual values in the ground truth dataset. This was done for the predictors for each log property as defined in Section 5.5 (Equations 5.13, 5.14, and 5.15). The combination of the scores of the three individual predictors (Equations 5.16, 5.17, and 5.18) in a single scoring function of log “interestingness” (Equation 5.19) is what we call our *Custom Predictor (CP)*. Additionally, we compared the accuracy of the individual predictors to three different regressors: (a) Multiple Linear Regressor (MLP), (b) Quantile Regressor (QR) [58], and (c) Neural Network Regressor (NN). Each of them was trained and tested using the features in Table 5.6. A 5-fold cross validation was performed in order to determine the accuracy of the predictors (our predictors, MLP, QR, and NN). To avoid underestimation of the prediction error, empty logs where filtered out of the dataset, using only 5180 case notions from the original 10622.

Figure 5.7 shows the mean absolute error (MAE) measured per normalized property for each predictor. We see that our predictors do not perform really well, presenting an average error of around 1.0 when predicting LoD or AE and around 1.1 when predicting SP. In comparison, within the scope of this experiment and for this specific dataset, the regressors perform better, in particular, the Quantile Regressor with an average error of around 0.8 for SP and LoD, and around 0.9 for AE. This figure, however, could be misleading, given that the MAE is computed on all the predictions, regardless of the existence of outliers. To get a better idea of the influence of extremely bad predictions on the overall performance, we include Figure 5.8, which shows box-plots for each log property per predictor. It is important to notice that a logarithmic scale has been used, in order to plot extreme outliers and still be able to visualize the details of each box.

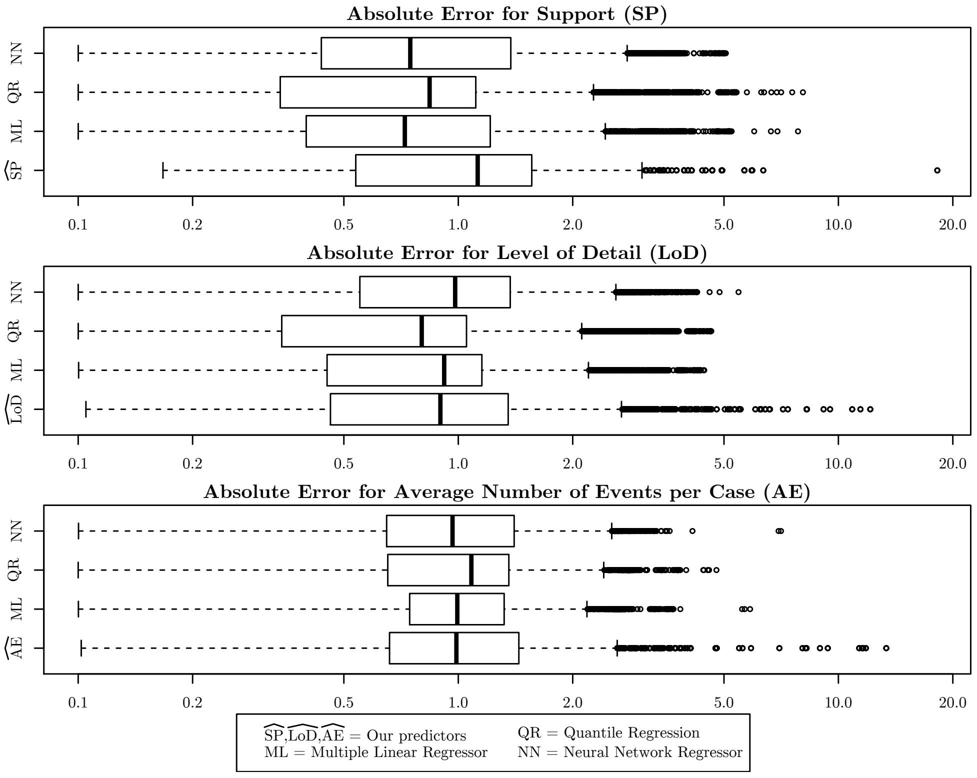


Figure 5.8: Comparison of absolute error for the three normalized log properties per predictor. The scale is logarithmic

We see that our predictors ($\widehat{SP}$, $\widehat{LoD}$, and $\widehat{AE}$) are the worst performing ones, especially when it comes to SP. Also, they are the ones presenting the most extreme outliers for the three log properties. Quantile Regression and Neural Network regressors present the most consistent results, with the least extreme outliers. These results show that there is considerable room for improvement to predict SP, LoD, and AE accurately. This can be achieved, for example, by selecting additional features that have a stronger correlation with the properties we aim to predict. It must be noted that our predictors are unsupervised, i.e., do not need a training set. This represents an advantage with respect to the regressors, since they can generate predictions on the absence of training data. Despite the inaccuracy of our predictors, their usefulness is yet to be determined. The aim of the prediction is to build a ranking of case notions based on their "interestingness" (Equation 5.19). This means that, as long as the relative interestingness is preserved, the ranking can be accurate. The following section will address this issue, using a metric to evaluate the quality of the rankings.

5.6.3 Evaluation of Ranking Quality

Until now we have evaluated the accuracy of our predictors and compared them to other existing regressors. However, the goal of predicting log properties is to assess the "interestingness" of the log *before* it is built. If we are able to predict the interestingness of the logs for a set of case notions, we can rank them from more to less interesting and provide a recommendation to the user. In this section, we evaluate how good the predictors are at ranking case notions according to their interestingness. To do so, we use the metrics on the resulting event logs as the ground truth to elaborate an ideal ranking (Equation 5.12). Then a new ranking is computed using our custom predictor (Equation 5.19) and it is compared to the ideal one. This comparison is done by means of the metric *normalized discounted cumulative gain at p* (*nDCG*$_p$), widely used in the information retrieval field.

$$DCG_p = \sum_{i=1}^{p} \frac{rel\_score_i}{\log_2(i+1)} = rel\_score_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2(i+1)} \tag{5.20}$$

$$IDCG_p = \sum_{i=1}^{|REL\_SCORES|} \frac{rel\_score_i}{\log_2(i+1)} \tag{5.21}$$

$$nDCG_p = \frac{DCG_p}{IDCG_p} \tag{5.22}$$

The *normalized discounted cumulative gain at p* (Equation 5.22) is a metric that assumes the existence of a relevance score for each result, penalizing the rankings in which a relevant result is returned in a lower position. This is done by adding the graded relevance value of each result, which is logarithmically reduced proportional to its position (Equation 5.20). Next, the accumulated score is normalized, dividing it by the ideal score in case of a perfect ranking (Equation 5.21). This means that the ranking $\langle 3,1,2\rangle$ will get a lower score than the ranking $\langle 2,3,1\rangle$ for an ideal ranking $\langle 1,2,3\rangle$ and a relevance per document of $\langle 3,3,1\rangle$.

When it comes to ranking, there is a large variety of *learning to rank* (LTR) algorithms in the information retrieval field [106]. These algorithms are trained on ranked lists of documents and learn the optimal ordering according to a set of features. A 5-fold cross-validation was performed on the unfiltered set of case notions (10 622 candidates) comparing the implementation[7] of 10 learning to rank algorithms (MART,

[7] `https://sourceforge.net/p/lemur/wiki/RankLib/`

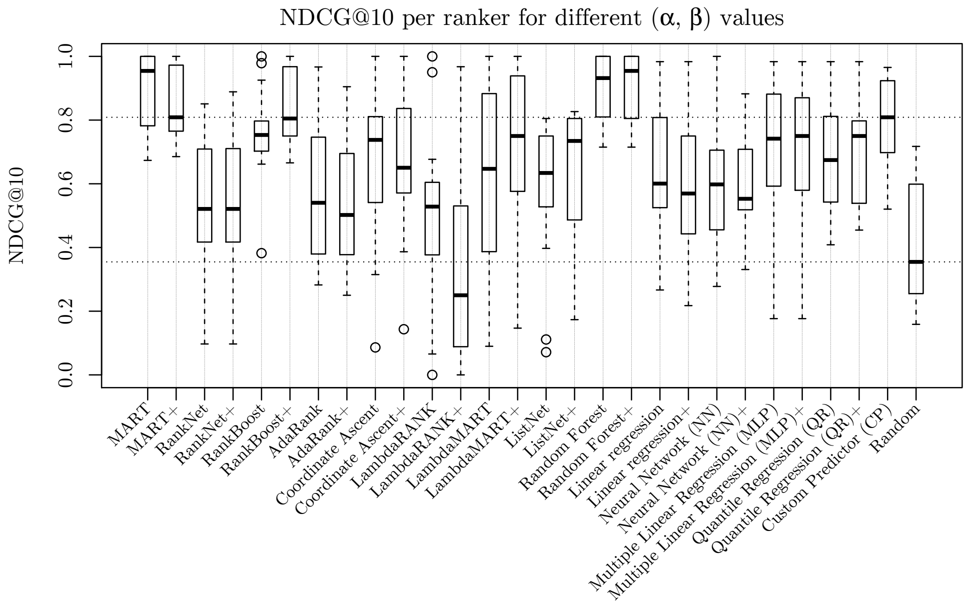


Figure 5.9: NDCG@10 per ranker given different combinations of α and β values.

RankNet, RankBoost, AdaRank, Coordinate Ascent, LambdaRank, LambdaMART, ListNet, Random Forest, and Linear Regression) with the predictors evaluated in Section 5.6.2 (Quantile Regression, Multiple Linear Regression, Neural Network Regressor, and our custom predictor). Two models were trained for each algorithm: one with the 9 input features in Table 5.6 and another one with 4 extra features (the estimated value for SP, LoD, AE, i.e., Equation 5.13, 5.14, and 5.15). The purpose of adding these extra features is to find out how the estimations made by our predictors affect the predictions of the other algorithms.

Figure 5.9 shows the result of the evaluation. The 13 algorithms (10 LTR + 3 regressors) were trained on two different sets of features (9 and 13 input features), 3 different combinations of α and β values for the log quality function ($(\alpha,\beta) \in \{(2,5),(5,2),(2,1)\}$), and with equal weight for the three metrics. That makes a total of 78 models ($(10+3)\times 2\times 3$). The NDCG@10 metric was measured for each model and the results were grouped per algorithm and feature set. That resulted in 27 categories ((10 LTR algorithms × 2 sets of features) + (3 regressors × 2 sets of features) + our custom predictor) with 15 NDCG@10 values each (5 folds × the 3 combinations of α and β values). The models trained with 13 features are represented in the figure with the symbol + at the end of their name. Additionally, the NDCG@10 was calculated for a set of random rankings, in order to set a baseline. In the case of our custom predictor, given that it only takes 6 features (the lower and upper bounds for SP, LoD, and AE) and that it does not need training, only 3 NDCG@10 values were computed, one for each pair of values for the α and β parameters. The horizontal dashed lines drawn in Figure 5.9 represent the median of the NDCG@10 for our custom predictor (upper) and the random ordering (lower). Any algorithm whose median is above the upper line will perform better than our custom predictor at least 50% of the time. Any algorithm whose median is above the lower line will perform better than random at least 50% of the time. Most of the algorithms perform better than random. But only two have the median above the upper line: MART and Random Forest. When trained with 9 input features, both MART and Random Forest show very similar behavior. However, when considering 13 input features, MART's median is lower. In the case of Random Forest, using 13 features is better than using 9 in every aspect.

5.6.4 Discussion

The aim of this evaluation has been twofold. First, to assess the precision of our predictors at estimating the value of each log characteristic. Second, to evaluate the quality of the rankings of case notions, based on their potential "interestingness", using our custom predictor and compare them to LTR algorithms. The results (Figures 5.7 and 5.8) show that our predictors are not very good at predicting log characteristics with precision. Other regressors, like Quantile Regression, have shown better results in this aspect. However, when it comes to ranking quality, the precision in the prediction of the log characteristics is of less importance than the relative differences between predictions for several case notions (i.e., it is not so important to predict accurately the log quality of case notions a and b, as long as we can predict that a will be more interesting than b). In fact, the results obtained from the ranking quality evaluation (Figure 5.9) show that our custom predictor performs better, on average, than other regressors, even though they showed better prediction accuracy.

We conclude that for the purpose of predicting accurately the value of log characteristics and when training data are available, the use of regressors such as QR is the best option. When it comes to ranking candidates, LTR algorithms such as Random Forest and MART provide better results. However, unlike our custom predictor, all these techniques require the existence of training data to build the models. Therefore, in the absence of such data, the proposed custom predictor provides close-to-optimal results when it comes to rankings and indicative values for the prediction of log characteristics.

5.7 Related Work

The field of process mining is dominated by techniques for process discovery, conformance, and enhancement. Yet event correlation and log building are crucial since they provide the data that other process mining techniques require to find insights. In fact, the choices made during the log building phase can drastically influence the results obtained in further phases of a process mining project. Therefore, it is surprising that there are only a few papers on these topics. Works like the one presented in [56] analyze the choices that users often need to make when building event logs from databases. Also, it proposes a set of guidelines to ensure that these choices do not negatively impact the quality of the resulting event log. It is a good attempt at providing structure and a clear methodology to a phase typically subject to experience and domain knowledge of the user. However, it does not aim at enabling automated log building in any form. It has been shown that extracting event logs from ERP systems like SAP is possible [55]. However, the existing techniques are ad-hoc solutions for ERP and SAP architectures and do not provide a general approach for event log building from databases. Another initiative for event log extraction is the *onprom* project [18, 19, 21]. The focus is on event log extraction by means of ontology-based data access (OBDA). OBDA requires to define mappings between the source data source and a final event log structure using ontologies. Then, the *onprom* tools perform an automatic translation from the manually defined mappings to the final event log.

Event log labeling deals with the problem of assigning case identifiers to events from an unlabeled event log. Only a few publications exist that address this challenge. In [30], the authors transform unlabeled event logs into labeled ones using an Expectation-Maximization technique. In [120], a similar approach is presented, which uses sequence partitioning to discover the case identifiers. Both approaches aim at correlating events that match certain workflow patterns. However, they do not handle complex structures such as loops and parallelism. The approach proposed in [7] makes use of a reference process model and heuristic information about the execution time of the different activities within the process in order to deduct case ids on unlabeled logs. Another approach called Infer Case Id (ICI) is proposed in [3] and [15]. The ICI approach assumes that the case id is a hidden attribute inside the event log. The benefit of this approach is that it does not require a reference process model or heuristics. The approach tries to identify the hidden case id attribute by measuring control-flow discovery quality dimensions on many possible candidate event logs. Its goal is to select the ones with a higher score in terms of fitness, precision, generalization, and simplicity. The mentioned approaches for event log labeling are clearly related to the problem we try to solve. However, they ignore the database setting, where event correlations are explicitly defined by means of foreign keys. This means that case identifiers do not need to be discovered. Therefore, the challenge of identifying interesting event logs remains open. Only the ICI approach tackles this issue by measuring control-flow metrics to select the best event log. This is similar to our idea of measuring log "interestingness". However, the ICI approach requires to build all the candidate event logs in order to measure such properties. Our approach is able to reduce the computational cost by predicting interestingness properties before the log is built.

Other authors have already considered the idea of evaluating event log characteristics. The metrics proposed in [50] aim at discovering the structural properties of event logs without actually mining the behavior. These metrics have proven to be of great value in order to develop our automated approach. The approach in [84] focuses on event correlation for business processes in the context of Web services. Additionally, it proposes semi-automatic techniques to generate process views with a certain level of "interestingness". Instead of focusing on what is interesting, it discards uninteresting correlations based on the variability of values on the correlating attributes, or on the ratio of process instances per log. The approach is certainly of value in the area of event correlation. On the other hand, it does not provide a framework for automatic case notion discovery. Also, the approach chosen by the authors to deal with the combinatorial explosion problem is search space pruning, which still requires to compute the event logs, but for a smaller set of candidates.

When it comes to computing rankings, in our case rankings of event logs or case notions, we must consider *learning to rank* (LTR) algorithms from the information retrieval field. These algorithms are able to learn an optimal ordering of documents with respect to certain features. Three main categories can be distinguished among them: pointwise, pairwise, and listwise. Pointwise algorithms try to predict the relevance score of each candidate, one by one. These algorithms are able to give a prediction of the score, but do not consider the position of a document in the ranking. Examples of pointwise algorithms are Random Forest [12], Linear regression [88], the predictors evaluated in Section 5.6.2, and any other algorithm that applies regression in general. Pairwise algorithms take pairs of candidates and predict which candidate ranks higher. In this case, the relative position of documents is taken into account. Examples of pairwise algorithms are MART [32], RankNet [17], RankBoost [31], and LambdaRANK [16]. Listwise algorithms take lists of candidates and learn to optimize the order. A disadvantage of this type of approach is the difficulty to obtain training sets of full ranked lists of candidates. Examples of listwise algorithms are AdaRank [124], Coordinate Ascent [79], LambdaMART [123], and ListNet [32].

As a summary, event correlation, log building, and process view "interestingness" are known topics in the field. Despite the attempts of authors, none of the approaches succeeded at reaching a satisfactory level of automation. Also, none of them proposes a way to recommend process views to the user, neither to rank them by interests.

5.8 Chapter Summary

Applying process mining in environments with complex database schemas and large amounts of data becomes a laborious task, especially when we lack the right domain knowledge to drive our decisions. The work presented in this chapter attempts to alleviate the problem of event log building by automatically computing case notions and by recommending the interesting ones to the user. By means of a new definition of case notion, events are correlated to construct the traces that form an event log. The properties of these event logs are analyzed to assess their "interestingness". Because of the computational cost of building the event logs for a large set of case notion candidates, a set of features was defined based on the characteristics of the case notion and the dataset at hand. Next, a custom predictor estimates the log metrics used to assess the "interestingness". This allows one to rank case notions even before their corresponding event logs are built. Finally, an extensive evaluation of the custom predictor was carried out, comparing it to different regressors and to state of the art learning to rank algorithms. We believe that evaluating the approach in comparison to techniques from the information retrieval field has not been considered before within the process mining discipline.

To conclude, this chapter proposes a framework that covers the log building process from the case notion discovery phase to the final event log computation, providing the tools to assess its “interestingness” based on objective metrics. This assessment can be done on the case notion itself *before* the event log is generated. The result of this assessment is used to provide recommendations to the user.

Beekeeper inspecting a comb from a hive.
"Cours complet d'apiculture", Georges de Layens and Gaston Bonnier, 1897

6 Process Mining Techniques Applied: Data Properties and Opportunities

In the previous chapters, we have proposed several techniques that aim at supporting the user on tasks of event data extraction and event log building. The goal of these techniques is to obtain event logs that can be analyzed with the existing process mining approaches. The purpose of this chapter is to give the reader an idea of what can be achieved with process mining when the right data are available. We present a small sample dataset that serves as an illustration of a rich event log with diverse data properties. The goal is to show the different insights that can be obtained depending on the use that process mining techniques make of the data properties of the event log at hand.

6.1 Introduction

A large part of the time needed to carry out a process mining project is spent on the data preparation phase. It is only after the data are extracted and event logs are built, that the analysis can be performed in order to obtain real insights. The aim of this thesis is to provide techniques that assist on the task of extracting and preparing data for the process mining analysis. This chapter does not introduce new knowledge, but provides some hints on some of the types of process mining analysis that can be performed once the data are ready.

According to the process mining framework introduced in Chapter 10 of [109], there are several process-mining related activities, which are grouped in three categories: cartography (*discover*, *enhance*, and *diagnose*), auditing (*detect*, *check*, *compare*, and *promote*), and navigation (*explore*, *predict*, and *recommend*). The possibility to apply the activities belonging to each of these categories depends on the origin of data to be analyzed. Cartography requires "post mortem" data, which refers to historical data that reflect previous states of the system under study; it does not necessarily represent the current situation. These data is used to discover behavior that has manifested itself in the past, learn from it, enhance it, or diagnose problems. Navigation activities are enabled when "pre mortem" or current data are used, i.e., data that are generated at real-time and reflects the current and most recent situation of the system. "Pre mortem" data are necessary in order to provide valuable predictions and recommendations at the time that they are needed. Finally, a combination of the two types of data are required to perform auditing activities, e.g., in order to compare current behavior with past trends.

The focus of this thesis is on "post mortem" data, i.e., data that have been generated and stored in databases, event stores, or other forms of data storage for off-line analysis. Therefore, in this chapter we will explore some of the techniques that can be applied to historical data and discuss how to leverage different data properties of event logs.

This chapter is structured as follows. Section 6.2 proposes a non-exhaustive list of process mining techniques and their data requirements. Section 6.3 introduces a sample dataset with different data properties. Section 6.4 extends the description of the proposed list of process mining techniques. Finally, Section 6.5 concludes this chapter.

6.2 Data Properties and Process Mining Techniques

Event data can come in many different forms. During a process mining project, extracting and generating event logs has a big cost in terms of time and effort. This motivates the work presented in this thesis, with the goal of bringing a certain level of standardization and automation to the process of data extraction and preparation. The techniques proposed in the previous chapters aim at providing, as an ultimate result, a set of event logs that capture different perspectives on the system under study. However, event logs are not the desired end result when carrying a process mining project, but an element that needs to be analyzed to obtain insights. In this section, we propose a non-exhaustive list of process mining techniques, and their data dependencies in order to be applied.

We limit the scope of this review to a few specific approaches. These cover the following types of process mining:

- Model discovery: the goal is to obtain (process) models that represent some perspective on the event data.

- Trace clustering: event data is analyzed to obtain clusters of traces based on different aspects, e.g., control-flow, data attributes.

Table 6.1: Attributes required (R), optional (O), or ignored (-) for the input data of different process mining techniques.

Name	Input	Output	Case Notion	Timestamp	Event order	Activity name	Life-cycle	Resource	Data attributes	Hierarchy
Model Discovery										
Inductive Miner [64]	Event log	Process tree	R	O	R	R	O	O	-	-
Data-aware Heuristic Miner [76]	Event log	Data Petri net	R	O	R	R	O	O	O	-
Statechart Workbench [61]	Event log	Petri net	R	O	R	R	O	O	O	O
Social Network Miner [112]	Event log	Social network	R	O	R	O	-	R	-	-
...										
Trace Clustering										
Log to Model Explorer [71]	Event log	Trace clusters	R	O	R	R	-	-	-	-
Cluster Traces using Markov Clustering [53]	Event log	Trace clusters	R	O	O	O	O	O	O	-
...										
Conformance Analysis										
Replay a Log on Petri Net for Conformance Analysis [111]	Event log & Petri net	Conformance Analysis	R	O	R	R	O	-	-	-
Conformance Checking for Data Petri Nets [75]	Event log & Data Petri net	Conformance Analysis	R	O	R	R	O	O	O	-
...										
Performance Analysis										
Replay a Log on Petri Net for Performance/Conformance Analysis [111]	Event log & Process model	Performance Analysis	R	R	R	R	O	-	-	-
Context-Aware Process Performance Analysis [52]	Event log	Performance Analysis	R	R	R	R	O	O	O	-
...										

- Conformance analysis: this is a way to enhance existing process models providing diagnostics on how faithfully a model represents an event log.
- Performance analysis: process models can be extended with performance statistics computed on the observed behavior.

On the basis of a review of some of the existing process mining approaches, we have identified the following attributes that can be present in the event data:

- Case Notion: a general assumption in many process mining techniques is the availability of event logs in which events are grouped into cases or traces. Each of these traces represents a process instance. A case notion defines how events should be grouped into traces, e.g., by means of a case identifier.

- Timestamp: a timestamp indicates the moment in which an event occurred. Time resolution can vary from days to milliseconds. Also, timestamps are commonly used to sort events in order of occurrence.
- Event order: when a timestamp is not available or when time resolution is not fine-grained enough to sort events without ambiguity, another way to find an order is required, e.g., partial orders. In some environments, events are recorded in the order of occurrence, even if the exact timestamp is not known.
- Activity name: an event reflects the occurrence of an activity in a certain moment in time, i.e., the happening of an activity instance. A common way to relate events to the activity they executed is by means of an activity name.
- Life-cycle: an event can indicate the execution of an atomic action or a step within the life-cycle of a non-atomic one. In the latter case, the trail of an activity instance can span through several events reflecting different phases of the life-cycle of the corresponding activity. The most common phases are *start* and *complete*, indicating the beginning and end of an activity instance respectively.
- Resource: activity instances can be carried out by certain resources, e.g., employees, machines, equipment. The reference to the specific resource involved in the execution can be reflected in the *resource* field of an event.
- Data attributes: additional event attributes can contain any extra information about the event itself or about the context in which it occurred. In a database environment, events can be directly related to database objects, e.g., purchase orders, bookings, customer profiles. Data about these related objects can be included in the event for further analysis.
- Hierarchy: not all events reflect activity instances executed at the same level. Different hierarchies can exist representing activities that invoke other activities, e.g., as part of a subprocess. In order to discover these different execution levels, it is necessary to encode this information somehow in the event data.

It is rare to face an event log that provides all of these attributes. But it is possible to find, at least, subsets of them. For instance, the most basic event logs provide, at least, a set of ordered event occurrences linked to activity names, as well as an implicit case notion. Sometimes, it is possible to find resource attributes, e.g., who performed the activity. In certain situations, a life-cycle model is known and can be incorporated into the data when extracted, e.g., *start* and *complete* steps within the execution of an activity instance. Other data attributes are easily obtained when extracting event logs from databases. In some exceptional cases, a notion of hierarchy is present in the data, and can be exploited to discover threads of execution within the process invoking other subprocesses. Depending on the subset of data aspects that our event

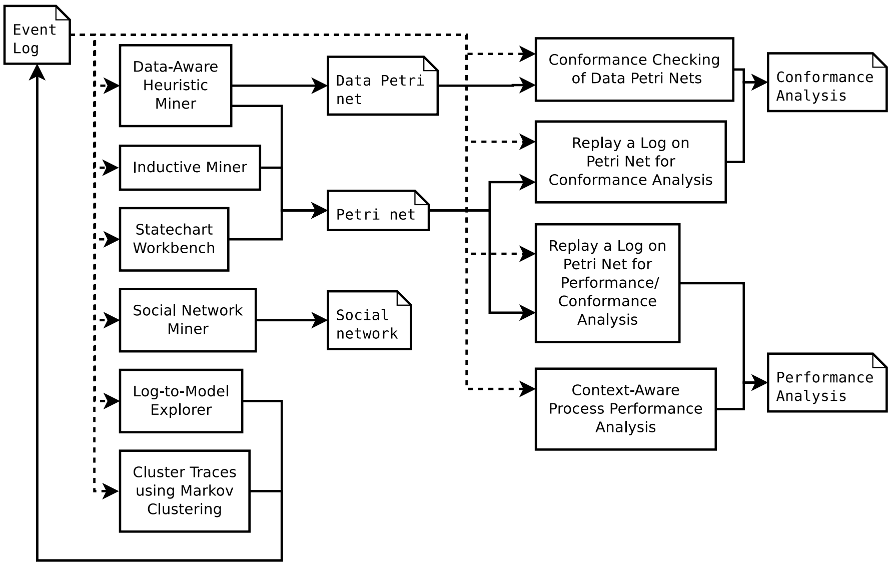

Event Log
Data-Aware Heuristic Miner
Inductive Miner
Statechart Workbench
Social Network Miner
Log-to-Model Explorer
Cluster Traces using Markov Clustering
Data Petri net
Petri net
Social network
Conformance Checking of Data Petri Nets
Replay a Log on Petri Net for Conformance Analysis
Replay a Log on Petri Net for Performance/ Conformance Analysis
Context-Aware Process Performance Analysis
Conformance Analysis
Performance Analysis

Figure 6.1: Landscape of the process mining techniques proposed in Table 6.1 with the corresponding input/output dependencies.

log exhibits, different techniques can be applied. Some techniques require additional inputs apart or instead of event logs, e.g., process models. In that case, different techniques can be applied (Figure 6.1) in order to use the output of one as the input of another. Table 6.1 presents a short list of process mining techniques with respect to the attributes that the input needs to fulfill in order to be applied. Attributes marked with an “R” are required, i.e., if the data do not have that attribute the technique cannot be applied. Attributes marked with an “O” are optional, i.e., the technique can use these attributes to provide more informative results, but they are not required in order to run. Attributes marked with the symbol “-” are not relevant or ignored by the technique.

We consider it as important to remain practical when proposing analysis techniques. For this reason, all the techniques used in this chapter are supported by plug-ins in the ProM[1] platform.

6.3 A Sample Event Log

A way to show the results that can be expected from the approaches proposed in this chapter is to run the algorithms on a dataset. For this purpose, we built a sample event log containing the most representative data characteristics as discussed in section 6.2. Table 6.2 shows this sample event log based on a fictitious ticket selling process. Each row represents an event, in total 32. The first column on the left serves as a unique

[1]http://promtools.org

identifier for the event, and it shows the global *event order*, too. The second column (*Case*) indicates the case identifier of the case/trace to which the event belongs, i.e., the *case notion*, with 6 cases in total. The third column shows the *activity name* associated to the event. In this event log, there are 4 different activities: "Insert Customer", "Update Customer", "Make Booking", and "Update Ticket". The fourth column provides an exact *timestamp* of the moment in which the event occurred. The next two columns reflect the values for *resource* and *life-cycle* attributes. As can be seen, there are two types of resources: the system ("System") and customers (e.g., "Cust.101"). The life-cycle attribute either indicates the "start" or the "complete" state of the activity instance. Finally, the last two columns are *data attributes* specific for this dataset, indicating if there was an issue with the payment (*pay_issue*) and what the price was of each purchased ticket (*price*). It can be noticed that the notion of *hierarchy* is not explicitly indicated in the events of this dataset. However, we will show how other fields such as *life-cycle* can be used with the purpose of obtaining hierarchy relations between events.

This is a simple example, but it serves as an illustration of a rich event log in terms of data properties and attributes. The goal is to show how different insights can be obtained depending on the use that different analysis techniques make of the available data properties. The following sections show the output obtained when applying the techniques proposed in Section 6.2 to this sample event log.

6.4 Process Mining Techniques Applied

In this section, we show the outputs obtained after applying the process mining techniques listed in Table 6.1 to the sample event log in Table 6.2. In each subsection we explain, at a high level, the functioning of the corresponding technique and we show an example of the insights that can be obtained.

6.4.1 Model Discovery

The purpose of discovery techniques is to obtain a (process) model that describes the behavior captured in the event log being analyzed. They are usually called "miners", and many variants exist. In this section, we look into four different miners: the Inductive miner, the Data-aware heuristic miner, the Statechart Workbench, and the Social network miner. These miners differ from each other in the data properties they take into account during the discovery process and the kind of behavioral structures they are able to detect.

Inductive Miner

One of the most recent and successful discovery techniques is Inductive Miner (IM) [64]. A big part of its popularity is due to its ability to handle large event logs, a good user interface, and the guaranteed soundness of the discovered process models. This feature of the discovered models (soundness) is due to the fact that, in order to discover process models, first IM splits the event log into smaller pieces.

Table 6.2: Example of an event log obtained from the database of a ticket selling system.

#	Case	Activity Name	Time stamp	Resource	Life-cycle	pay_issue	price
1	1	Insert Customer	2014-11-27 15:55:35	System	complete		
2	1	Make Booking	2014-11-27 15:58:23	Cust.101	start	FALSE	
3	2	Insert Customer	2014-11-27 15:59:10	System	complete		
4	1	Update Ticket	2014-11-27 16:03:25	System	complete		30.5
5	1	Make Booking	2014-11-27 16:03:26	Cust.101	complete		
6	3	Make Booking	2014-11-27 17:12:50	Cust.103	start	FALSE	
7	3	Update Ticket	2014-11-2 717:15:22	System	complete		27.5
8	2	Make Booking	2014-11-27 17:23:45	Cust.102	start	FALSE	
9	2	Update Ticket	2014-11-27 17:55:15	System	complete		40.0
10	3	Update Ticket	2014-11-27 18:07:50	System	complete		18.0
11	3	Make Booking	2014-11-27 18:07:51	Cust.103	complete		
12	2	Update Ticket	2014-11-27 18:13:32	System	complete		37.5
13	2	Make Booking	2014-11-27 18:13:33	Cust.102	complete		
14	4	Insert Customer	2014-11-28 10:12:46	System	complete		
15	4	Make Booking	2014-11-28 10:20:35	Cust.104	start	TRUE	
16	5	Insert Customer	2014-11-28 11:38:13	System	complete		
17	5	Make Booking	2014-11-28 12:01:15	Cust.105	start	TRUE	
18	5	Update Customer	2014-11-28 12:02:05	Cust.105	complete		
19	4	Update Customer	2014-11-29 22:40:21	Cust.104	complete		
20	4	Update Ticket	2014-11-29 22:45:12	System	complete		15.5
21	4	Update Ticket	2014-11-29 23:01:51	System	complete		17.0
22	5	Update Ticket	2014-11-29 23:05:10	System	complete		25.5
23	4	Update Ticket	2014-11-29 23:15:28	System	complete		23.0
24	4	Make Booking	2014-11-29 23:15:29	Cust.104	complete		
25	6	Update Customer	2014-11-30 13:34:26	Cust.106	complete		
26	6	Make Booking	2014-11-30 13:38:14	Cust.106	start	FALSE	
27	5	Update Ticket	2014-11-30 14:45:08	System	complete		28.5
28	5	Make Booking	2014-11-30 14:45:09	Cust.105	complete		
29	6	Update Ticket	2014-11-30 14:56:42	System	complete		24.5
30	6	Update Ticket	2014-11-30 15:10:45	System	complete		22.5
31	6	Update Ticket	2014-11-30 15:25:36	System	complete		23.0
32	6	Make Booking	2014-11-30 15:25:37	Cust.106	complete		

Once it has been recursively split to a certain level, IM builds a specific type of block-structured process model called process trees. By definition, this type of model guarantees soundness. However, one of the drawbacks of this algorithm is that it tends to favor parallel constructs to model behavior. In extreme cases, this can result in the discovery of flower models, i.e., models that have a perfect fitness but allow for too much behavior, in this way scoring very low in terms of precision.

Figure 6.2 shows an example of a discovered process tree using Inductive Miner. We see that the activity *Insert Customer* has been placed at the beginning of the process. However, the model allows skipping the execution following the skip-arc placed on top. Next, we find an *and* operator (+) that will continue the execution in parallel on the upper and lower branches. The top branch only offers one option, to execute the activity *Make Booking*, which could be followed by the end of this branch or by the execution of *Update Ticket*. In case *Update Ticket* gets executed,

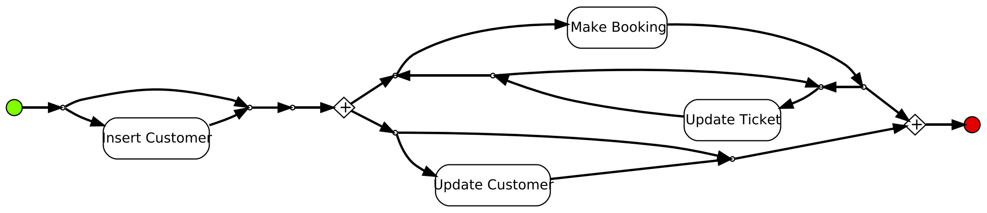


Figure 6.2: Process tree discovered by Inductive Miner.

we have the option to either repeat in a loop thanks to the arc going back from left to right above the activity, or to execute *Make Booking* again. The fact that *Make Booking* can be executed twice is due to the fact that *Make Booking* appears in our event log twice per trace with different life-cycle values (*start* and *complete*). A way to differentiate these two events is to select a different event classifier that combines the activity name with the life-cycle. Therefore, we would have two activities instead of one. In the meantime, the lower branch is executed in parallel (according to IM). This branch allows to either execute or skip the *Update Customer* activity. Once both branches have ended, there is an *and-join* operator that leads to the end of the process.

The process tree in Figure 6.2 is just one of the many process models that can be mined with IM on this event log by tuning the parameters offered by the miner. It is possible to filter by activity and path frequency in order to simplify the model, filter out noise, or show the most frequent behavior. Process trees discovered with the Inductive Miner can be converted to Petri nets in order to be used by other plug-ins.

Data-Aware Heuristic Miner

Another recently developed plug-in is the Data-aware heuristic miner [76]. This miner is based on the principles of the heuristic miner [122], but evolved in order to handle the data perspective of processes. This algorithm is able to discover models that reflect the most common behavior, being robust to noise. Also, it is able to detect long-distance relationships between activities that do not happen in a sequence. This miner allows us to discover directly-follows graphs, causal nets, data causal nets (Figure 6.3), Petri nets, and data Petri nets. When the *data* variants of causal and Petri nets are discovered, data attributes present in the event log are used in order to uncover data guards that model the observed behavior.

In our example dataset, we deliberately included a data attribute that relates to the behavior of the underlying process (*payment_issue*). When we use the Data-aware heuristic miner to discover a data causal net, we obtain the model in Figure 6.3. We see that the process resembles the one discovered by the Inductive Miner. It is important to note that the discovered model is expressed in the notation of causal nets. The dots placed on top of the arcs connecting activities are used to indicate whether activities happen in parallel or in exclusion. For example, at the beginning of the process, there is the option to execute the activity *Insert Customer+complete* (there is a dot

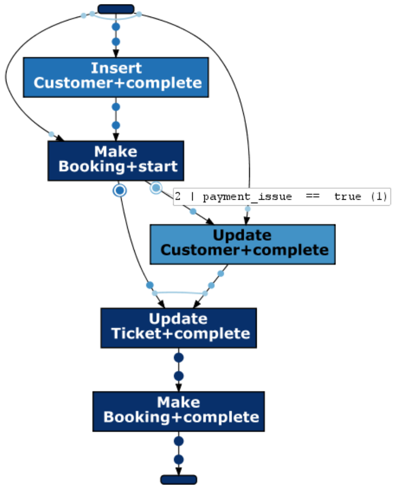


Figure 6.3: Data causal net discovered with the Data-aware heuristic miner showing one of the discovered data guards.

at the start of the initial edge) or to execute the activities *Make Booking+start* and *Update Customer+complete* in parallel (there is a small arc connecting the dots on top of the edges leading to each activity). This notation makes it possible to represent complex control-flow structures in a very compact way. The difference between causal nets and data causal nets is that the latter are able to represent data guards. These guards are represented by a dot with a double border on top of the output arcs of certain activities. In the example in Figure 6.3, two data guards have been discovered that model the choice after the execution of *Make Booking+start*. If the attribute *payment_issue* has the value *true*, the path on the right is enabled and the activity *Update Customer+complete* will be executed. If the value is *false*, the path on the left will be enabled and the activity *Update Ticket+complete* will be executed.

Some of the techniques listed in Table 6.1 require a process model next to an event log as input. In those cases, a data Petri net equivalent to the data causal net shown in Figure 6.3 will be used.

Statechart Workbench

In real-life scenarios, we commonly find business processes that include activities nested within other higher level activities. An example is the case of subprocesses. The Statechart workbench is an approach based on Inductive Miner that extends the definition of process trees to include more advanced modeling constructs common in software systems such as subprocesses and nested calls, and cancellation regions. In order to discover subprocesses and nested calls, the miner expects to find a notion of hierarchy within the data attributes of the event log (several conversions and heuristics are provided). This hierarchy can be decoded in an ad-hoc manner to adapt to the system under study or it can be inferred from other attributes and control-flow

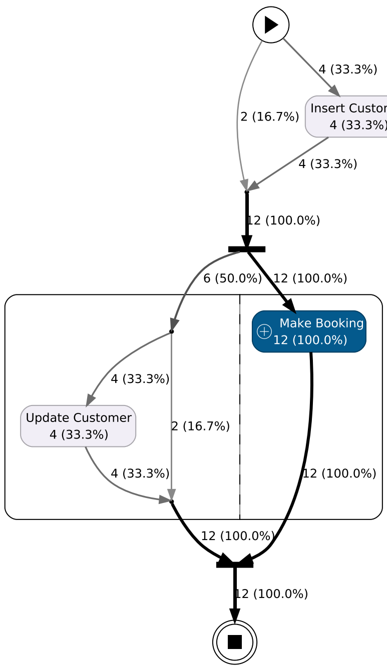


Figure 6.4: Process model discovered by the Statechart workbench with the activity *Make Booking* showing a nested subprocess (still collapsed).

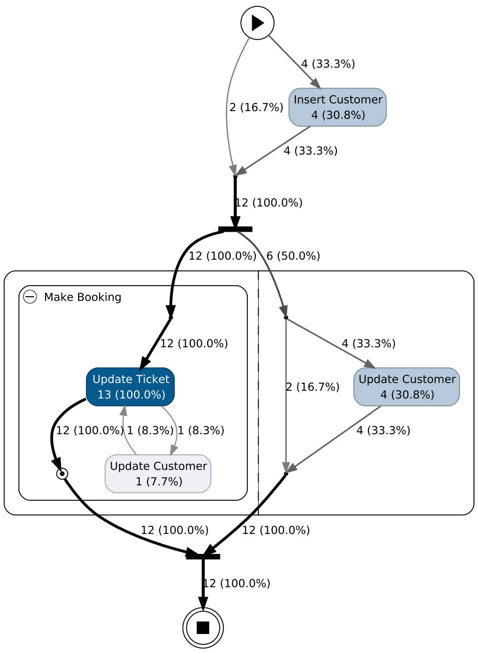
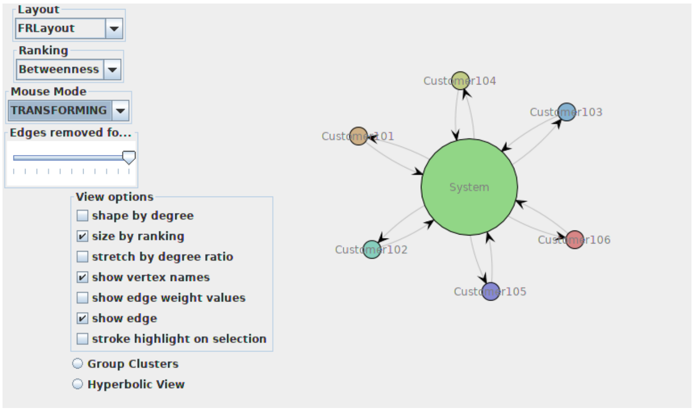


Figure 6.5: Process model discovered by the Statechart workbench with the activity *Make Booking* showing two nested activities, one of them duplicated.

structures. In this example, we are interested in discovering hierarchy from our sample event log making use of the life-cycle attribute of the events.

The Statechart workbench offers the possibility to use the life-cycle to discover nested activities that occur between the *start* and *complete* stages of a higher-level activity. Figure 6.4 shows the discovered model from the sample event log. We see that it detects a parallel construct, as delimited by the square with rounded corners, with two parallel tracks: one with the *Update Customer* activity, and another one with the *Make Booking* activity. In the latter, we observe that the activity has a (+) sign next to its name. Figure 6.5 shows that expanding the activity block shows the nested behavior: a loop between *Update Ticket* and *Update Customer*. The miner has detected that the activity *Update Customer* can happen in two different contexts: in parallel to the *Make Booking* activity, or as a nested call within the *Make Booking* activity. This distinction makes it possible to simplify the represented model eliminating arcs between activities, obtaining simpler control-flow structures and providing a better representation of the real behavior of the underlying system.


Layout: FRLayout
Ranking: Betweenness
Mouse Mode: TRANSFORMING
Edges removed fo...

View options
- [ ] shape by degree
- [x] size by ranking
- [ ] stretch by degree ratio
- [x] show vertex names
- [ ] show edge weight values
- [x] show edge
- [ ] stroke highlight on selection

○ Group Clusters
○ Hyperbolic View

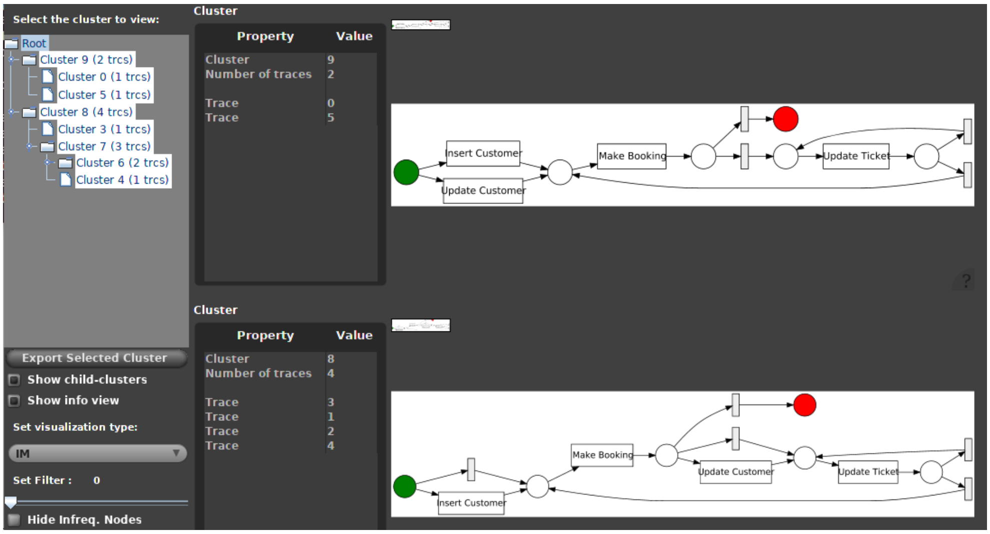
Figure 6.6: Social network discovered by the Handover-of-Work Social Network miner. The nodes represent different resources involved in the execution of the process.

Social Network Miner

The previous miners presented in this section are focused on discovering the control-flow between activities. This is important to understand how the different steps of the process are executed, but we miss an important perspective: who or what is executing those activities? The *resource* attribute of an event log provides the necessary information to analyze interactions between resources during the execution of a business process. The Social network miner provides a set of algorithms that focus on this task: analyzing interactions between resources with the purpose of discovering behavioral patterns such as handover-of-work, reassignments of tasks, similar-task relations, subcontracting behavior, and work-together patterns.

Due to the simplicity of our sample event log, we will only show the result of discovering a handover-of-work social network. Figure 6.6 shows the discovered network represented by a graph. Each node represents a resource (*System*, *Customer.101*, *Customer.102*, etc.) and the edges represent the handover-of-work. The presence of an edge between two resources means that one of the resources performed an activity interleaved between the execution of other activities by the other resource. A common scenario is the handover of work from department A to department B and, after the completion of a certain task, department B returns the work back to department A. In our example, we see that the resource *System* plays a central role within the process, being involved in all the interactions with the other resources. This might indicate that some activities within the process can only be executed by the resource *System*. The analysis of social networks behind the execution of business processes allows us to identify social interactions, roles, risks, and inefficiencies in the way an organization operates and utilizes its resources.

Here, we conclude the overview of process discovery techniques. The following sections present the rest of the techniques on clustering, conformance, and performance analysis, showing the results obtained on the proposed sample event log.



Select the cluster to view:

- Root
  - Cluster 9 (2 trcs)
    - Cluster 0 (1 trcs)
    - Cluster 5 (1 trcs)
  - Cluster 8 (4 trcs)
    - Cluster 3 (1 trcs)
    - Cluster 7 (3 trcs)
      - Cluster 6 (2 trcs)
      - Cluster 4 (1 trcs)

Show child-clusters
Show info view
Set visualization type: IM
Set Filter : 0
Hide Infreq. Nodes

Cluster

Property	Value
Cluster	9
Number of traces	2
Trace	0
Trace	5

Cluster

Property	Value
Cluster	8
Number of traces	4
Trace	3
Trace	1
Trace	2
Trace	4

Figure 6.7: Clusters discovered by the Log to Model Explorer tool on the sample event log.

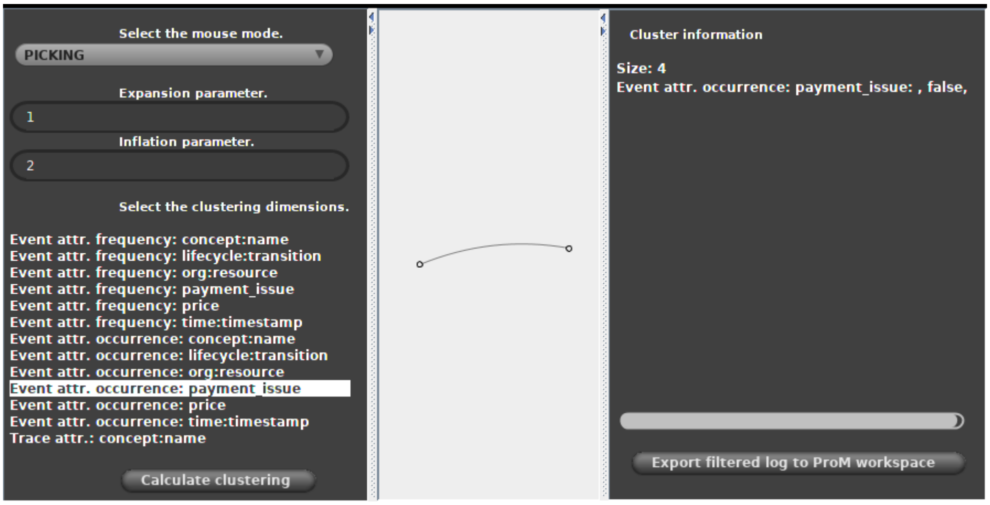
6.4.2 Trace Clustering

In this subsection, we discuss some clustering techniques applied to event data. To be more specific, the clustering techniques we cover are focused on grouping traces, or process instances, with respect to a certain similarity measure. The purpose is to show that, when the event logs contain the right data, it is possible to identify clusters of traces that share a common characteristic such as similar behavior, control-flow structures, data values, etc. We will see how different possibilities exist when extra data properties are available in order to be analyzed.

L2Me: Log to Model Explorer

This trace clustering technique has an explorative nature, since it allows to hierarchically inspect trace clusters and automatically obtain process models from them in order to appreciate differences. Traces are grouped in clusters based on similar behavior, i.e., similar control-flow patterns are observed. This allows us to obtain simpler process models that describe different trace variants. Without these clusters, all these variants would be mixed in the same event log and would pollute the resulting process model.

An example of clustering can be observed in Figure 6.7. On the left we see that the traces within the event log have been clustered hierarchically. A tree view shows the root node as a cluster of all the traces in the log. When it is selected, the right side shows the detailed view of the first level of child clusters. In this case, two clusters have been discovered: *Cluster 9* with two traces (top) and *Cluster 8* with four traces (bottom). The main difference between the two has to do with the moment in which the *Update Customer* activity is executed. In the traces belonging to the cluster on


Select the mouse mode.
PICKING

Expansion parameter.
1

Inflation parameter.
2

Select the clustering dimensions.

Event attr. frequency: concept:name
Event attr. frequency: lifecycle:transition
Event attr. frequency: org:resource
Event attr. frequency: payment_issue
Event attr. frequency: price
Event attr. frequency: time:timestamp
Event attr. occurrence: concept:name
Event attr. occurrence: lifecycle:transition
Event attr. occurrence: org:resource
Event attr. occurrence: payment_issue
Event attr. occurrence: price
Event attr. occurrence: time:timestamp
Trace attr.: concept:name

Cluster information

Size: 4
Event attr. occurrence: payment_issue: , false,

Figure 6.8: Details of the first cluster discovered by Markov clustering based on the attribute *payment_issue*.

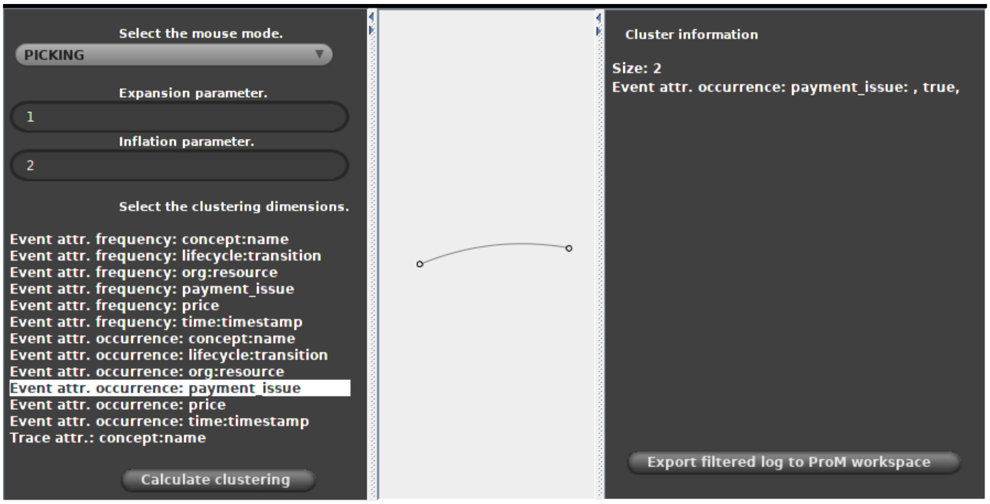
top, the customer update is executed before a booking is made, as part of an exclusive choice between the insertion or update of a customer. In the cluster at the bottom, the update of a customer is performed after a booking has started and before any ticket is updated. It is possible to continue expanding the tree in order to explore the rest of clusters until, finally, reaching the leaves, which are represented by single-trace clusters. *Cluster 8* is divided into another two subclusters: *Cluster 3*, which is a leaf cluster, and *Cluster 7*, which contains three traces.

This way of hierarchically exploring the traces of an event log allows us to make a decision on the desired level of granularity and the level of specialization of the resulting sublog. The resulting clusters can be exported as event logs and used as input for other analysis techniques.

Cluster Traces using Markov Clustering

We have seen that the traces in an event log can be clustered according to their control flow behavior. However, there are other criteria that can be considered to group traces together. The technique proposed in [53] introduces a method to discover groups of traces based on Markov clustering, using any available event or trace attribute according to the frequency or occurrence of their values.

Figure 6.8 shows the output obtained after applying the technique on the sample event log. The selected clustering criterion is the occurrence of the event attribute called *payment_issue*. The result, as can be seen at the center of the figure, is a graph of two clusters. When clicking on one of the clusters we obtain a detailed view (on the right side) showing the size of the cluster (number of traces), and the value of the attribute used during the clustering. In this case, one of the clusters groups traces


Select the mouse mode.
PICKING

Expansion parameter.
1

Inflation parameter.
2

Select the clustering dimensions.

Event attr. frequency: concept:name
Event attr. frequency: lifecycle:transition
Event attr. frequency: org:resource
Event attr. frequency: payment_issue
Event attr. frequency: price
Event attr. frequency: time:timestamp
Event attr. occurrence: concept:name
Event attr. occurrence: lifecycle:transition
Event attr. occurrence: org:resource
Event attr. occurrence: payment_issue
Event attr. occurrence: price
Event attr. occurrence: time:timestamp
Trace attr.: concept:name

Cluster information

Size: 2
Event attr. occurrence: payment_issue: , true,

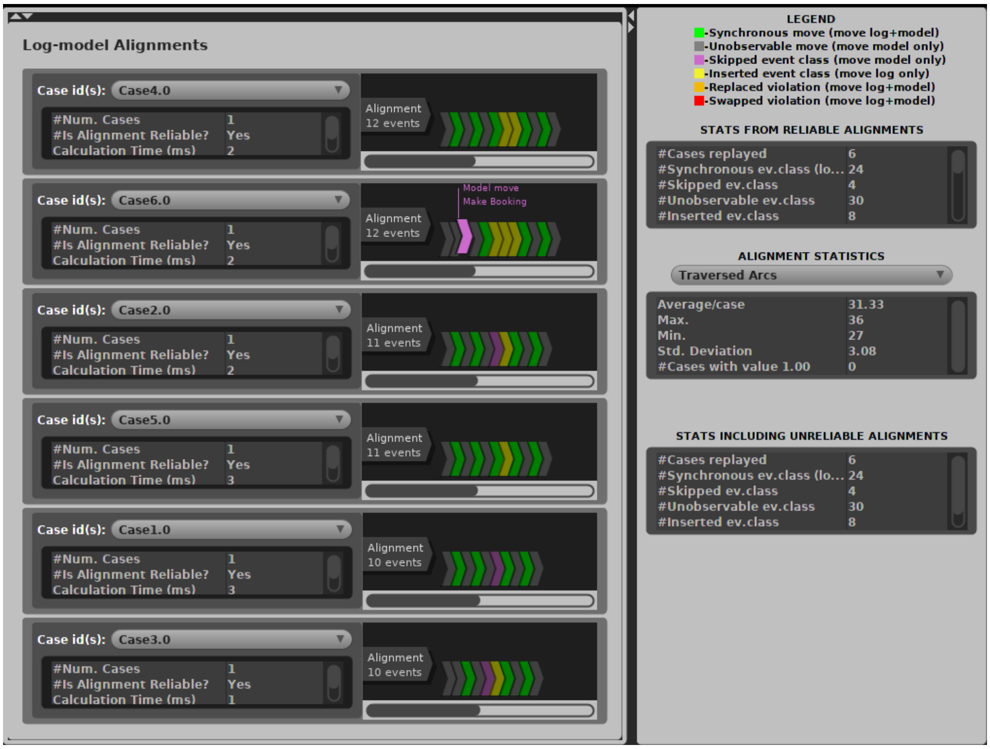
Figure 6.9: Details of the second cluster discovered by Markov clustering based on the attribute *payment_issue*.

that contain events with the value *false* for the attribute *payment_issue*. The other cluster (Figure 6.9) has a size of two and groups the rest of the traces with events for which the value of *payment_issue* is *true*.

The result matches the one obtained with the clustering technique used by L2Me. The main difference is the criteria followed to obtain these clusters. The method used by L2Me only takes into account the differences in control-flow and behavior. Markov clustering, on the other hand, takes into account differences in values for certain properties, in this case the attribute *payment_issue*. In fact, we see that when the value of *payment_issue* is *true*, a customer update is performed right after the execution of *Make Booking*. In case the value of *payment_issue* is *false*, the customer update is not required. This shows how different techniques can arrive at the same output by different approaches. It also shows how combining the results helps to obtain better insights.

6.4.3 Conformance Analysis

Until now, we have showcased techniques that only required an event log as an input. However, there is a group of techniques that leverages on the existence of a process model to obtain more insights about the behavior contained within the event logs. Conformance checking techniques are able to analyze event logs showing differences with respect to a designed or discovered *de facto* model. The goal is to pinpoint the main differences between what is expected (the model) and what is observed (the log). We will show two approaches that can perform conformance checking from two perspectives: control-flow only or data-aware conformance checking.



Log-model Alignments

Case id(s): Case4.0
#Num. Cases 1
#Is Alignment Reliable? Yes
Calculation Time (ms) 2
Alignment 12 events

Case id(s): Case6.0
#Num. Cases 1
#Is Alignment Reliable? Yes
Calculation Time (ms) 2
Alignment 12 events
Model move
Make Booking

Case id(s): Case2.0
#Num. Cases 1
#Is Alignment Reliable? Yes
Calculation Time (ms) 2
Alignment 11 events

Case id(s): Case5.0
#Num. Cases 1
#Is Alignment Reliable? Yes
Calculation Time (ms) 3
Alignment 11 events

Case id(s): Case1.0
#Num. Cases 1
#Is Alignment Reliable? Yes
Calculation Time (ms) 3
Alignment 10 events

Case id(s): Case3.0
#Num. Cases 1
#Is Alignment Reliable? Yes
Calculation Time (ms) 1
Alignment 10 events

LEGEND
🟩-Synchronous move (move log+model)
◼-Unobservable move (move model only)
🟪-Skipped event class (move model only)
🟨-Inserted event class (move log only)
🟧-Replaced violation (move log+model)
🟥-Swapped violation (move log+model)

STATS FROM RELIABLE ALIGNMENTS

#Cases replayed	6
#Synchronous ev.class (lo...	24
#Skipped ev.class	4
#Unobservable ev.class	30
#Inserted ev.class	8

ALIGNMENT STATISTICS

Traversed Arcs

Average/case	31.33
Max.	36
Min.	27
Std. Deviation	3.08
#Cases with value 1.00	0

STATS INCLUDING UNRELIABLE ALIGNMENTS

#Cases replayed	6
#Synchronous ev.class (lo...	24
#Skipped ev.class	4
#Unobservable ev.class	30
#Inserted ev.class	8

Figure 6.10: Detailed view of alignments per trace on the sample event log with respect to a discovered Petri net.

Replay a Log on Petri Net for Conformance Analysis

The conformance analysis tool called *Replay a Log on Petri Net for Conformance Analysis* is based on the alignment technique presented in [111]. An alignment is able to show that an event trace can be replayed on a process model. The result is the set of steps that need to be followed to do so. This is not trivial, since deviations might exist in the log with respect to the model, and non-synchronous moves, called “log moves” and “model moves”, need to be considered. The result of replaying an event log on a model using this method is a set of alignments that can be visualized per trace. Figure 6.10 shows a detailed view per trace, with a color-coded representation of the synchronous moves (green), model moves (pink), log moves (yellow), and unobservable moves (gray).

Another way to visualize the result of a conformance analysis is to project the alignments onto the process model, enhancing it with augmented information about synchronous, log, and model moves. Figure 6.11 shows a view of the process model with conformance information plotted on top. The color code is the same as in Figure 6.10. We see that, in this case, the transitions in the Petri net can be involved either in always synchronous behavior (*Insert Customer*), only model moves (all the

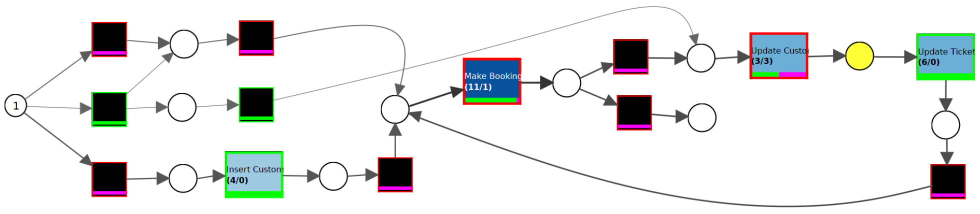
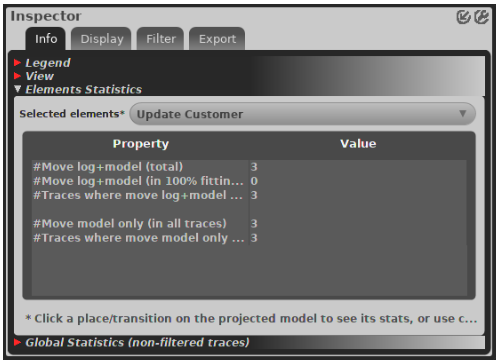


Figure 6.11: Petri net showing enhanced conformance information based on the sample event log.



Selected elements* Update Customer

Property	Value
#Move log+model (total)	3
#Move log+model (in 100% fittin...	0
#Traces where move log+model ...	3
#Move model only (in all traces)	3
#Traces where move model only ...	3

* Click a place/transition on the projected model to see its stats, or use c...

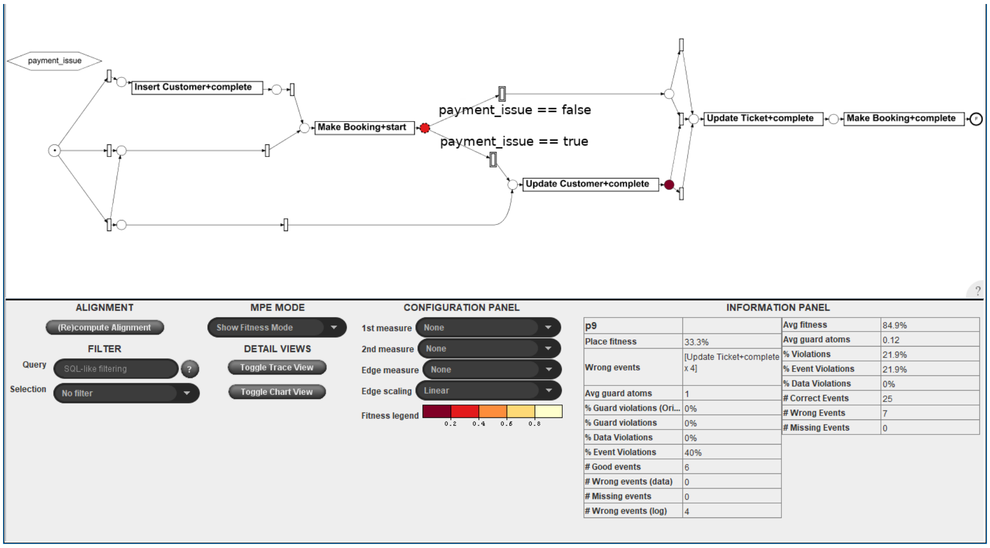
Figure 6.12: Conformance details for activity *Update Customer* on the model shown in Figure 6.11.

tau or silent transitions), or a mixture of both (*Update Customer*). More information about the aggregated statistics on the alignments where this activity is involved can be observed in Figure 6.13: three traces performed a synchronous move (log and model) in this activity; another three traces performed a model only move.

The asynchronous moves observed in the resulting alignments are due to deviations in the log with respect to the model. This means that the log contains behavior that does not fit in the process model. Not all deviations are undesirable, since they can represent exceptional cases that we do not want to be reflected on or even allowed by the general model. In other situations, deviations can uncover problems in the process that must be investigated in more detail. A more permissive process model that allows for more behavior would be able to replay more traces with less model and log moves. Therefore, it is important to choose a process model at the right level of generalization, which depends on the goal that we want to achieve when performing a conformance analysis.

Conformance Checking for Data Petri Nets

The previous technique allowed us to compare the behavior captured in an event log with respect to the behavior allowed by a process model. However, only control-



Query: SQL-like filtering
Selection: No filter
MPE mode: Show Fitness Mode
1st measure: None
2nd measure: None
Edge measure: None
Edge scaling: Linear

p9	
Place fitness	33.3%
Wrong events	[Update Ticket+complete x 4]
Avg guard atoms	1
% Guard violations (Ori...	0%
% Guard violations	0%
% Data Violations	0%
% Event Violations	40%
# Good events	6
# Wrong events (data)	0
# Missing events	0
# Wrong events (log)	4

Avg fitness	84.9%
Avg guard atoms	0.12
% Violations	21.9%
% Event Violations	21.9%
% Data Violations	0%
# Correct Events	25
# Wrong Events	7
# Missing Events	0

Figure 6.13: Conformance checking result on the sample event log and a data Petri net using the ProM plug-in Multi-perspective process explorer.

flow behavior was taken into account when computing the alignments. As we have seen before, it is possible to discover process models that restrict the behavior based on data guards. When data process models are available, it is possible to compute alignments that take data guards into account in order to discover data violations in the event log. The technique described in [75] makes this possible, taking a data Petri net and an event log as inputs, and computing data-aware alignments in order to provide a conformance analysis. For this example, we have used the plug-in *Multi-perspective process explorer* [74], which provides the mentioned functionality to compute alignments on data Petri nets.

Figure 6.13 shows a process model after the data-aware alignments have been computed. Places are color coded based on their *fitness* value (ratio of deviations observed in the place with respect to all the aligned traces). We see that place *p9* (on the right side of transition *Make Booking+start*) shows a low fitness value. In the information panel (bottom right) there are some details about the alignment. We see that there are no data violations at model or place level. That means that, despite showing deviations, they do not violate any of the data guards established on the model, which involve the data attribute *payment_issue*.

6.4.4 Performance Analysis

The last group of process mining techniques we show is devoted to performance analysis of processes. A difference with the presented techniques for conformance analysis is that, in order to analyze performance, we do not necessarily require a process model. In fact, we will describe two methods to analyze performance: one based on log replay

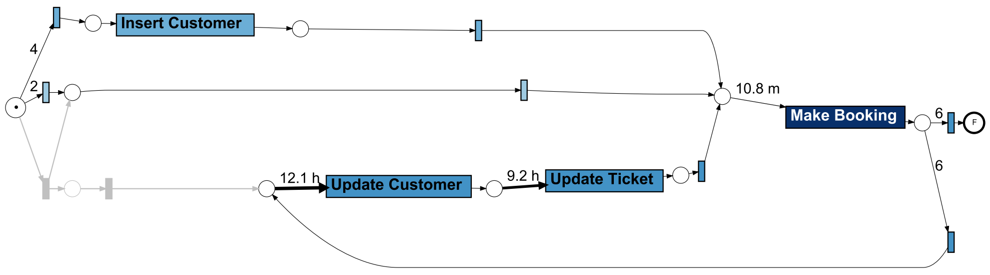


Figure 6.14: Performance metrics computed based on the alignment of an event log and a Petri net.

on model and another one based on event log analysis only.

Replay a Log on Petri Net for Performance/Conformance Analysis

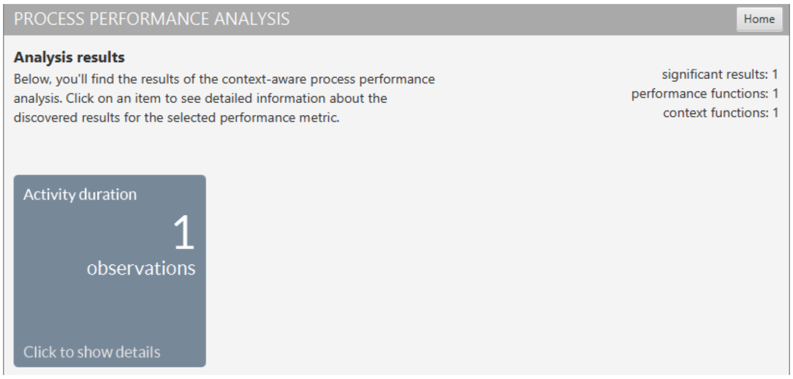
A way to analyze the performance of processes based on an event log is to look at the individual activity names and compute performance metrics that depend on a single activity instance, such as duration. In order to compute the duration of an activity instance, we typically compute the time difference between the complete and the start life-cycle phases. However, for other metrics such as waiting time it is necessary to compute the time spent between the completion of an activity and its predecessor. This can be done at the trace level considering event windows of size two. However, this can result in unreliable results when parallelism comes into play. In that case, it is necessary to find, for each event involved in a parallel branch, the closest previous event that happened just before a parallel split. This can be easily achieved by computing alignments first and performance metrics afterwards.

Figure 6.14 shows the result of computing performance metrics on an event log aligned with a Petri net. We see that waiting times are displayed on top of the incoming edges of the transitions, and that *Update Customer* has an average waiting time of 12.1 hours. This type of visualization allows us to identify sources of inefficiency and bottlenecks in our process. This is of great importance to improve the efficiency and performance of business processes.

Context-Aware Process Performance Analysis

Another way to tackle performance analysis is to focus only on the evidence, i.e., the event log without considering an explicit model. This is the philosophy adopted by the technique described in [52]. This technique carries out a performance analysis considering other data aspects beyond control flow. It tries to find statistically significant differences in terms of performance between groups of traces that show a certain level of similarity, e.g., same value for a certain data attribute.

When performing this analysis on the sample event log, first we obtain a global summary of results. In this case, one significant result was obtained with respect to



PROCESS PERFORMANCE ANALYSIS

Home

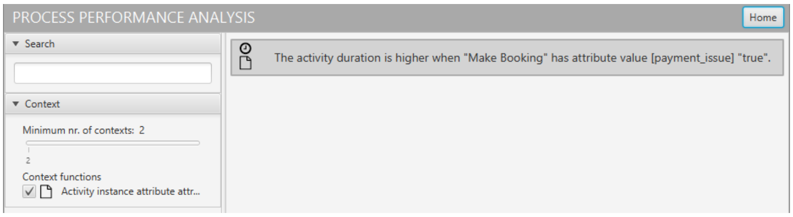
Analysis results

Below, you'll find the results of the context-aware process performance analysis. Click on an item to see detailed information about the discovered results for the selected performance metric.

significant results: 1
performance functions: 1
context functions: 1

Activity duration
1
observations
Click to show details

Figure 6.15: Summary of the results of a context-aware process performance analysis on the sample event log.



PROCESS PERFORMANCE ANALYSIS

Home

Search

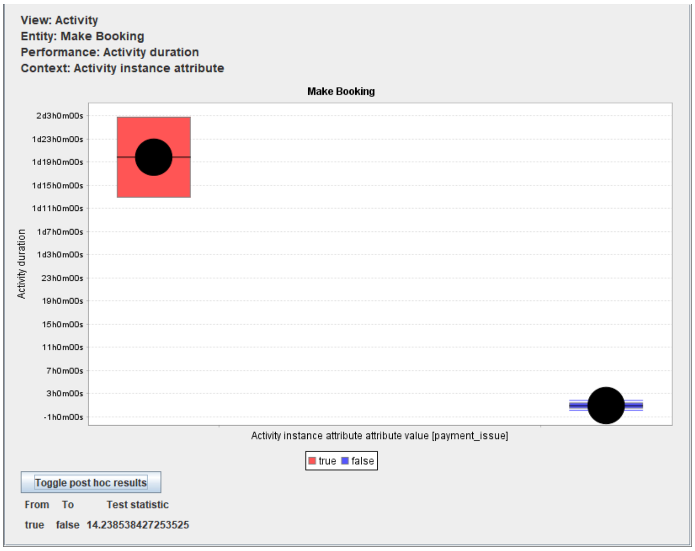
Context

Minimum nr. of contexts: 2

Context functions
✓ Activity instance attribute attr...

The activity duration is higher when "Make Booking" has attribute value [payment_issue] "true".

Figure 6.16: Description of the observation detected by the context-aware process performance analysis plug-in.



View: Activity
Entity: Make Booking
Performance: Activity duration
Context: Activity instance attribute

From	To	Test statistic
true	false	14.238538427253525

Figure 6.17: Boxplot of the differences in terms of duration for activity instances of *Make Booking* with respect to the value of the attribute *payment_issue*.

the duration of an activity, as can be seen in Figure 6.15. A descriptive message in natural language (Figure 6.16) explains the reason of the detected observation: *The duration is higher when "Make Booking" has attribute value [payment_issue] "true".* A more detailed view on the performance difference in terms of duration for *Make Booking* can be observed in Figure 6.17. A boxplot shows that when the attribute *payment_issue* has the value *true*, the duration of the *Make Booking* activity is, on average, more than one day longer. When the value of *payment_issue* is *false*, the average duration of the activity instances of *Make Booking* is around two hours.

This shows how we can leverage on the presence of additional data attributes in order to obtain a more detailed performance analysis and better insights. Without considering data attributes, we would only observe an unusually high average duration of the activity *Make Booking*. In order to find the reason, we would need to dig deeper and, possibly, return to the source of data to find more details about the root of the detected inefficiencies.

6.5 Chapter Summary

The focus of this thesis is on the data rather than on new process analytics. The thesis provides techniques to extract event data and to transform these into event logs. However, the event logs are not the goal of a process mining project, but the means to obtain valuable insights. In this chapter we have shown a sample of the possibilities that process mining offers in order to analyze event logs. The goal was not to present an exhaustive list of techniques, but to provide an idea, by means of examples, on how process mining techniques can exploit different data properties encoded within event logs. First, we presented the selected approaches with respect to their data requirements, inputs, and outputs. Next, we provided a sample event log showing the presented data properties. Finally, this event log was analyzed using each of the proposed techniques and the resulting output of each of them was explained.

Beekeeper placing an uncapped frame inside of an extractor.

"Cours complet d'apiculture", Georges de Layens and Gaston Bonnier, 1897

7 Data-Aware Process Oriented Querying

In the previous chapters, several techniques have been introduced that help us to extract and store event data from different event stores and databases (Chapters 3 and 4). Also, we have seen how it is possible to build event logs from different perspectives (Chapter 5). Next, we have explored some of the types of analysis that can be performed on these data using different process mining techniques (Chapter 6). The goal of this chapter is to provide the mechanisms to *effectively query process data*. We propose a domain-specific query language, focused on process data, that takes advantage of data stores based on the OpenSLEX meta-model presented in Chapter 3. The use of the OpenSLEX meta-model helps to ease the data querying task. The proposed query language allows us to explore the data extracted by the methods presented in Chapter 4, refine the result of the event log building process in Chapter 5, and to focus on the interesting parts of the event logs and related data.

7.1 Introduction

One of the main goals of process mining techniques is to obtain insights into the behavior of systems, companies, business processes, or any kind of workflow under study. Obviously, it is important to perform the analysis on the right data. Being able to extract and query some specific subset of the data becomes crucial when dealing with complex and heterogeneous datasets. In addition, the use of querying tools allows one to find specific cases or exceptional behavior. Whatever the goal, analysts often find themselves in the situation in which they need to develop ad-hoc

Query 7.1: DAPOQ-Lang query to retrieve cases with an event happening between two dates that changed the address of a customer from “Fifth Avenue” to “Sunset Boulevard”.

```
def P1 = createPeriod("1986/09/17 00:00","2016/11/30 19:44","yyyy/MM/dd HH:mm")

casesOf(
  eventsOf(
    versionsOf(
      allClasses().where {name == "CUSTOMER"}
    ).where { changed([at:"ADDRESS", from:"Fifth Avenue", to:"Sunset Boulevard"])}
  ).where
  {
    def P2 = createPeriod(it.timestamp)
    during(P2,P1)
  }
)
```

software to deal with specific datasets, given that existing tools might be difficult to use, too general, or just not suitable for process analysis.

Different approaches exist to support the *querying of process data*. Some of them belong to the field of Business Process Management (BPM). In this field, events are the main source of information. They represent transactions or activities that were executed at a certain moment in time in the environment under study. Querying this kind of data allows us to obtain valuable information about the behavior and execution of processes. There are other approaches originating from the field of data provenance [14], which are mainly concerned with recording and observing the origins of data. This field is closely related to that of *scientific workflows* [25] in which the traceability of the origin of experimental results becomes crucial to guarantee correctness and reproducibility. In the literature, we find many languages to query process data. However, none of these approaches succeeds at combining process and data aspects in an integrated way. An additional challenge to overcome is the development of a query mechanism that allows exploiting this combination while being intuitive and easy to use.

In order to make the querying of process event data easier and more efficient, we propose the *Data-Aware Process Oriented Query Language* (DAPOQ-Lang). This query language, first introduced in [41], exploits both process and data perspectives. The aim of DAPOQ-Lang is not to theoretically enable new types of computations, but to ease the task of writing queries in the specific domain of process mining. Therefore, our focus is on the ease of use. We want to illustrate the ease of use of DAPOQ-Lang using an example. Let us consider a generic question that could be asked by an analyst when carrying out a process mining project:

> **GQ:** In which cases there was (a) an event that happened between time T1 and T2, (b) that performed a modification in a version of class C (c) in which the value of field F changed from X to Y?

This query involves several types of elements: cases, events, object versions, and

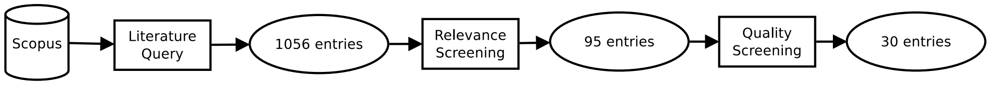


Figure 7.1: Pipeline of the systematic review process

attributes. We instantiate this query with some specific values for T1 = “1986/09/17 00:00”, T2 = “2016/11/30 19:44”, C = “CUSTOMER”, F = “ADDRESS”, X = “Fifth Avenue”, and Y = “Sunset Boulevard”. Query 7.1 presents the corresponding DAPOQ-Lang query. This example shows how compact a DAPOQ-Lang query can be. The specifics of this query will be explained in the coming sections.

The rest of this chapter is organized as follows. In Section 7.2, we show the related work with a systematic literature review on process data query languages. Section 7.3 presents the query language, focusing on the syntax and semantics. Section 7.4 provides information about the implementation and its evaluation. Section 7.5 presents possible use cases. Section 7.6 concludes the chapter.

7.2 Systematic Literature Review

In order to get an overview of existing approaches for event data querying, we first concluded a systematic literature review [115]. Figure 7.1 shows an overview of the procedure. First, a coarse set of candidate papers needs to be obtained from a scientific publications database or through a search engine by means of a *literature query*. Afterwards, a *relevance screening* is performed in order to identify which papers are actually within the scope. We define a set of criteria. Only papers that fulfill these criteria pass to the next phase. Next, a *quality screening* is conducted on the relevant papers. This is done by defining minimum quality criteria that the papers must satisfy. Finally, with the selected papers that are relevant and have sufficient quality, a detailed review is performed.

In accordance with the procedure described in Figure 7.1, first, we performed a search of related papers. We chose Scopus[1], one of the largest abstract and citation databases of peer-reviewed literature, including scientific journals, books, and conference proceedings. This database provides a search engine that, by means of queries, allows to specify different kinds of criteria to filter the results.

In our case, we are interested in papers that refer to *business processes* or *workflows*, that relate to *queries* and that analyze *events*, *logs*, *provenance*, *data* or *transactions*. In addition, we want to filter out any work that does not belong to the area of *Computer Science*, or that is not written in English. The exact query as executed by the search engine can be observed in Literature Query 7.2.

The query above returned 1056 results, from the years 1994 to 2018, with the distribution depicted in Figure 7.2.

However, not all the papers found proved relevant to our topic. When performing

[1] `http://www.scopus.com`

Literature Query 7.2: Query as executed in Scopus

```
TITLE-ABS-KEY("business process" OR "workflow") AND
TITLE-ABS-KEY("query" OR "querying") AND
TITLE-ABS-KEY("event" OR "log" OR "provenance" OR "data" OR "transaction") AND
LIMIT-TO(SUBJAREA,"COMP") AND
LIMIT-TO(LANGUAGE,"English")
```

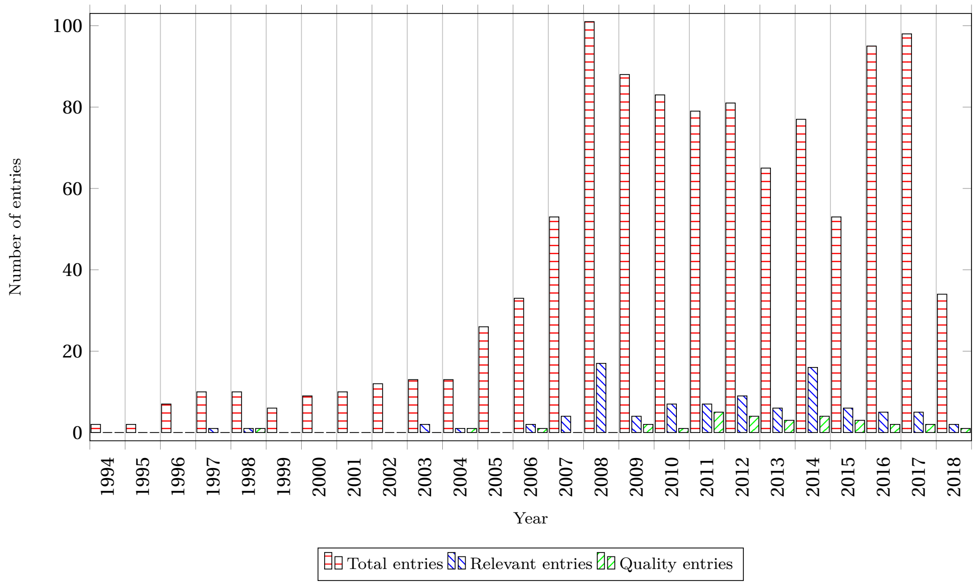


Figure 7.2: Distribution per year of the related work as a result of the first query on Scopus.

the *relevance screening*, we make a distinction between Inclusive (I) and Exclusive (E) criteria. In this way, candidates that satisfy *all* the inclusive criteria and do not satisfy any of the exclusive will be included in the next phase of the review. We will discard any paper that does not satisfy all the inclusive criteria or satisfies at least one of the exclusive ones. The specific criteria used in this review are listed below:

1. Does the study consider process data as input? (I)

2. Does the study use only process models as input? (E)

3. Does the study propose a language to query the data? (I)

As a result of the *relevance screening* phase, the whole set of 1056 entries was reduced to a set of 95 relevant works. These 95 entries are considered to be related and within the scope of process and provenance data querying. However, to guarantee a minimum level of detail, we defined the following criteria for the *quality screening* phase:

1. Does the study provide a sufficiently detailed description of the language?
2. If the language already exists, are the extensions, modifications, adaptations sufficiently explained?
3. Does the study include concrete examples of the application of the language?

As a result of this phase, the set of 95 relevant works was further reduced to a set of 30 truly related papers to be analyzed in detail. At the final stage, these papers have been analyzed to identify their most important features, and then compared to our approach. The content of each paper was reviewed, closely considering the main characteristics of the approaches they describe. These characteristics refer to the kind of input data that is used by each approach (*Input data aspects*), qualities related to provenance (*Provenance aspects*), business processes (*Business process aspects*), database or artifact environments (*Database or Artifact aspects*), and the nature of the questions that can be queried with them (*Query aspects*). Table 7.1 presents the main characteristics of the remaining 30 references and how they can be classified when looking at the features listed below:

Input data aspects:

- **Event data**: The approach allows to query and analyze event data.
- **Model-based**: The approach takes into account execution models such as workflows, BPEL, or Petri nets.
- **Storage model**: A meta-model for provenance or event data storage is proposed by the approach.
- **Complex event processing**: Different sources of events are analyzed by the approach to infer new kinds of more complex events.
- **Streams**: It is possible to query streams of events instead of complete logs.

Provenance aspects:

- **Provenance-oriented**: The approach is provenance oriented or allows to record and query provenance information on the execution of workflows, scientific workflows, business processes, etc.
- **OPM-compliant**: The storage model used by the approach complies with the Open Provenance Model [83]. This model proposes a standard on how provenance data must be stored in order to ensure interoperability between tools.
- **Data lineage**: The language allows to query about the life cycle of data, its origins and where it moves over time.
- **Dependency graphs/relations**: Relations between data entities are considered by the approach. For example, dependency graphs, common in the provenance area, are used.

Business Process aspects:

- **Business process oriented**: The approach is applied to the field of business processes management. In addition, it considers business process specific aspects while querying, e.g., using concepts such as activities, cases, resources, etc.

Database or Artifact aspects:

- **Entities/artifacts**: The approach captures information about the objects, data entities or artifacts related to the event or provenance data. This information can be queried as well.
- **Database-oriented**: The approach captures database-specific data such as schema, tables, column information, keys, objects, etc., in case the event information is extracted from that kind of environment.
- **State-aware**: The state of the analyzed system is considered by the approach at the time of capturing and querying data. This can be related to process state, data state, versioning, etc.

Query aspects:

- **Graph-based**: Queries and results are expressed as graphs, in which edges are relations and nodes can be data objects, events, activities, etc.
- **Relevance querying**: It is possible to query about data relevance, i.e., relations between data that do not only reflect data origin.
- **Semantic querying**: The query language is based on or compatible with semantic technologies such as RDF, OWL, SPARQL, etc.
- **Regular path queries (RPQ)**: The language allows to make queries that select nodes connected by a path on a graph based database.
- **Projection queries**: It is possible to query cases that fit a partial pattern using projection.
- **Temporal properties/querying**: The language or technique makes it possible to query temporal properties related to the data. It can also refer to temporal logic languages such as LTL, CTL, etc.
- **Event correlation**: The approach does not consider events in isolation, but allows to query the correlation between them, e.g., querying evens related to the same artifact.
- **Multi process**: The approach allows to query aspects related to several processes at the same time on the same dataset.
- **Multi log**: Several event logs can be queried at the same time in order to combine results for a single query.

Table 7.1: Comparison of features for the references at the end of the systematic review.

Ref	Title	Event data	Model-based	Storage model	Complex event processing	Streams	Provenance-oriented	OPM-compliant	Data lineage	Dependency graphs/Rels.	Business process oriented	Entities/Artifacts	Database-oriented	State-aware	Graph-based	Relevance querying	Semantic querying	Regular path queries	Projection queries	Temporal prop./querying	Event correlation	Multi process/log/schema
[54]	Answering regular path queries on workflow provenance	✓	-	-	-	-	✓	-	-	-	-	-	-	-	✓	-	-	✓	-	-	-	-
[23]	Capturing and querying workflow runtime provenance with PROV: A practical approach	✓	-	✓	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-
[24]	Modeling and querying scientific workflow provenance in the D-OPM	✓	-	✓	-	-	✓	✓	-	-	-	-	-	-	✓	-	-	✓	-	-	-	-
[99]	Towards a scalable semantic provenance management system	✓	-	-	-	-	✓	✓	✓	✓	-	-	-	-	-	-	✓	-	-	-	-	-
[22]	Towards integrating workflow and database provenance	✓	-	✓	-	-	✓	✓	✓	✓	-	✓	✓	✓	-	-	-	-	-	-	-	-
[33]	MTCProv: A practical provenance query framework for many-task scientific computing	✓	-	✓	-	-	✓	✓	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-
[67]	OPQL: A first OPM-level query language for scientific workflow provenance	✓	-	-	-	-	✓	✓	✓	-	-	-	-	-	✓	-	-	-	-	-	-	-
[68]	Storing, reasoning, and querying OPM-compliant scientific workflow provenance using relational databases	✓	-	✓	-	-	✓	✓	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-
[69]	XQuery meets Datalog: Data relevance query for workflow trustworthiness	✓	-	-	-	-	✓	-	-	✓	-	-	-	-	-	✓	-	-	-	-	-	-
[11]	A model for user-oriented data provenance in pipelined scientific workflows	✓	-	✓	-	-	✓	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-
[27]	Developing provenance-aware query systems: an occurrence-centric approach	✓	-	✓	-	-	✓	-	✓	✓	-	✓	-	-	-	-	-	-	-	✓	-	-
[102]	A knowledge driven approach towards the validation of externally acquired traceability datasets in supply chain business processes	✓	-	-	-	-	-	-	✓	-	✓	-	-	-	-	-	✓	-	-	-	-	-
[82]	Process query language: A way to make workflow processes more flexible	✓	-	✓	-	-	-	-	✓	-	✓	-	-	-	-	-	-	-	-	-	-	-
[59]	Workflow History Management	✓	-	✓	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
[93]	The HIT model: Workflow-aware event stream monitoring	✓	✓	-	✓	✓	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
[70]	Semantic Enabled complex Event Language for business process monitoring	✓	-	-	✓	-	-	-	✓	-	✓	-	-	-	✓	-	✓	-	✓	✓	-	-
[29]	A framework supporting the analysis of process logs stored in either relational or NoSQL DBMSS	✓	-	✓	-	-	-	-	-	-	✓	-	-	-	✓	-	-	-	-	-	-	-
[94]	Business impact analysis: a framework for a comprehensive analysis and optimization of business processes	✓	-	✓	-	-	-	-	-	✓	✓	✓	✓	-	-	-	-	-	-	-	-	-
[5]	Model-driven event query generation for business process monitoring	✓	✓	-	✓	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
[103]	Querying process models based on the temporal relations between tasks	-	✓	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	✓	-	-
[8]	A query language for analyzing business processes execution	✓	-	-	-	-	-	-	✓	✓	✓	✓	-	-	✓	-	✓	-	-	-	✓	-
[26]	Top-k projection queries for probabilistic business processes	✓	✓	✓	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	✓	-	-	-
[6]	Integration of Event Data from Heterogeneous Systems to Support Business Process Analysis	✓	✓	✓	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	✓	-
[78]	Enabling semantic complex event processing in the domain of logistics	✓	-	-	✓	-	-	-	-	-	✓	-	-	-	-	-	✓	-	-	-	-	-
[96]	Optimizing complex sequence pattern extraction using caching	✓	-	-	✓	-	-	-	-	-	✓	-	-	-	-	-	-	-	✓	-	-	-
[10]	Trace retrieval for business process operational support	✓	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	✓	-	-	-
[91]	Process Instance Query Language to Include Process Performance Indicators in DMN	✓	-	-	-	-	-	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-
[105]	Querying Workflow Logs	✓	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	✓	-	-	-
[4]	A Continuous Query Language for Stream-Based Artifacts	✓	-	-	✓	✓	-	-	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-
[95]	Log-based understanding of business processes through temporal logic query checking	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-
	Our approach: DAPOQ-Lang	✓	✓	✓	-	-	✓	-	✓	✓	✓	✓	✓	✓	-	✓	-	-	-	✓	✓	✓

- **Multi data schema**: Several data schemas can be considered in a single query.

If we focus on the columns named *Provenance-oriented* and *Business process oriented* in Table 7.1, we notice that none of the approaches qualifies as *provenance-oriented* and *business oriented* at the same time. In fact, all the approaches seem to present one of these two aspects. Therefore, we categorized the approaches into two groups: *provenance-oriented* and *business process oriented.* The *provenance-oriented* approaches ([11,22–24,27,33,54,67–69,99]) usually support some kind of provenance model, data lineage or so. However, not all the approaches under this category support every aspect of data provenance. Only one of them [22] is database-oriented and considers states and artifacts. Most of the *business process oriented* approaches ([4–6,8,10,26,29,59,70,78,82,91,93,94,96,102,103,105]) seem to ignore data provenance aspects. They focus mainly on capturing causal relations of business activities and supporting different ways to query the data. There is an outlier [95] that focuses only on the temporal analysis of event logs using temporal logic checking. However, this solution ignores all other aspects of the data. If we focus on the last eight columns at the right of Table 7.1, we see that most of the approaches only present one or two of what we call *Query aspects.* There are two exceptions though. The first one is the approach presented in [70], which supports *graph-based*, *semantic*, *projection*, and *temporal properties* querying. However, it does not support any database aspect. The second exception is the approach in [8], which supports *graph-based*, *semantic*, and *event correlation* queries, as well as *artifacts.* However, it lacks support for *temporal properties* and it is not *database-oriented.* As can be seen, none of the existing approaches succeeds at combining data provenance and business processes. Additionally, none of them is able to provide good support for different querying aspects while being database-oriented.

The insight from this literature review is that in the field of process data querying, there is a need for a solution that combines business process analysis with the data perspective, which also allows querying all this information in an integrated way. Taking into account that, in most cases, the execution of business processes is supported by databases, the consideration of the data perspective becomes especially relevant.

7.3 DAPOQ-Lang

DAPOQ-Lang is a data-aware process oriented query language that allows the user to query data and process information stored in a structure compatible with the OpenSLEX meta-model [42]. As described in Chapter 3, OpenSLEX combines database elements (data models, objects, and object versions) with common process mining data (events, logs, and processes), considering them as first-class citizens. DAPOQ-Lang considers the same first-class citizens as OpenSLEX, which makes it possible to write queries in the process mining domain enriched with data aspects with lower complexity than in other general purpose query languages like SQL.

Intuitively, we could think that considering all the elements of the OpenSLEX meta-model as first-class citizens would introduce a lot of complexity in our language.

However, these elements have been organized in a type hierarchy as subtypes of higher level superclasses (Figure 7.3). In this figure, it can be seen that *MMElement* is the highest level *superclass*. Next, we distinguish two subtypes of elements: (1) stored elements (*StoredElement*), i.e., elements that can be found directly stored in an OpenSLEX structure, such as activities, events, objects, and logs; and (2) computed elements (*ComputedElements*), i.e., elements that are computed based on the rest, e.g., temporal periods of cases, and temporal periods of events. We will exploit this hierarchy to design a simple language, providing many basic functions that can operate on any *MMElement*, as well as some functions that focus on specific subtypes. Given a connected meta-model instance $CMI = (DM, OC, classOfObject, OVC, objectOfVersion, EC, eventToOVLabel, IC, eventAI, PMC, activityOfAI, processOfLog)$ (Definition 22), we define the concept of *MMElement* in DAPOQ-Lang as the union of all its terminal subtypes:

$$\begin{aligned} MMElement = \; & AC \cup LG \cup EV \cup REL \cup OC \cup AT \cup CL \cup PER \cup PM \cup CS \cup AI \cup \\ & OV \cup RS \cup DM \end{aligned} \tag{7.1}$$

Some of the functions that we define operate on sets of elements ($\mathscr{P}(MMElement)$). However, as a constraint of our query language, we require that the elements of those sets belong to the same subtype (i.e., a set of *Class* elements, a set of *Version* elements, or a set of *Event* elements, etc.). Therefore, we define the set *MMSets* as the set of all possible subsets of each element type in a meta-model *MM*:

$$\begin{aligned} MMSets = \; & \mathscr{P}(AC) \cup \mathscr{P}(LG) \cup \mathscr{P}(EV) \cup \mathscr{P}(REL) \cup \mathscr{P}(OC) \cup \mathscr{P}(AT) \cup \mathscr{P}(CL) \cup \\ & \mathscr{P}(PER) \cup \mathscr{P}(PM) \cup \mathscr{P}(CS) \cup \mathscr{P}(AI) \cup \mathscr{P}(OV) \cup \mathscr{P}(RS) \cup \mathscr{P}(DM) \end{aligned} \tag{7.2}$$

The output of any query will be an element set $es \in MMSets$, i.e., a set of elements of the same type. The following subsections describe the syntax and semantics of DAPOQ-Lang in detail.

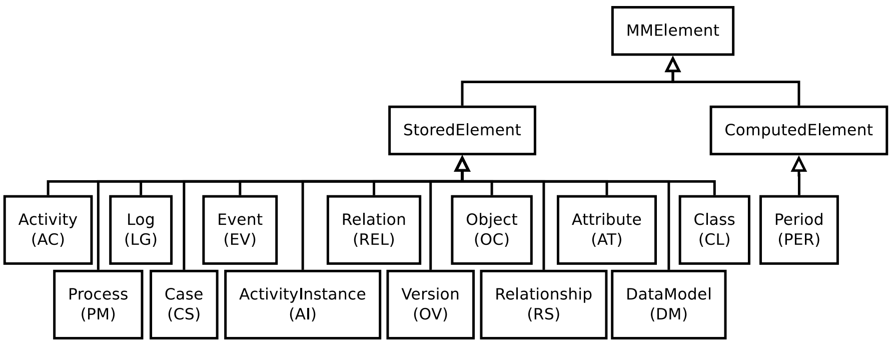


Figure 7.3: DAPOQ-Lang types hierarchy in UML notation. Arrows indicate subtype relations.

7.3.1 Syntax

The DAPOQ-Lang language has been designed with a focus on the *ease of use*, trying to find a balance between simplicity and expressive power. We exploited the specifics of the underlying meta-model defining a total of 57 basic functions, as organized in five well-defined blocks, that can be applied in the context of a given meta-model MM. The functions proposed in this section are used to define the syntax, while Section 7.3.2 will focus on the semantics of DAPOQ-Lang.

Terminal Meta-Model Elements

We define a set of 13 basic terminal functions. Each of them maps to the set of all elements of the corresponding type (Figure 7.3) in the OpenSLEX meta-model structure (Definition 22). Given a connected meta-model instance, we define the following :

1. *allDatamodels*: the set of all data models, i.e., DM,
2. *allClasses*: the set of all classes, i.e., CL,
3. *allAttributes*: the set of all class attributes, i.e., AT,
4. *allRelationships*: the set of all class relationships, i.e., RS,
5. *allObjects*: the set of all objects, i.e., OC,
6. *allVersions*: the set of all object versions, i.e., OV,
7. *allRelations*: the set of all relations, i.e., REL,
8. *allEvents*: the set of all events, i.e., EV,
9. *allActivityInstances*: the set of all activity instances, i.e., AI,
10. *allCases*: the set of all cases, i.e., CS,
11. *allLogs*: the set of all logs, i.e., LG,
12. *allActivities*: the set of all activities, i.e., AC,
13. *allProcesses*: the set of all processes, i.e., PM.

Elements Related to Elements

The following 14 functions take as an input a set of elements of the same type and return a set of elements related to them of the type corresponding to the return type of the function. For example, a call to *eventsOf*(*es*), being $es \in \mathscr{P}(LG)$ will return the set of events that are related to the logs in the set es. Thanks to the subtype hierarchy, in most of the cases we can reuse the same function call for input sets of any type, which leads to a more compact syntax. In the cases when an input of any type would not make sense, we can still restrict the input type to a particular kind.

This is the case with the function *versionsRelatedTo*, which only accepts sets of object versions as input.

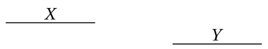
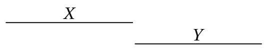
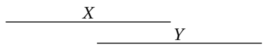
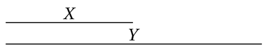
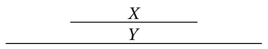
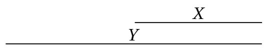
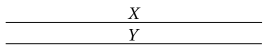
14. $\mathit{datamodelsOf} \in \mathit{MMSets} \rightarrow \mathscr{P}(\mathit{DM})$: returns the set of data models related to the input,
15. $\mathit{classesOf} \in \mathit{MMSets} \rightarrow \mathscr{P}(\mathit{CL})$: returns the set of classes related to the input,
16. $\mathit{attributesOf} \in \mathit{MMSets} \rightarrow \mathscr{P}(A)$: returns the set of attributes related to the input,
17. $\mathit{relationshipsOf} \in \mathit{MMSets} \rightarrow \mathscr{P}(\mathit{RS})$: returns the set of relationships related to the input,
18. $\mathit{objectsOf} \in \mathit{MMSets} \rightarrow \mathscr{P}(O)$: returns the set of objects related to the input,
19. $\mathit{versionsOf} \in \mathit{MMSets} \rightarrow \mathscr{P}(\mathit{OV})$: returns the set of object versions related to the input,
20. $\mathit{relationsOf} \in \mathit{MMSets} \rightarrow \mathscr{P}(\mathit{REL})$: returns the set of relations related to the input,
21. $\mathit{eventsOf} \in \mathit{MMSets} \rightarrow \mathscr{P}(E)$: returns the set of events related to the input,
22. $\mathit{activityInstancesOf} \in \mathit{MMSets} \rightarrow \mathscr{P}(\mathit{AI})$: returns the set of activity instances related to the input,
23. $\mathit{activitiesOf} \in \mathit{MMSets} \rightarrow \mathscr{P}(\mathit{AC})$: returns the set of activities related to the input,
24. $\mathit{casesOf} \in \mathit{MMSets} \rightarrow \mathscr{P}(\mathit{CS})$: returns the set of cases related to the input,
25. $\mathit{logsOf} \in \mathit{MMSets} \rightarrow \mathscr{P}(\mathit{LG})$: returns the set of logs related to the input,
26. $\mathit{processesOf} \in \mathit{MMSets} \rightarrow \mathscr{P}(\mathit{PM})$: returns the set of processes related to the input,
27. $\mathit{versionsRelatedTo} \in \mathscr{P}(\mathit{OV}) \rightarrow \mathscr{P}(\mathit{OV})$: returns the set of object versions directly related (distance 1) to the input object versions through any kind of relationship.

Computation of Temporal Values

Some elements in our meta-model contain temporal properties (e.g., events have a timestamp, object versions have a lifespan, etc.), which allows us to make temporal computations with them. We provide the following eight functions to compute time periods (with a start and an end), as well as durations. Durations (DUR) are special in the sense that they can be considered as scalars. Also, they are not part of the *MMElement* subtype hierarchy. Durations can only be used to be compared with other durations.

28. $\mathit{periodsOf} \in \mathit{MMSets} \rightarrow \mathscr{P}(\mathit{PER})$: returns the computed periods for each of the elements of the input set,
29. $\mathit{globalPeriodOf} \in \mathit{MMSets} \rightarrow \mathit{PER}$: returns a global period for all the elements in the input set, i.e., the period from the earliest to the latest timestamp,

Table 7.2: Relations in Allen's interval algebra.

Relation	**Name**	**Illustration**	**Interpretation**
$X < Y$ $Y > X$	before after		X takes place before Y
$X\ \mathbf{m}\ Y$ $Y\ \mathbf{mi}\ X$	meets meetsInv		X meets Y (*i* stands for ***i**nverse*)
$X\ \mathbf{o}\ Y$ $Y\ \mathbf{oi}\ X$	overlaps overlapsInv		X overlaps with Y
$X\ \mathbf{s}\ Y$ $Y\ \mathbf{si}\ X$	starts startsInv		X starts Y
$X\ \mathbf{d}\ Y$ $Y\ \mathbf{di}\ X$	during duringInv		X during Y
$X\ \mathbf{f}\ Y$ $Y\ \mathbf{fi}\ X$	finishes finishesInv		X finishes Y
$X = Y$	matches		X is equal to Y

30. *createPeriod* $\in TS \times TS \rightarrow PER$: returns a period for the specified start and end timestamps,

31. *getDuration* $\in PER \rightarrow DUR$: returns the duration of a period in milliseconds,

32. *Duration.ofSeconds* $\in \mathbb{N} \rightarrow DUR$: returns the duration of the specified seconds,

33. *Duration.ofMinutes* $\in \mathbb{N} \rightarrow DUR$: returns the duration of the specified minutes,

34. *Duration.ofHours* $\in \mathbb{N} \rightarrow DUR$: returns the duration of the specified hours,

35. *Duration.ofDays* $\in \mathbb{N} \rightarrow DUR$: returns the duration of the specified days.

Temporal Interval Algebra

Allen's Interval Algebra, described in [2], introduces a calculus for temporal reasoning, which defines possible relations between time intervals. It provides the tools to reason about the temporal descriptions of events in the broadest sense. Table 7.2 shows the 13 base relations between two intervals. These temporal relations are used in our approach to reason about data elements for which we can compute a temporal interval.

We have introduced the functions to compute and create periods of time. The following 13 functions cover all the interval operators, described by Allen's Interval Algebra, which we can use to compare periods. Take (a, b) to be a pair of periods for which:

36. *before* $\in PER \times PER \rightarrow \mathbb{B}$: *true* if a takes place before b,

37. *after* $\in PER \times PER \rightarrow \mathbb{B}$: *true* if a takes place after b,

38. *meets* $\in PER \times PER \rightarrow \mathbb{B}$: *true* if the end of a is equal to the start of b,

39. *meetsInv* $\in PER \times PER \to \mathbb{B}$: *true* if the start *a* is equal to the end of *b*,
40. *overlaps* $\in PER \times PER \to \mathbb{B}$: *true* if the end of *a* happens during *b*,
41. *overlapsInv* $\in PER \times PER \to \mathbb{B}$: *true* if the start of *a* happens during *b*,
42. *starts* $\in PER \times PER \to \mathbb{B}$: *true* if both start at the same time, but *a* is shorter,
43. *startsInv* $\in PER \times PER \to \mathbb{B}$: *true* if both start at the same time, but *a* is longer,
44. *during* $\in PER \times PER \to \mathbb{B}$: *true* if *a* starts after *b* started, and ends before *b* ends,
45. *duringInv* $\in PER \times PER \to \mathbb{B}$: *true* if *a* starts before *b* starts, and ends after *b* ends,
46. *finishes* $\in PER \times PER \to \mathbb{B}$: *true* if both end at the same time, but *a* is shorter,
47. *finishesInv* $\in PER \times PER \to \mathbb{B}$: *true* if both end at the same time, but *a* is longer,
48. *matches* $\in PER \times PER \to \mathbb{B}$: *true* if both have the same start and the same end.

Operators on Attributes of Elements

Some elements of the OpenSLEX meta-model can be enriched with attribute values. These elements are events, object versions, cases, and logs. The following functions allow the user to query these attributes and obtain their value:

49. *eventHasAttribute* $\in EVAT \times EV \to \mathbb{B}$: *true* if the event contains a value for a certain attribute name,
50. *versionHasAttribute* $\in AT \times OV \to \mathbb{B}$: *true* if the object version contains a value for a certain attribute name,
51. *caseHasAttribute* $\in CSAT \times CS \to \mathbb{B}$: *true* if the case contains a value for a certain attribute name,
52. *logHasAttribute* $\in LGAT \times LG \to \mathbb{B}$: *true* if the log contains a value for a certain attribute name,
53. *getAttributeEvent* $\in EVAT \times EV \not\to V$: returns the value for an attribute of an event,
54. *getAttributeVersion* $\in AT \times OV \not\to V$: returns the value for an attribute of an object version,
55. *getAttributeCase* $\in CSAT \times CS \not\to V$: returns the value for an attribute of a case,
56. *getAttributeLog* $\in LGAT \times LG \not\to V$: returns the value for an attribute of a log,
57. *versionChange* $\in AT \times V \times V \times OV \to \mathbb{B}$: *true* if the value for an attribute linked to an object version changed from a certain value (in the previous object version) to another (in the provided object version).

By definition, *getAttribute** functions (items 53 to 56) are only defined for combinations of elements and attributes for which the corresponding **HasAttribute* function (items 49 to 52) evaluates to *True*.

DAPOQ-Lang's Abstract Syntax

The syntax of DAPOQ-Lang is defined in the form of an abstract syntax, using the notation proposed in [81]. In DAPOQ-Lang, a query is a sequence of *Assignments* combined with an *ElementSet*:

$$\begin{aligned} \mathit{Query} &\triangleq s : \mathit{Assignments}; \mathit{es} : \mathit{ElementSet} \\ \mathit{Assignments} &\triangleq \mathit{Assignment}^{*} \end{aligned}$$

The result of a query is an *ElementSet*, i.e., the set of elements (of the same type) from the queried OpenSLEX dataset that satisfies certain criteria. An *Assignment* assigns an *ElementSet* to a variable. Then, any reference to such variable, via its identifier, will be replaced by the corresponding *ElementSet*.

$$\begin{aligned} \mathit{Assignment} &\triangleq v : \mathit{Varname}; \mathit{es} : \mathit{ElementSet} \\ \mathit{Varname} &\triangleq \mathit{identifier} \end{aligned}$$

An *ElementSet* can be defined over other *ElementSets* by *Construction* or *Application*. It can also be defined by means of a variable identifier, i.e., an *ElementSetVar*, by a call to a terminal element function with a *ElementSetTerminal* (Section 7.3.1), by computation or creation of *Periods*, or by filtering elements of the previous options with a *FilteredElementSet*.

$$\begin{aligned} \mathit{ElementSet} &\triangleq \mathit{Construction} \mid \mathit{Application} \mid \mathit{Period} \mid \mathit{ElementSetVar} \mid \\ &\quad\ \mathit{ElementSetTerminal} \mid \mathit{FilteredElementSet} \\ \mathit{ElementSetVar} &\triangleq \mathit{identifier} \end{aligned}$$

An *ElementSetTerminal* is the *ElementSet* resulting from a call to the corresponding terminal element function (e.g., allEvents).

$$\begin{aligned} \mathit{ElementSetTerminal} &\triangleq \mathit{AllDatamodels} \mid \mathit{AllClasses} \mid \mathit{AllAttributes} \mid \\ &\quad\ \mathit{AllRelationships} \mid \mathit{AllObjects} \mid \mathit{AllVersions} \mid \\ &\quad\ \mathit{AllRelations} \mid \mathit{AllActivityInstances} \mid \mathit{AllEvents} \mid \\ &\quad\ \mathit{AllCases} \mid \mathit{AllLogs} \mid \mathit{AllActivities} \mid \mathit{AllProcesses} \end{aligned}$$

An *ElementSet* can be composed from other *ElementSets* by applying set operations such as *union*, *exclusion*, and *intersection*.

$$\begin{aligned} \mathit{Construction} &\triangleq \mathit{es}_1, \mathit{es}_2 : \mathit{ElementSet}; o : \mathit{Set\_Op} \\ \mathit{Set\_Op} &\triangleq \mathit{Union} \mid \mathit{Excluding} \mid \mathit{Intersection} \end{aligned}$$

Also, an *ElementSet* can be constructed by means of a call to one of the *ElementOf_Op* functions. These include the functions described in Section 7.3.1 that return sets of elements related to other elements and the *periodsOf* function described in Section 7.3.1 that computes the periods of elements.

$$
\begin{aligned}
\mathit{Application} &\triangleq \mathit{es} : \mathit{ElementSet}; \mathit{o} : \mathit{ElementOf\_Op} \\
\mathit{ElementOf\_Op} &\triangleq \mathit{datamodelsOf} \mid \mathit{classesOf} \mid \mathit{attributesOf} \mid \mathit{relationshipsOf} \mid \\
&\quad \mathit{objectsOf} \mid \mathit{versionsOf} \mid \mathit{relationsOf} \mid \mathit{eventsOf} \mid \\
&\quad \mathit{activityInstancesOf} \mid \mathit{casesOf} \mid \mathit{activitiesOf} \mid \mathit{logsOf} \mid \mathit{processesOf} \mid \\
&\quad \mathit{periodsOf} \mid \mathit{versionsRelatedTo}
\end{aligned}
$$

An *ElementSet* can be built by means of filtering, discarding elements of another *ElementSet* according to certain criteria. These criteria are expressed as a *PredicateBlock*, which will be evaluated for each member of the input *ElementSet*. Depending on the result of evaluating the *PredicateBlock*, each element will be filtered out or included in the new *ElementSet*.

$$
\mathit{FilteredElementSet} \triangleq \mathit{es} : \mathit{ElementSet}; \mathit{pb} : \mathit{PredicateBlock}
$$

A *PredicateBlock* is a sequence of *Assignments* combined with a *Predicate*. Such *Predicate* can be defined recursively as a binary (*and*, *or*) or unary (*not*) combination of other *Predicates*.

$$
\begin{aligned}
\mathit{PredicateBlock} &\triangleq s : \mathit{Assignments}; p : \mathit{Predicate} \\
\mathit{Predicate} &\triangleq \mathit{AttributePredicate} \mid \mathit{Un\_Predicate} \mid \mathit{Bin\_Predicate} \mid \\
&\quad \mathit{TemporalPredicate} \\
\mathit{Bin\_Predicate} &\triangleq p_1, p2 : \mathit{Predicate}; o : \mathit{BinLogical\_Op} \\
\mathit{Un\_Predicate} &\triangleq p : \mathit{Predicate}; o : \mathit{UnLogical\_Op} \\
\mathit{BinLogical\_Op} &\triangleq \mathit{And} \mid \mathit{Or} \\
\mathit{UnLogical\_Op} &\triangleq \mathit{Not}
\end{aligned}
$$

Also, a *Predicate* can be defined as an *AttributePredicate*, which will either refer to *AttributeExists* functions that check the existence of an attribute for a certain element, an operation on attribute values (e.g., to compare attributes to substrings, constants, or other attributes), or an *AttributeChange* predicate, making use of the functions specified in Section 7.3.1.

$$
\begin{aligned}
\mathit{AttributePredicate} &\triangleq \mathit{AttributeExists} \mid \mathit{AttributeValuePred} \mid \mathit{AttributeChange}\\
\mathit{AttributeExists} &\triangleq \mathit{at} : \mathit{AttributeName}\\
\mathit{AttributeValuePred} &\triangleq \mathit{at_1}, \mathit{at_2} : \mathit{Attribute}; o : \mathit{Value\_Op}\\
\mathit{AttributeChange} &\triangleq \mathit{at} : \mathit{AttributeName}; \mathit{from}, \mathit{to} : \mathit{Value}\\
\mathit{AttributeName} &\triangleq \mathit{identifier}\\
\mathit{Value\_Op} &\triangleq \;==\mid>=\mid<=\mid>\mid<\mid \mathit{startsWith} \mid \mathit{endsWith} \mid \mathit{contains}\\
\mathit{Attribute} &\triangleq \mathit{AttributeName} \mid \mathit{Value}\\
\mathit{Value} &\triangleq \mathit{literal}
\end{aligned}
$$

Finally, a *Predicate* can be defined as a *TemporalPredicate*, i.e., a Boolean operation comparing periods or durations. *Period* comparisons based on Allen's Interval Algebra are supported by the functions defined in Section 7.3.1. *Duration* comparisons are done on simple scalars (e.g., ==, >, <, ≥, and ≤). A *Period* can be either created from some provided timestamps with the function *createPeriod* or computed as the global period of an element or set of elements with the function *globalPeriodOf*. Also, a *Period* can be constructed referring to a variable containing another period by means of an identifier. *Durations* can be obtained from existing periods (with the function *durationOf*) or created from specific durations in seconds, minutes, hours or days with the functions defined in Section 7.3.1.

$$
\begin{aligned}
\mathit{TemporalPredicate} &\triangleq \mathit{per_1}, \mathit{per_2} : \mathit{Period}; o : \mathit{Period\_Op} \mid\\
&\quad\; \mathit{dur_1}, \mathit{dur_2} : \mathit{Duration}; o : \mathit{Numerical\_Comp\_Op}\\
\mathit{Period} &\triangleq \mathit{PeriodCreation} \mid \mathit{PeriodVar}\\
\mathit{PeriodCreation} &\triangleq \mathit{ts_1}, \mathit{ts_2} : \mathit{Timestamp}; o : \mathit{createPeriod} \mid\\
&\quad\; \mathit{es} : \mathit{ElementSet}; o : globalPeriodOf\\
\mathit{PeriodVar} &\triangleq \mathit{identifier}\\
\mathit{Period\_Op} &\triangleq \mathit{before} \mid \mathit{after} \mid \mathit{meets} \mid \mathit{meetsInv} \mid \mathit{overlaps} \mid \mathit{overlapsInv} \mid\\
&\quad\; \mathit{starts} \mid \mathit{startsInv} \mid \mathit{during} \mid \mathit{duringInv} \mid \mathit{finishes} \mid\\
&\quad\; \mathit{finishesInv} \mid \mathit{matches}\\
\mathit{Duration} &\triangleq p : \mathit{Period}; o : \mathit{getDuration} \mid v : \mathit{Value}; o : \mathit{DurationOf}\\
\mathit{DurationOf} &\triangleq \mathit{Duration.ofSeconds} \mid \mathit{Duration.ofMinutes} \mid\\
&\quad\; \mathit{Duration.ofHours} \mid \mathit{Duration.ofDays}\\
\mathit{Numerical\_Comp\_Op} &\triangleq \;==\mid>=\mid<=\mid>\mid<
\end{aligned}
$$

7.3.2 Semantics

In this section, we make use of denotational semantics, as proposed in [81], to formally describe DAPOQ-Lang. We define several semantic functions M_T that describe the meaning of the nonterminal T (e.g., M_{Query} describes the meaning of the nonterminal $Query$). First, we introduce some notation that will be used in further definitions.

Definition 29 (overriding union) *The overriding union of $f : X \nrightarrow Y$ by $g : X \nrightarrow Y$, is denoted as $f \oplus g : X \nrightarrow Y$ such that $\mathit{dom}(f \oplus g) = \mathit{dom}(f) \cup \mathit{dom}(g)$ and:*

$$f \oplus g(x) = \begin{cases} g(x) & \text{if } x \in \mathit{dom}(g) \\ f(x) & \text{if } x \in \mathit{dom}(f) \setminus \mathit{dom}(g) \end{cases}$$

In the previous section, we introduced the use of variables in the language. These variables must be translated into a value in $MMSets$ (Eq. 7.2) during the execution of our queries. A *Binding* assigns a set of elements to a variable name:

$$\mathit{Binding} \triangleq \mathit{Varname} \rightarrow \mathit{MMSets}$$

Queries are computed based on a dataset complying with the structure of the OpenSLEX meta-model. Such a meta-model can be seen as a tuple of sets of elements of each of the basic types:

$$\mathit{MetaModel} \triangleq (AC,\ LG,\ EV,\ REL,\ OC,\ AT,\ CL,\ PER,\ PM,\ CS,\ AI,\ OV,\ RS,\ DM)$$

The meaning function of a query takes a query and a meta-model dataset as an input and returns a set of elements that satisfy the query:

$$M_{Query} : \mathit{Query} \times \mathit{MetaModel} \rightarrow \mathit{MMSets}$$

This function is defined as:

$$M_{Query}\left[q : \mathit{Query}, MM : \mathit{MetaModel}\right] \triangleq M_{ElementSet}(q.es, MM, M_{Assignments}(q.s, MM, \emptyset))$$

The evaluation of the meaning function of a query depends on the evaluation of the assignments and the element set involved. Evaluating the assignments involves resolving their corresponding element sets and remembering the variables to which they were assigned.

$$M_{Assignments} : \mathit{Assignments} \times \mathit{MetaModel} \times \mathit{Binding} \rightarrow \mathit{Binding}$$

A sequence of assignments resolves to a binding, which links sets of elements to variable names. Assignments that happen later in the order of declaration take precedence over earlier ones when they share the variable name.

```
$M_{Assignments}\left[s : Assignments,\ MM : MetaModel,\ B : Binding\right] \triangleq$
  **if** $\neg(s.TAIL).EMPTY$ **then**
    $M_{Assignments}(s.TAIL,\ MM,\ B \oplus M_{Assignment}(s.FIRST,\ MM,\ B))$
  **else** $B$
```

The result of an assignment is a binding, linking a set of elements to a variable name.

$$M_{Assignment} : Assignment \times MetaModel \times Binding \rightarrow Binding$$

$$M_{Assignment}\left[a : Assignment,\ MM : MetaModel,\ B : Binding\right] \triangleq \{(a.v,\ M_{ElementSet}(a.es,\ MM,\ B))\}$$

An *ElementSet* within the context of a meta-model and a binding returns a set of elements of the same type that satisfy the restrictions imposed in the *ElementSet.*

$$M_{ElementSet} : ElementSet \times MetaModel \times Binding \rightarrow MMSets$$

An *ElementSet* can be resolved as a *Construction* of other *ElementSets* with the well-known set operations union, exclusion, and intersection. It can be the result of evaluating an *Application* function, returning elements related to other elements, to the creation of *Periods*, or the value of a variable previously declared (*ElementSet-Var*). Also, it can be the result of a terminal *ElementSet*, e.g., the set of all the events (*allEvents*). Finally, an *ElementSet* can be the result of filtering another *ElementSet* according to a *PredicateBlock*, which is a *Predicate* preceded by a sequence of *Assignments.* These *Assignments* will only be valid within the scope of the *PredicateBlock*, and will not be propagated outside of it (i.e., if a variable is reassigned, it will maintain its original value outside of the *PredicateBlock*). The resulting *FilteredElementSet* will contain only the elements of the input *ElementSet* for which the evaluation of the provided *Predicate* is *True.*

```
$M_{ElementSet}[es : ElementSet,\ MM : MetaModel,\ B : Binding] \triangleq$
  case $es$ of
    $Construction \Rightarrow$
      case $es.o$ of
        $Union \Rightarrow M_{ElementSet}(es.es_1,\ MM,\ B) \cup M_{ElementSet}(es.es_2,\ MM,\ B)$
        $Excluding \Rightarrow M_{ElementSet}(es.es_1,\ MM,\ B) \setminus M_{ElementSet}(es.es_2,\ MM,\ B)$
        $Intersection \Rightarrow M_{ElementSet}(es.es_1,\ MM,\ B) \cap M_{ElementSet}(es.es_2,\ MM,\ B)$
      end
    $Application \Rightarrow es.o(M_{ElementSet}(es.es,\ MM,\ B))$
    $Period \Rightarrow M_{Period}(es,\ MM,\ B)$
    $ElementSetVar \Rightarrow \begin{cases} B(es) & \textbf{if } es \in dom(B) \\ \emptyset & \textbf{otherwise} \end{cases}$
    $ElementSetTerminal \Rightarrow es^{MM}$
    $FilteredElementSet \Rightarrow \{e \in M_{ElementSet}(es.es,\ MM,\ B) \mid$
                $M_{Predicate}(es.pb.p,\ MM,\ M_{Assignments}(es.pb.s,\ MM,\ B \oplus (it,\ e)))\}$
  end
```

A *Predicate* is evaluated as a Boolean, with respect to a *MetaModel* and a *Binding*:

$$M_{Predicate} : Predicate \times MetaModel \times Binding \rightarrow \mathbb{B}$$

The meaning function of *Predicate* ($M_{Predicate}$) evaluates the Boolean value of a *Predicate*, which can be recursively constructed combining binary (*and*, *or*) or unary (*not*) predicates. Also, *Predicates* can be defined as *AttributePredicates* that evaluate the existence of attributes, comparisons of attribute values, or attribute value changes. Finally, they can be defined as *TemporalPredicates*, which can compare durations or periods by means of Allen's Interval Algebra operators.

```
$M_{Predicate}[p : Predicate,\ MM : MetaModel,\ B : Binding] \triangleq$
  case $p$ of
    $AttributePredicate \Rightarrow$
      case $p$ of
        $AttributeExists \Rightarrow$  if $B(it) \in EV : eventHasAttribute(p.at,\ B(it))$
                            elif $B(it) \in OV : versionHasAttribute(p.at,\ B(it))$
                            elif $B(it) \in CS : caseHasAttribute(p.at,\ B(it))$
                            elif $B(it) \in LG : logHasAttribute(p.at,\ B(it))$
                            else : $\varnothing$
        $AttributeValuePred \Rightarrow$
          $p.o(M_{Attribute}(p.at_1,\ B(it),\ MM),\ M_{Attribute}(p.at_2,\ B(it),\ MM))$
        $AttributeChange \Rightarrow$
          if $B(it) \in MM.OV$ then : $versionChange(p.at,\ p.from,\ p.to,\ B(it))$ else : $\varnothing$
      end
    $Un\_Predicate \Rightarrow \neg M_{Predicate}(p.p,\ MM,\ B)$
    $Bin\_Predicate \Rightarrow$
      case $p.o$ of
        $And \Rightarrow M_{predicate}(p.p_1,\ MM,\ B) \wedge M_{predicate}(p.p_2,\ MM,\ B)$
        $Or \Rightarrow M_{predicate}(p.p_1,\ MM,\ B) \vee M_{predicate}(p.p_2,\ MM,\ B)$
      end
    $TemporalPredicate \Rightarrow$
      case $p.o$ of
        $Period\_Op \Rightarrow p.o(M_{Period}(p.per_1,\ MM,\ B), M_{Period}(p.per_2,\ MM,\ B))$
        $Duration\_Op \Rightarrow p.o(M_{Duration}(p.dur_1,\ MM,\ B), M_{Duration}(p.dur_2,\ MM,\ B))$
      end
  end
```

A *Period* for a given meta-model dataset and a binding returns an instance of

PER, i.e., a single period element:

$$M_{Period} : Period \times MetaModel \times Binding \rightarrow PER$$

The meaning function of *Period* will return a period element that can be created (*PeriodCreation*) or assigned from a variable name containing a period (*PeriodVar*). In the case of a *PeriodCreation*, a period can be created for the specified start and end timestamps using the *createPeriod* function or it can be computed as the global period of another set of periods (*globalPeriodOf*).

```
$M_{Period}[p : Period,\ MM : MetaModel,\ B : Binding] \triangleq$
  **case** $p$ **of**
    $PeriodCreation \Rightarrow$
      **case** $p.o$ **of**
        $createPeriod \Rightarrow p.o(p.ts_1,\ p.ts_2)$
        $globalPeriodOf \Rightarrow p.o^{MM}(M_{ElementSet}(p.es,\ MM,\ B))$
      **end**
    $PeriodVar \Rightarrow \begin{cases} B(p) & \textbf{if } p \in dom(B) \\ \emptyset & \textbf{otherwise} \end{cases}$
  **end**
```

A *Duration* is simply a value representing the length of a period, and it is computed within the context of a meta-model dataset and a binding:

$$M_{Duration} : Duration \times MetaModel \times Binding \rightarrow DUR$$

A *Duration* can be evaluated based on the duration of a period (*getDuration*), or a duration specified in scalar units (*DurationOf*).

```
$M_{Duration}[d : Duration,\ MM : MetaModel,\ B : Binding] \triangleq$
  **case** $d.o$ **of**
    $DurationOf \Rightarrow d.o(d.v)$
    $getDuration \Rightarrow d.o(M_{Period}(d.p,\ MM,\ B))$
  **end**
```

Finally, an *Attribute* is a value assigned to an element in the context of a meta-model:

$$M_{Attribute} : Attribute \times Element \times MetaModel \rightarrow Value$$

In order to evaluate the value of an *Attribute*, we can refer to the *AttributeName*, in which case the value will be obtained in different ways depending on the type of element (event, object version, case, or log). Also, an *Attribute* can be explicitly defined by its *Value*.

```
$M_{Attribute}[at : Attribute,\ e : Element,\ MM : MetaModel] \triangleq$
  case $at$ of
    $AttributeName \Rightarrow$
      case $e$ of
        $Event \Rightarrow$ if $eventHasAttribute(at,\ e)$ then : $getAttributeEvent(at,\ e)$ else : $\emptyset$
        $Version \Rightarrow$ if $versionHasAttribute(at,\ e)$ then : $getAttributeVersion(at,\ e)$ else : $\emptyset$
        $Case \Rightarrow$ if $caseHasAttribute(at,\ e)$ then : $getAttributeCase(at,\ e)$ else : $\emptyset$
        $Log \Rightarrow$ if $logHasAttribute(at,\ e)$ then : $getAttributeLog(at,\ e)$ else : $\emptyset$
      end
    $Value \Rightarrow at$
  end
```

Figure 7.4 shows the syntax tree of Query 7.1 according to the presented abstract syntax. This concludes the formal definition of DAPOQ-Lang in terms of syntax and semantics at an abstract level. The later sections provide some details about the specific syntax, implementation, and its performance.

7.4 Implementation & Evaluation

DAPOQ-Lang[2] has been implemented as a Domain-Specific Language (DSL) on top of Groovy[3], a dynamic language for the Java platform. This means that, on top of all the functions and operators provided by DAPOQ-Lang, *any syntax allowed by Groovy or Java can be used within DAPOQ-Lang queries.* DAPOQ-Lang heavily relies on a Java implementation of the OpenSLEX[4] meta-model using SQLite[5] as a storage and querying engine. However, DAPOQ-Lang abstracts from the specific storage choice, which allows it to run on any SQL database and not just SQLite. We have developed the platform PADAS[6] (Process Aware Data Suite), which integrates DAPOQ-Lang and OpenSLEX in a user-friendly environment to process the data and run queries. The current implementation relies on the SQLite library to store the data and execute certain subqueries. Therefore, it is to be expected that DAPOQ-Lang introduces certain overhead, given that data retrieval and object creation on the client side consume extra time and memory compared to an equivalent SQL query.

In order to assess the impact of DAPOQ-Lang on query performance, we conducted the following experiment. We run a benchmark of pairs of equivalent queries, as

[2]`https://github.com/edugonza/DAPOQ-Lang/`
[3]`http://groovy-lang.org/`
[4]`https://github.com/edugonza/OpenSLEX/`
[5]`https://www.sqlite.org`
[6]`https://github.com/edugonza/PADAS/`

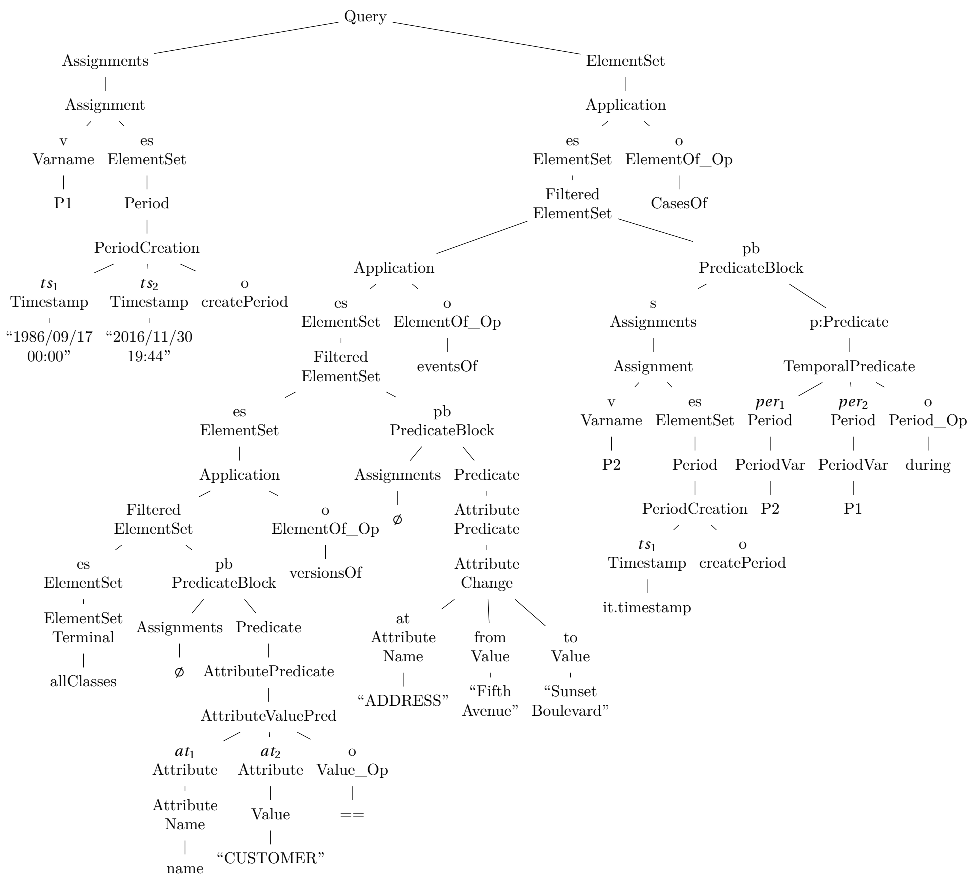


Figure 7.4: Syntax tree for DAPOQ-Lang Query 7.1.

expressed in DAPOQ-Lang and SQL, on the same SQLite database. The queries are organized in 3 categories and run on the 3 datasets described in [42]: dataset A (from the redo-logs of a simulated ticket selling platform), dataset B (from a financial organization) and dataset C (a sample from a SAP system) (Table 7.3).

The DAPOQ-Lang queries of each pair were run with two different configurations: memory-based and disk-based caching. *Memory-based caching* uses the heap to store all the elements retrieved from the database during the execution of the query. This is good for speed when dealing with small or medium size datasets, but represents a big limitation to deal with big datasets given the impact on memory use and garbage collection overhead. *Disk-based caching* makes use of MapDB[7], a disk-based implementation of Java hash maps, to serialize and store in disk all the elements retrieved from the database. This significantly reduces the memory consumption and allows us

[7]`http://www.mapdb.org`

Table 7.3: Size of datasets.

Dataset	Objects	Versions	Events	Cases	Logs	Activities
A	6740	8424	8512	108 751	34	14
B	7 339 985	7 340 650	26 106	82 113	10 622	172
C	162 287	277 094	277 094	569 026	29	62

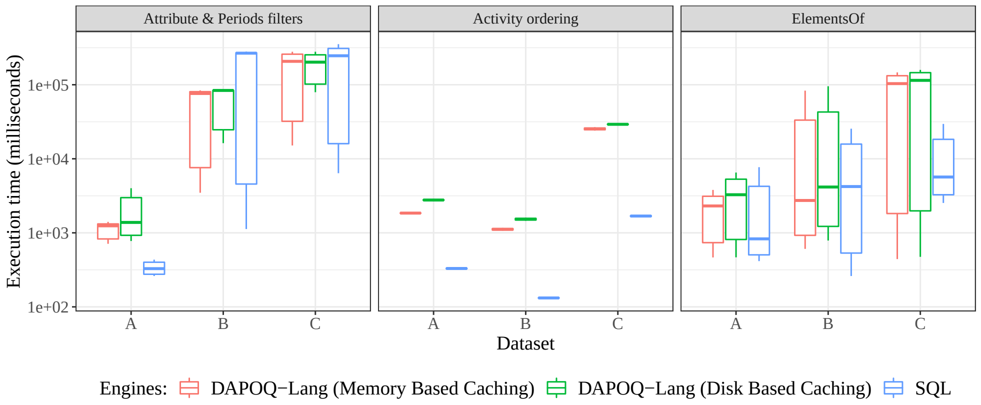


Figure 7.5: Benchmark of queries run with DAPOQ-Lang, DAPOQ-Lang with disk-based caching, and SQL on an SQLite backend. Note that the vertical axis is logarithmic.

to handle much larger datasets, which comes at the cost of speed given the overhead introduced by serialization and disk I/O. Figure 7.5 shows the results of the benchmark, with one graph per query type, one line per query engine (SQL, DAPOQ-Lang, and DAPOQ-Lang with disk caching), and for the three datasets ordered by size on the horizontal axis. Analyzing the results we see that the disk-based implementation of DAPOQ-Lang is slower than the memory-based for every dataset, which is due to the I/O factor. Also, the largest absolute difference in execution time is observed when evaluating the *attribute & periods filters* queries on dataset B. In this case, DAPOQ-Lang-based engines perform ten times faster than SQL, with a difference of around 450 seconds. This can be due to certain optimizations introduced in DAPOQ-Lang, designed to take advantage of the OpenSLEX structure, transforming complex DAPOQ-Lang queries into smaller and simpler SQL queries, which execute faster than an equivalent monolithic SQL query. In the rest of the tests, we can observe that the performance of DAPOQ-Lang queries is either similar, or poorer than the one of the SQL engine. In the worst cases observed, DAPOQ-Lang-based engines are one order of magnitude slower than the SQL engine, especially when it comes to queries regarding the order of activities. This is due to the overhead on transmission and processing of data and the fact that many filtering operations are performed on the client instead of the server side. Obviously, there is a trade-off between ease-of-use and performance. Nevertheless, performance was never the main motivation for the development of DAPOQ-Lang, but the ease of use and speed of query writing. In future versions, further efforts will be made to improve performance and to provide more exhaustive benchmarks.

7.5 Application / Use Cases

The purpose of this section is to demonstrate the applicability of our approach and tools. First, we explore the professional profiles to which this language is directed to and we identify the most common data aspects to query given each profile. Then, we provide some use cases of DAPOQ-Lang with examples of relevant queries for each data aspect. Finally, we compare DAPOQ-Lang to SQL by means of an example. The example highlights the expressiveness and compactness of our query language in the context of process mining.

7.5.1 Business Questions in Process Mining

Process mining is a broad field, with many techniques available tailored towards a variety of analysis questions. "Process miners" are often interested in discovering process models from event data. Sometimes, these models are provided beforehand and the focus is on conformance. It can be the case that assessing the performance of specific activities is critical. Also, finding out bottlenecks in the process can be of interest for the analysts. In some contexts, where existing regulations and guidelines impose restrictions on what is allowed and what is not in the execution of a process, compliance checking becomes a priority. In existing literature, we can find examples of frequently posed questions for specific domains, like healthcare [77], in which root cause analysis becomes relevant in order to trace back data related to a problematic case. All these perspectives pose different challenges to process miners, who need to dig into the data to find answers to relevant questions.

Previous works [60, 86] have tried to identify professional roles and profiles in the area of business process management by analyzing job advertisements and creating a classification based on the expected competencies. We make use of this classification to point out data aspects relevant for each profile. Table 7.4 presents, in the two leftmost columns, the classification of roles according to the authors of the study. In the column *Main Focus* we propose, based on the role description, the sub-disciplines of process mining and data engineering that become relevant for each job profile (i.e., discovery, compliance checking, conformance checking, performance analysis, root cause analysis, integration, and data integrity). The rest of the columns indicate whether certain event data aspects become particularly interesting to be queried for each professional role, considering the role description and the main focus. We have grouped these event data aspects into two big categories that reflect the expected output of the queries:

1. *Specialized sublogs* are event logs that contain a subset of the events of another event log. This selected subset reflects certain desired properties (e.g., temporal

Table 7.4: Types of BPM professionals, according to [86], and relation to querying in process mining.

			Specialized Sublogs					**Metrics, Artifacts & Provenance**		
Role [86]	**Description** [86]	**Main focus**	**Temporal constraints**	**Activity occurrence**	**Order of actions**	**Data properties**	**Data changes**	**Data lineage**	**Dependency relations**	**Performance metrics**
Business Process Analyst	Elicits, analyses, documents and communicates user requirements and designs according to business processes and IT systems; acts as a liaison between business and IT	Discovery, Compliance & Conformance	✓	✓	✓	✓	✓			
Business Process Compliance Manager	Analyses regulatory requirements and ensures compliance of business processes and IT systems	Compliance & Conformance	✓	✓	✓	✓	✓			
Business Process Manager Sales & Marketing	Designs sales processes and analyses requirements for related IT systems; supports and executes sales and marketing processes	Compliance	✓	✓	✓	✓	✓			
Business Process Improvement Manager	Analyses, measures and continuously improves business process, e.g. through the application of Lean or Six Sigma management techniques	Performance, Conformance & Root Cause Analysis	✓	✓	✓	✓	✓	✓	✓	✓
ERP Solution Architect	Implements business processes in ERP systems	Performance, Conformance & Root Cause Analysis	✓	✓	✓	✓	✓	✓	✓	✓
IT-Business Strategy Manager	Aligns business and IT strategies; monitors technological innovations and identifies business opportunities	Performance & Conformance	✓	✓	✓	✓	✓			✓
Technical Architect	Develops and integrates hardware and software infrastructures	Integration & Data Integrity				✓	✓	✓	✓	✓

constraints, activity occurrence constraints),

2. *Metrics, artifacts and provenance* are the resulting values of the computation of certain event data properties (e.g., performance metrics).

We see that there is a clear distinction between roles interested in performance and root cause analysis, in contrast to those mainly interested in compliance. The former will need to obtain *performance metrics* from the data, e.g., average case duration, most time-consuming tasks, etc. Also, they will be interested in finding data related to problematic cases, e.g., obtaining all the products purchased in an unpaid order (*dependency relations*), or finding out providers of a defective batch of products (*data lineage*). However, those with a focus on compliance will typically need to answer questions related to *temporal constraints* (e.g., if cases of a particular type of client are resolved within the agreed SLAs), *activity occurrence* constraints (e.g., whether a purchase was always paid), and *order of actions* (e.g., if an invoice is created before a delivery is dispatched).

We see that, as the roles get more concerned with the technical aspects of IT systems, more focus is put on performance and data properties. Especially for technical architects, data integrity is crucial, since they are the ones in charge of integrating both applications and data systems. Being able to filter information based on *data properties* and find irregular *data changes* is important to verify a correct integration of different infrastructures.

Now that we have identified data aspects of interest, we present a set of example DAPOQ-Lang queries. The aim of these examples is two-fold: to serve as a template to write queries with a similar purpose, as well as to demonstrate that the features of DAPOQ-Lang indeed cover all the aspects described in Table 7.4.

7.5.2 Exporting Logs

One of the main purposes when querying process execution data is to export it as a compatible event log format. DAPOQ-Lang provides utilities to export logs, cases, and events as XES event logs, which can be further analyzed using process mining platforms such as ProM[8] or RapidProM[9]. The following queries show the way to export XES logs for different types of data. When a set of logs is retrieved, an independent XES log is generated for each of them.

Query 7.3: Export all the logs with a specific name. The result can be one or many logs being exported according to the XES format.

```
exportXLogsOf(allLogs().where{ name == "log01" })
```

When the input of *exportXLogsOf* is a set of cases, one single XES log is exported.

Query 7.4: Export in a single XES log all the cases of different logs.

```
exportXLogsOf(casesOf(allLogs().where { name.contains("1") }))
```

In the case of a set of events, a single XES log with a single trace is exported.

Query 7.5: Export in a single XES log all the events of different logs.

```
exportXLogsOf(eventsOf(allLogs().where{ name.contains("1") }))
```

A special situation is when we want to export a set of logs or cases while filtering out events that do not comply with some criteria. In that case, we call *exportXLogsOf* with a second argument representing the set of events that can be exported. Any event belonging to the log to be exported not contained in this set of events will be excluded from the final XES log.

[8]http://www.promtools.org
[9]http://www.rapidprom.org/

Query 7.6: Export one or many XES logs excluding all the events that do not belong to a specific subset.

```
exportXLogsOf(allLogs(), eventsOf(allClasses().where { name == "BOOKING"}))
```

7.5.3 Specialized Sublogs

So far, we have seen how to export logs as they are stored in the dataset under analysis. However, it is very common to focus on specific aspects of the data depending on the questions to answer. This means that we need to create specialized sublogs according to certain criteria. This section presents examples of queries to create specialized sublogs that comply with certain constraints in terms of *temporal* properties, *activity occurrence*, *order of action*, *data properties*, or *data changes*.

Temporal constraints

A way to create specialized sublogs is to filter event data based on temporal constraints. The creation and computation of periods makes it possible to select only data relevant during a certain time span. Query 7.7 returns events that happened during period p and that belong to the log "log01".

Query 7.7: Retrieve all the events of "log01" that happened during a certain period of time.

```
def evLog01 = eventsOf(allLogs().where{ name == "log01" })
def p = createPeriod("2014/11/27 15:57","2014/11/27 16:00", "yyyy/MM/dd HH:mm")

eventsOf(p).intersection(evLog01)
```

Query 7.8 focuses on the duration of cases rather than on the specific time when they happened. Only cases of log "log01" with a duration longer than 11 minutes will be returned.

Query 7.8: Retrieve cases of "log01" with a duration longer than 11 minutes. The variable "it" is used to iterate over all values of "c" within the "where" closure

```
def c = casesOf(allLogs().where{ name == "log01" })

c.where { globalPeriodOf(it).getDuration() > Duration.ofMinutes(11) }
```

Activity occurrence

Another way to select data is based on activity occurrence. The following query shows an example of how to retrieve cases in which two specific activities were performed regardless of the order. First, cases that include the first activity are retrieved (*casesA*). Then, this happens for cases that include the second activity (*casesB*). Finally, the intersection of both sets of cases is returned.

Query 7.9: Retrieve cases where activities that contain the words “INSERT” or “UPDATE” and “CUSTOMER” happened in the same case.

```
def actA = allActivities().where {
  name.contains("INSERT") && name.contains("CUSTOMER") }

def actB = allActivities().where {
  name.contains("UPDATE") && name.contains("CUSTOMER") }

def casesA = casesOf(actA)
def casesB = casesOf(actB)

casesA.intersection(casesB)
```

Order of actions

This time we are interested in cases in which the relevant activities happened in a specific order. The following query, an extended version of Query 7.9, selects the cases that include both activities. Yet, before storing the intersection of cases containing events of the activities in the set *actA* with cases containing events of the activities in the set *actB* in a variable (line 13), the query performs a filter based on the order of these two activities. For each case, the set of events is retrieved (line 15). Next, the events of the first activity are selected (line 16) and the ones of the second activity (line 17). Finally, the periods of both events are compared (line 18), evaluating the condition to the value *True* for each case in which all the events of activity *A* happened *before* the events of activity B. Only the cases for which the condition block (lines 14 to 18) evaluated to *True* are stored in the variable *casesAB* and returned.

Query 7.10: Retrieve cases where activities that contain the words “INSERT” and “CUSTOMER” happen before activities that contain the words “UPDATE” and “CUSTOMER”.

```
 1 def actA = allActivities().where {
 2   name.contains("INSERT") && name.contains("CUSTOMER") }
 3
 4 def actB = allActivities().where {
 5   name.contains("UPDATE") && name.contains("CUSTOMER") }
 6
 7 def casesA = casesOf(actA)
 8 def casesB = casesOf(actB)
 9
10 def eventsA = eventsOf(actA)
11 def eventsB = eventsOf(actB)
12
13 casesAB = casesA.intersection(casesB)
14 .where {
15   def ev = eventsOf(it)
16   def evA = ev.intersection(eventsA)
17   def evB = ev.intersection(eventsB)
18   before(globalPeriodOf(evA),globalPeriodOf(evB))
19 }
```

Data properties

Some elements in our OpenSLEX dataset contain attributes that can be queried. These elements are object versions, events, cases, and logs. The following query shows how to filter events based on their attributes. First, the query compares the value of the attribute *resource* to a constant. Also, it checks if the attribute *ADDRESS* contains a certain substring. Finally, it verifies that the event contains the attribute *CONCERT_DATE*. Only events that satisfy the first and either the second or the third will be returned as a result of the query.

Query 7.11: Retrieve events of resource “SAMPLE” that either have an attribute *ADDRESS* which value contains “35” or have a *CONCERT_DATE* attribute.

```
allEvents().where {
 resource == "SAMPLEDB" && (at.ADDRESS.contains("35") || has(at.CONCERT_DATE))}
```

Data changes

An important feature of our query language is the function *changed*. This function determines if the value of an attribute for a certain object version changed. The function has the attribute name as a required parameter (*at:*) and two optional parameters (*from:*, and *to:*). Query 7.12 returns all the events related to object versions for which the value of the attribute “BOOKING_ID” changed. No restrictions are set on the specific values. Therefore, the call to *changed* will be evaluated to *True* for an object version only if the value of the attribute in the preceding version was different from the value in the current one.

Query 7.12: Retrieve events that affected versions where the value of “BOOKING_ID” changed.

```
eventsOf(allVersions().where { changed([at: "BOOKING_ID"]) })
```

Query 7.13 shows a similar example. This time we want to obtain the events related to object versions for which the attribute “SCHEDULED_DATE” changed from “11-JUN-82”, to a different one.

Query 7.13: Retrieve events that affected versions where the value of “SCHEDULED_-DATE” changed from “11-JUN-82” to a different value.

```
eventsOf(allVersions().where { changed([at: "SCHEDULED_DATE", from: "11-JUN-82"])})
```

Query 7.14 instead retrieves the events related to object versions for which the attribute “SCHEDULED_DATE” changed to “22-MAY-73”, from a different one.

Query 7.14: Retrieve events that affected versions where the value of “SCHEDULED_-DATE” changed to “22-MAY-73” from a different value.

```
eventsOf(allVersions().where { changed([at: "SCHEDULED_DATE", to: "22-MAY-73"]) })
```

Finally, Query 7.15 imposes a stricter restriction, retrieving only the events related to object versions for which the attribute "SCHEDULED_DATE" changed from "24-MAR-98", to "22-MAY-73".

Query 7.15: Retrieve events that affected versions where the value of "SCHEDULED_-DATE" changed from "24-MAR-98" to "22-MAY-73".

```
eventsOf(allVersions().where {
  changed([at: "SCHEDULED_DATE", from: "24-MAR-98", to: "22-MAY-73"]) })
```

7.5.4 Metrics, Artifacts & Provenance

In the previous section, we have seen examples of how to obtain specialized sublogs given certain criteria. However, we do not always want to obtain events, cases, or logs as a result of our queries. In certain situations, the interest is in data objects, and their relations to other elements of the dataset, e.g., objects of a certain type, artifacts that coexisted during a given period, data linked to other elements, etc. Also, we can be interested in obtaining performance metrics based on existing execution data. All these elements cannot be exported as an event log, since they do not always represent event data. However, they can be linked to related events or traces. This section shows example queries that exploit these relations and provide results that cannot be obtained as plain event logs.

Data lineage

Data lineage focuses on the life-cycle of data, its origins, and where it is used over time. DAPOQ-Lang supports data lineage mainly with the *ElementsOf* functions listed in Section 7.3.1. These functions return elements of a certain type linked or related to input elements of another type. As an example, we may have an interest in obtaining all the products in the database affected by a catalog update process during a certain period in which prices were wrongly set. Query 7.16 finds the cases of log "log01" whose lifespan overlaps with a certain period, and returns the object versions related to them.

Query 7.16: Retrieves versions of objects affected by any case in "log01" whose lifespan overlapped with a certain period of time. A date template is specified.

```
def P1 = createPeriod("2014/11/27 15:56","2014/11/27 16:30","yyyy/MM/dd HH:mm")

versionsOf(
  casesOf(allLogs().where{name=="log01"})
  .where {
    overlaps(globalPeriodOf(it),P1)
  }
)
```

Dependency relations

An important feature of the language is the ability to query existing relations between elements of different types, as well as within object versions of different classes. Query 7.16 showed an example of relations between elements of different types (logs to cases, cases to versions). The following query shows an example of a query on object versions related to other object versions. First, two different classes of the data model are obtained (lines 1 and 2). Then, the versions of the class “TICKET” are retrieved (line 3). Finally, the object versions related to object versions belonging to class “BOOKING” are obtained (lines 5 and 6) and only the ones belonging to class “TICKET” are selected (line 7).

Query 7.17: Retrieve versions of ticket objects that are related to versions of booking objects.

```
1 def ticketClass = allClasses().where{ name == "TICKET"}
2 def bookingClass = allClasses().where{ name == "BOOKING"}
3 def ticketVersions = versionsOf(ticketClass)
4
5 versionsRelatedTo(
6   versionsOf(bookingClass)
7 ).intersection(ticketVersions)
```

Performance metrics

As has been previously discussed, measuring performance and obtaining metrics for specific cases or activities is a very common and relevant question for many professional roles. The way DAPOQ-Lang supports this aspect is with the computation of periods and durations to measure performance. The resulting periods can be externally used to compute performance statistics such as average execution time or maximum waiting time. The following query shows how to compute periods for a subset of the events in the dataset.

Query 7.18: Retrieve periods of events belonging to activities that contain the words “UPDATE” and “CONCERT” in their name.

```
def actUpdateConcert = allActivities().where {
    name.contains("UPDATE") && name.contains("CONCERT")
  }

periodsOf(eventsOf(actUpdateConcert))
```

Query 7.19 demonstrates how to filter out periods based on their duration. Cases with events executed by a certain resource are selected and their periods are computed. Next, only periods with a duration longer than 11 minutes are returned.

Query 7.19: Retrieve periods of a duration longer than 11 minutes computed on cases which had at least one event executed by the resource “SAMPLEDB”.

```
def c = casesOf(allEvents().where { resource == "SAMPLEDB" })

periodsOf(c).where { it.getDuration() > Duration.ofMinutes(11) }
```

7.5.5 DAPOQ-Lang vs. SQL

So far, we have seen several examples of *toy* queries to demonstrate the use of the functions and operators provided by DAPOQ-Lang. Obviously, there is nothing in DAPOQ-Lang that cannot be computed with other Turing-complete languages. When it comes to data querying on databases, SQL is the undisputed reference. It is the common language to interact with most of the relational database implementations available today. It is a widespread language, known by many professionals from different fields. Even without considering scripting languages like PL/SQL, the combination of SQL with CTE (Common Table Expressions) and Windowing has been proven to be Turing-complete [35]. Other programming languages like Python[10] and R[11] are receiving a lot of attention in the data science scene for their ease of use, expressive power, and data manipulation capabilities. In the case of Python, despite not being designed as a query language, libraries like Pandas[12] provide powerful data manipulation and filtering capabilities similar to the ones included in R. However, none of these options is oriented to a process mining use case. The aim of DAPOQ-Lang is not to enable new types of computations, but to ease the task of writing queries in the specific domain of process mining. Our focus is on the ease of use, but not at the cost of expressiveness.

Let us consider the generic question (GQ) presented in Section 7.1:

> **GQ:** In which cases there was (a) an event that happened between time T1 and T2, (b) that performed a modification in a version of class C (c) in which the value of field F changed from X to Y?

This query involves several types of elements: cases, events, object versions, and attributes. We instantiate this query with some specific values for T1 = “1986/09/17 00:00”, T2 = “2016/11/30 19:44”, C = “CUSTOMER”, F = “ADDRESS”, X = “Fifth Avenue”, and Y = “Sunset Boulevard”. Assuming that our database already complies with the structure proposed by the OpenSLEX meta-model, we can write the SQL query in Query 7.20 to answer the question.

The logic is not hard to follow. Two subqueries are nested in order to retrieve (a) object versions preceding another object version (lines 31-38) and object versions that contain the attribute that changed (lines 20-39). Parts of the query focus on checking the value of the attributes (lines 25-28), the timestamp of the events (lines 15-16), and the class of the object versions (lines 7-9). The rest of the query is concerned with joining rows of different tables by means of foreign keys.

The equivalent DAPOQ-Lang query, previously presented in Query 7.1, removes most of the clutter and boilerplate code in order to join tables together and lets the user focus on the definition of the constraints. The query is built up with an assignment and several nested queries. First, a period of time is defined (line 1). Then, object versions of a certain class are retrieved (lines 5–6) and filtered based on the changes of one of the attributes (line 7). Next, the events related to such object

[10] `https://www.python.org/`

[11] `https://www.r-project.org/`

[12] Python Data Analysis Library: `https://pandas.pydata.org/`

Query 7.20: Standard SQL Query executed on the Redo-Log populated meta-model in [42] and equivalent to the DAPOQ-Lang Query 7.1.

```
 1 SELECT distinct C.id as "id", CAT.name, CATV.value, CATV.type
 2 FROM "case" as C
 3 JOIN activity_instance_to_case as AITC ON AITC.case_id = C.id
 4 JOIN activity_instance as AI ON AI.id = AITC.activity_instance_id
 5 JOIN event as E ON E.activity_instance_id = AI.id
 6 JOIN event_to_object_version as ETOV ON ETOV.event_id = E.id
 7 JOIN object_version as OV ON ETOV.object_version_id = OV.id
 8 JOIN object as O ON OV.object_id = O.id
 9 JOIN class as CL ON O.class_id = CL.id AND CL.name = "CUSTOMER"
10 JOIN attribute_name as AT ON AT.name = "ADDRESS"
11 JOIN attribute_value as AV ON AV.attribute_name_id = AT.id AND AV.object_version_id = OV.id
12 LEFT JOIN case_attribute_value as CATV ON CATV.case_id = C.id
13 LEFT JOIN case_attribute_name as CAT ON CAT.id = CATV.case_attribute_name_id
14 WHERE
15   E.timestamp > "527292000000" AND
16   E.timestamp < "1480531444303" AND
17   AV.value LIKE "Sunset Boulevard" AND
18   EXISTS
19   (
20     SELECT OVP.id
21     FROM
22       object_version as OVP,
23       attribute_value as AVP
24     WHERE
25       AVP.attribute_name_id = AT.id AND
26       AVP.object_version_id = OVP.id AND
27       OVP.object_id = OV.object_id AND
28       AVP.value LIKE "Fifth Avenue" AND
29       OVP.id IN
30       (
31         SELECT OVPP.id
32         FROM object_version as OVPP
33         WHERE
34           OVPP.end_timestamp <= OV.start_timestamp AND
35           OVPP.end_timestamp >= 0 AND
36           OVPP.object_id = OV.object_id AND
37           OVPP.id != OV.id
38         ORDER BY OVPP.end_timestamp DESC LIMIT 1
39       )
40   )
```

versions are obtained (lines 4–8) and filtered, based on the time when they occurred (lines 8–12). Finally, the cases of these events are returned (lines 3–13). Figure 7.6 shows the discovered process model by the Inductive Miner [63] from the exported sublog, where we can observe that insertions of new customers are followed by updates that, in our sublog, modify the address attribute.

In essence, the advantage of DAPOQ-Lang over SQL is on the ease of use in the domain of process mining. The fact that we can assume how logs, cases, events and objects are linked, allows us to focus on the important parts of the query. Also, providing functions that implement the most frequent operations on data (such as period and duration computation) makes writing queries faster and less prone to errors. This comes at the cost of a slight decrease in performance, mainly due to implementation choices. Nevertheless, we believe that the ability to express complicated queries with

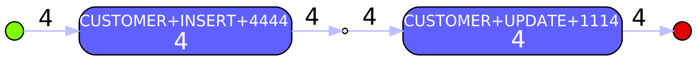


Figure 7.6: Process model corresponding to 4 traces selected by Query 7.1

ease overweights the burden of longer execution times.

7.6 Chapter Summary

In the field of process mining, the need for better querying mechanisms has been identified. This chapter presented a systematic literature review of process data querying. Next, we proposed a method to combine both process and data perspectives in the scope of process querying, helping with the task of obtaining insights about processes. DAPOQ-Lang, a Data-Aware Process Oriented Query Language, has been developed, which allows the analyst to select relevant parts of the data in a simple way to, among other things, generate specialized event logs to answer meaningful business questions. We have formally described the syntax and semantics of the language. We presented its application by means of simple use cases and query examples in order to show its usefulness and simplicity. In addition, we provide an efficient implementation that not only enables the execution, but also the fast development of queries. This chapter shows that it is feasible to develop a query language that satisfies the most common needs of process mining analysts. Nevertheless, DAPOQ-Lang presents certain limitations in terms of performance, expressiveness, and ease of use. As future work, efforts will be made on (a) expanding the language with new functionalities and constructs relevant in the process mining context, (b) improving the query planning and execution steps in order to achieve better performance, and (c) carrying out an empirical evaluation with users in order to objectively assess the suitability of the language within the process mining domain.

Cotton netting veil with silk tulle front.
"Beekeeping: a discussion of the life of the honeybee and of the production of honey", Everett Franklin Phillips, 1923

8 Case Study: Process Mining on a Health Information System

In this thesis, we have presented several techniques focused on the data preparation for process mining analysis. Our main purpose is to assist the analyst during the initial phases of a process mining project by reducing the time needed to obtain an event log. Also, we provide the tools to speed up the execution of each of the steps, from data extraction to event log creation. We believe that an effective way to show the feasibility of a technique is to test it in a real-life environment. For this purpose, we approached a Dutch ICT organization focused on the deployment and maintenance of Health Information Systems (HISs) to obtain real-life data. In fact, this organization is a distributor of the main HIS software within The Netherlands. We got access to a large database of one real instance of the HIS to test our methodology. In this chapter, we describe the steps followed to extract data and build event logs suitable for process mining analysis using the techniques presented in this thesis.

8.1 Introduction

The case study described in the present chapter was carried out following the premise of *automating as many steps as possible.* Traditionally, when initiating a process mining project, a lot of time is invested in acquiring enough domain knowledge in order to understand the data schema, tables, and relations at hand, and to map them to the main underlying business processes. We acknowledge the importance of domain knowledge to interpret results and to be able to make the right decisions. However, our contributions aim at enabling analysts to carry out the data extraction and log-building phases with less domain knowledge. Therefore, this case study was envisioned as a real-life test for the techniques and methods proposed in this thesis. Only a short meeting was held with the business owners in order to set up the project and initiate the extraction. During this meeting, we obtained a general overview of the challenges to overcome when dealing with these data, as well as the technical aspects in order to enable the extraction. The main points were:

- **Size**. The data schema is large: it involves more than 3000 tables. Therefore, it is not feasible to extract all of it with manual methods.
- **Incompleteness**. The data schema is incomplete, i.e., tables and columns are defined, but primary and foreign key definitions are missing. These keys are managed at the application level. Given that we are dealing with a proprietary software application, it is not possible to obtain a complete specification of the data schema from documentation or code. In order to correlate database objects (rows), we need to discover primary and foreign keys.
- **Privacy**. Due to the fact that we are handling real medical data, we must handle the raw information within the private network and systems of the database owner, not being allowed to extract it or process it locally.

These aforementioned challenges give an idea of the complexity of the setting at hand. However, this is not an extremely rare environment. In fact, when tackling a real-life dataset, it is quite common to face issues related to data incompleteness, privacy, lack of documentation, or data dimensionality. Therefore, we consider this as a good *fire test* in order to evaluate the proposed techniques, and to identify new challenges and opportunities for improvement. The following section describes in detail the steps followed to carry out this study in practice.

8.2 From Database to Event Log in Six Commands

In order to demonstrate the feasibility of our approach, we aim at obtaining a complete view of the database at hand and to build meaningful event logs. First, we must overcome some challenges and follow a set of steps. Each of these steps require using very specific techniques, most of which have been presented in the previous chapters of this thesis. However, when dealing with this dataset, we faced some challenges for which we did not have a solution in place yet. Therefore, we developed additional general solutions that could be used to solve similar issues in other scenarios. We propose a list of steps to follow, each of them implemented as a single command. Below you can find a list of the steps and a description of their purpose:

1. **Data exploration**: to get a feeling of the size and dimension of the data. Also, to look for any high-level structure that can be extracted from it.

2. **Data schema discovery**: to discover the data relations (PKs, UKs, and FKs) in order to be able to correlate data objects in future steps.

3. **Data extraction**: to obtain an off-line copy of the data that we can transform into a format suitable for analysis. Also, this allows us to complete the data once a schema has been discovered.

4. **Event data discovery**: event data might be implicitly stored within or across different tables in the dataset. We need to discover the events and make them explicit.

5. **Case notion discovery**: defining a case notion allows us to correlate events into traces. Many alternative case notions can be defined depending on the perspective we want to take.

6. **Event log building**: from the discovered events and a case notion we can build an event log. Many case notions can be defined, and the corresponding event logs can be constructed in order to analyze different coexisting processes, or the same process from different perspectives.

We claim that these steps can be executed in a semi-automatic way, given that they allow for a certain customization depending on the characteristics of the environment to analyze. The goal is to obtain a preview of the processes interacting with the database at hand. We provide the Python library *eddytools*[1] (which stands for Event Data Discovery Tools) as a convenient implementation of the techniques proposed in this thesis with individual and customizable commands. This tool assists the user when executing the six commands to build event logs. The output of each command becomes the input of the next one. This allows us to perform a semi-supervised extraction too, since it is possible to review the intermediate results and correct them if necessary using domain knowledge. For the sake of this study, we decided to keep the corrections to the minimum, in order to evaluate the feasibility of the approach when facing a real-life database with reduced domain knowledge. In the coming subsections, we provide more details about the steps we executed, the challenges we faced, and the approach we used to solve them.

8.2.1 Data Exploration

The first step in any process mining project is data exploration. We need to get a feeling of what the data look like. In our case, we are dealing with the database of a Health Information System (HIS) belonging to a major provider within The Netherlands. The first thing we do is to connect to the database host. We need to know the location where the database is hosted within the network. Also, we need the access credentials to be granted by the IT department. The database runs on Microsoft SQL Server 2016. Since we intend to obtain the data from a terminal server within the same network as the database, we use the credentials of the

[1]`https://github.com/edugonza/eddytools`

Table 8.1: Some statistics obtained querying the source database before the extraction.

Number of tables	3570
Non-empty tables	1516
Primary keys	27
Unique keys	0
Foreign keys	0
Number of columns	40 666
Number of timestamp columns	2679
Number of rows	2.446×10^8
Average number of columns per table	11.39
Average number of timestamp columns per table	0.75
Average number of rows per non-empty table	1.613×10^5

Windows account to authenticate against it. Command 8.1 shows the command to execute using our python library (eddytools) to access the database (with the url *"mssql+pymssql://localhost/DB"*) and list the tables and their details.

Command 8.1: Command to obtain the list of tables in a database.

```
$ eddytools schema list-classes 'mssql+pymssql://localhost/DB' --details --o=tables.txt
```

Table 8.1 provides some details about the database at hand. The number of tables is high, more than 3000, with an average of around 11 columns per table. Also, we find an average of 0.75 timestamp columns per table, i.e., at best 75% of the tables have a timestamp attribute. Also, we see that the number of primary keys is minimal with respect to the number of tables. Unfortunately, foreign keys are simply non-existent. Normally, when foreign keys are not explicitly defined in the data schema, it does not mean that the tables are completely independent. In fact, relations exist within the data, but these relations are enforced at the application level instead. Since we are dealing with a proprietary closed application, we do not have access to any documented data schema. This represents quite a big challenge since, in order to correlate events, we need to correlate the rows from different tables. The *data schema discovery* step will tackle this issue.

Domain Knowledge

Observing the list of tables within the database, we realized that a certain structure is encoded in the name. Every table is preceded by a prefix indicating a category. From the total number of tables, we identified 169 different prefixes. Each prefix represents a cluster of tables that belongs to a specific department or a functional aspect of the system. Some example clusters are:

- OK: 102 tables. Tables related to *operatiekamer*, which in Dutch stands for operation rooms. In fact, the tables involved in this cluster handle surgery appointments that require the use of an operation room.

- AGENDA: 95 tables. Tables that handle appointments between doctors and patients.
- PATIENT: 46 tables. Information about patient profiles are stored in the tables in this group.
- FAKTUUR: 208 tables. Tables in this cluster refer to financial information such as invoicing and payments.
- ORDERCOM: 83 tables. This group of tables handles communications and task requests between hospital staff, such as doctors and nurses.
- SEH: 102 tables. Tables about *SpoedEisendeHulp*, which is Dutch for first aid or emergency room.

For the purpose of this study, we focused on three clusters: *OK*, *AGENDA*, and *PATIENT*. The following subsection covers the challenge of missing primary and foreign keys, and we propose a solution to tackle it.

8.2.2 Data Schema Discovery

The output of the previous step gives us an indication of the dimensionality and complexity of the dataset. Also, we have identified an important factor: incompleteness of the data schema. The lack of defined unique and foreign keys is a problem if we want to be able to correlate pieces of information belonging to different tables. In the literature, we find several approaches in the field of data profiling that can be applied in this context. Some of these approaches aim at identifying primary keys [101] by checking the uniqueness of values. Also, there exists work on foreign key discovery [126], which tries to identify inclusion of data from candidate foreign keys into certain given unique keys.

Checking **uniqueness** and inclusion of data is not a difficult task. However, it is computationally complex due to the size of the search-space to explore when approached in a naive way. If we want to identify valid unique keys, we need to check the uniqueness of values for column combinations per table. The complexity of unique key discovery grows linearly with respect to the number of tables, since it is an independent problem for each of them. However, the number of candidate (possible) unique keys per table t that we need to check ($|C^t_{uk}|$) is determined by the number of columns in the table (n) and the maximum number of columns accepted in the key candidate (m), resulting in the sum of the k-combinations of columns (binomial coefficient) for $k = 1..n$:

$$|C^t_{uk}| = \sum_{k=1}^{m} \binom{n}{k} \tag{8.1}$$

The number of combinations of foreign key candidates and unique keys to check for **inclusion** grows substantially with respect to the number of maximum columns per key. To accept a candidate foreign key as valid, we need to check the inclusion of its values into one of the valid unique keys (UK). The problem is more complex than unique key discovery, since the number of tables involved plays a role. Actually, the number of discovered unique keys has a large effect on the complexity, since we need to check the inclusion of values from all candidate foreign keys into all given unique keys. If we have a certain number of unique keys (of column lengths $1..m$) that we accept as valid ($|UK|$) and we want to discover foreign keys of a maximum size of m columns, we need to check the inclusion of the foreign key candidates of size $k = 1..m$ into the primary keys of the same size k. The number of column combinations (c) for foreign keys in a table t given a maximal key length of m in a table with n columns is given by the k-combinations of columns (binomial coefficient): $c = \binom{n}{k}$. However, when checking inclusion with respect to a unique key of the same length, the order of the columns is important. This means that, for each combination of columns of length k, we need to consider all the permutations of the combination. Therefore, given $k = 1..m$, the number of candidate foreign keys in table t to consider for inclusion ($|C^t_{fk}|$) is equal to the number of k-permutations of n columns:

$$|C^t_{fk}| = \sum_{k=1}^{m} {}^{n}P_k = \sum_{k=1}^{m} \frac{n!}{(n-k)!} \tag{8.2}$$

Calculating the sum of applying Equation 8.2 to each table in which we want to discover foreign keys gives us the number of foreign key candidates to check. Multiplying this sum by the number of valid unique keys $|UK|$ gives the number of inclusion checks to perform in the worst case scenario in order to explore the whole search space, i.e., when adopting a naive approach.

Optimizations

There are ways to prune the search space in order to reduce the complexity of the problem. A way to do it is to reduce the number of unique keys to consider. This can be done by reducing the detected unique keys to the minimum. For example, if we found a valid unique key of length k and columns $C' \in C$, any additional set of columns $C'' \in C$ that is a superset of C': $C'' \supseteq C'$ will be a valid unique key by definition. Therefore, we can exclude any candidate unique key that is a superset of verified unique keys. This will also have an impact on the number of inclusion checks of foreign key candidates when discovering foreign keys, since the number of valid unique keys will be smaller.

Another way to reduce the complexity is to reduce the number of permutations of the columns of foreign key candidates based on precomputed inclusion checks. Consider a set of columns C^{fk} for table t that represent a candidate foreign key, and a set of columns C^{uk} belonging to a certain valid unique key in table $t' \neq t$. If we precompute the result of the inclusion check for each pair of columns from C^{fk} and C^{uk} as a set of valid inclusions $VI = \{(c^{fk}, c^{uk}) \in C^{fk} \times C^{uk} \mid values(c^{fk}) \subseteq values(c^{uk})\}$, then we can skip any permutation of C^{fk} in which $c_i^{fk} \in C^{fk}$ is paired to $c_j^{uk} \in C^{uk}$ such that $(c_i^{fk}, c_j^{uk}) \notin VI$, i.e., if we know that their column values do not satisfy inclusion.

Until now we have approached the problem of foreign key discovery from a naive perspective for the sake of simplicity, ignoring any additional information about the columns and tables involved. However, in real life, we have access to certain meta-data that can help us to reduce the complexity of the computation. The data type of columns is a piece of information that we can leverage. Similarly to the optimization discussed in the previous paragraph, we can also skip permutations of columns for which the data type of the (c_i^{fk}, c_j^{uk}) pairs do not match. It would not make sense to check inclusion of the values of a text string column into a numeric column.

One way to reduce the execution time per uniqueness or inclusion check is to apply sampling. This means that, for large tables, we only perform the check on a random subset of the table rows. When the sample size is sufficiently large, the probability that the sample satisfies the uniqueness or inclusion check when the whole population (all rows) does not can be considered negligible. Applying sampling to the uniqueness check means that we will only verify the uniqueness of the values of a subset of the table rows. In the case of the inclusion check, we will obtain a sample of the table rows in the foreign key table, and check their inclusion on the values from the whole unique key table. Applying sampling, significantly increases the number of checks we can perform per unit of time when dealing with very large tables.

Additionally, meta data can be used too to reduce ambiguity and false positives. That is the case when matching foreign keys to unique keys. Several matches can occur, such that a foreign key satisfies inclusion with respect to more than one unique key. Since only one of these pairs of foreign-unique keys can be valid, we make use of meta data to select the most likely one. First, we look at column names. We compute a string similarity score between the pairs of matched columns for each candidate pair. Next, we select the unique key that presents a lower average string distance for all pairs of column names, discarding the other matches. This helps to reduce false positives and provide a more meaningful result.

The aforementioned strategies represent the main optimizations that have been implemented in our software package. Further optimizations and more complicated search space pruning techniques could be added in the future in order to execute the schema discovery step faster and more efficiently.

Execution

Our library (eddytools) implements a basic schema discovery method that includes the optimizations mentioned in the previous section. The goal is to integrate schema discovery within our pipeline so it can be carried out when the situation requires it, allowing for manual corrections too. In order to execute the schema discovery method we run the following command:

Command 8.2: Command to discover the data schema given a database and a list of tables for keys of length 1 and using sampling.

Table 8.2: Discovered unique and foreign keys for each cluster of tables.

Cluster	Tables	Discovered UKs	Discovered FKs	Pruned UKs
OK	102	367	75	268
AGENDA	95	458	71	331
PATIENT	46	189	28	143

```
$ eddytools schema discover 'mssql+pymssql://localhost/DB' ./schema_disc_dir --classes=tables.txt
    ↪ --max-fields=1 --sampling=5000
```

This command accesses the source database, computes unique and foreign key candidates, and performs all the required uniqueness and inclusion checks to obtain a final set of valid unique and foreign keys. The result is stored in editable text files for easy review and correction before the actual data extraction is carried out. We run this command for each of the considered table clusters (*OK*, *AGENDA*, and *PATIENT*). The number of columns per key has been limited to one for the sake of simplicity and to speed up the discovery. The results can be observed in Table 8.2.

For each cluster, the number of discovered unique keys (third column in Table 8.2) largely exceeds the number of existing tables. This means that, for a single table, we find more than one unique key. One of them will match the supposedly primary key, but any of them could be referred to by foreign keys belonging to other tables. After the foreign keys are discovered, we pruned the unique key set. We looked into tables referred to by foreign keys in other tables, and removed the unique keys that were not used by any foreign key. The result of the data schema discovery step is necessary in order to extract the data establishing the correct relations between objects obtained from different tables.

8.2.3 Data Extraction

During the data extraction step, relevant information from the source database is extracted and stored locally in the OpenSLEX format. The extracted data comprise the data model, objects, object versions, and relations between object versions found in the source database. After this step is executed, all the data will be stored locally and access to the original database will no longer be required. In order to perform the extraction and correlation of object versions, we need either a data schema obtained from explicitly defined unique and foreign keys or a discovered data schema as a result of the execution of the previous step (data schema discovery). In our case, because of the lack of a preexisting data schema, we use the one we discovered for each of the clusters. Command 8.3 performs the extraction, creating one object and object version for each table row found in the original tables, as well as one relation for each foreign key instance between two rows. In order to reduce the data dimensionality, we focused on the extraction of attributes that represent non-binary data, i.e., numbers, strings, timestamps, dates, and booleans. Any other binary format, e.g., *binary large objects* (BLOBs), were omitted during the extraction.

Table 8.3: Result of the data extraction phase for each cluster. Objects and object versions coincide in number given the direct transformation from rows to object versions. Relations represent foreign key instances.

Cluster	Classes	Objects	Object Versions	Relations
OK	102	1.047×10^6	1.047×10^6	5.950×10^5
AGENDA	95	6.727×10^6	6.727×10^6	5.153×10^6
PATIENT	46	3.116×10^6	3.116×10^6	3.271×10^4

Command 8.3: Command to extract the content of a database given a discovered data schema and a list of tables, and store it locally using the OpenSLEX format.

```
$ eddytools extract 'mssql+pymssql://localhost/DB' ./extraction_dir ./schema_disc_dir
    ↪ --classes=tables.txt
```

Table 8.3 shows the results of the extraction. As an example, we see that more than one million objects and object versions were obtained from the *OK* cluster, holding more than half a million relations. The number of objects is more than six times larger in the case of the *AGENDA* cluster, with almost ten times more relations. Despite having half the number of tables, the *PATIENT* cluster provided three times as many objects as the *OK* cluster. However, the *OK* cluster contains almost twenty times more relations than *PATIENT*. We see that the size of the clusters in number of objects is not proportional to the number of tables involved. Also, we can get an idea of the size of the data at hand when tackling a real-life dataset. Even focusing on relatively small parts of the database forced us to be able to handle millions of instances. Despite the relative simplicity of the extraction process, the size of the data was, in fact, one of the challenges to overcome during this phase. It forced us to adopt caching strategies in order to handle the extraction in an environment with very limited resources (a remote terminal server). Network and connectivity problems and source database uptime were some of the additional issues we faced when performing this task.

At the end of the data extraction step, we obtained *an OpenSLEX file for each cluster*. Each of these files contained a description of the data model, objects and object versions for each row, together with their non-binary attributes and relations that reflect instances of foreign keys between the extracted object versions. Having this information available, we are ready to apply the rest of the steps, which focus on preparing the data for process mining analysis.

8.2.4 Event Discovery

When analyzing the content of database tables, it is common to find that events are not explicitly defined. In that case, we need to discover them. Previously, we have carried out some experimental work to tackle the challenge of event discovery.

Table 8.4: Results of the event discovery task. The discovered event definitions provide a basic pattern (timestamp, activity name) in order to perform the event extraction.

Cluster	Classes	Event definitions	Extracted events
OK	102	58	1.005×10^6
AGENDA	95	93	1.535×10^7
PATIENT	46	86	7.543×10^6

However, in this case study we applied a more conservative approach that produces more reliable results than the experimental techniques. We consider the presence of a timestamp column as evidence of the existence of an event. For every timestamp column in a table, we create as many events as rows it contains, taking the value of the timestamp field as the time of the event occurrence and the column name as the name of the corresponding activity. This approach provides meaningful results easy to interpret with a low level of false events. Command 8.4 performs the event discovery task, plus the event extraction based on the discovered event definitions.

Command 8.4: Command to discover and create events in an extracted dataset considering timestamp columns as activity names.

```
$ eddytools events ./extraction_dir/mm-extracted.slexmm ./event_disc_dir --ts --build-events
```

Table 8.4 shows the number of event definitions discovered and the number of events extracted. We see that the *PATIENT* cluster contains almost twice as many event definitions as the number of classes. It also contains the highest amount of extracted events, more than 7 million. In the *OK* cluster, the number of discovered event definitions is roughly half of the number of classes and around 1 million of extracted events. The *AGENDA* cluster presents a ratio of almost one event definition per class and a total of more than 15 million events extracted.

The events extracted during this phase have been added to the content of the extracted OpenSLEX files. Although these events are the necessary building blocks to generate the event logs we are looking for, they are not the only requirement. We need to correlate these events in a meaningful way, such that they can be grouped in traces. There are many ways to correlate them depending on the chosen case notion. The next section focuses on the case notion discovery task.

8.2.5 Case Notion Discovery

Another step towards the creation of an event log is to choose a meaningful case notion. We need a case notion in order to group events together into traces, which we will collect into event logs. As we described in Chapter 5, first we generate candidate case notions based on the discovered data schema. Second, we collect certain statistics at the class level. Next, we predict certain log metrics (support, level of detail, and average number of events per trace) before the event log is built. Finally, we combine the predicted metrics to compute a case notion "interestingness" score. The scores are used to rank the case notions from more to less (potentially) interesting.

During the execution of this step, we made a modification on the global scoring function presented in Chapter 5. The previously proposed function (Equation 5.19) computes the score as the weighted average of the three metrics (support, level of detail, and average number of events per trace). However, while carrying out this study, we realized that computing the weighted harmonic average (Equation 8.3) instead yielded more meaningful results for this specific dataset. Due to the fact that the framework proposed in Chapter 5 is easily extendable, it was not an issue to replace the default scoring function by the one in Equation 8.4.

$$H = \frac{\sum_{i=1}^{n} w_i}{\sum_{i=1}^{n} \frac{w_i}{x_i}} \text{ where } w_1, \dots w_n \text{ are weights, and } x_1, \dots, x_n \text{ are values.} \tag{8.3}$$

The weighted harmonic average takes into account the trade-off between the three metrics (Equation 8.4), favoring candidates that score well in all three dimensions, and punishing candidates that score low in some of the dimensions.

$$pgsf(CN, CNS) = \frac{w_{sp} + w_{lod} + w_{ae}}{\frac{w_{sp}}{\widehat{s_{sp}}(CN,CNS)} + \frac{w_{lod}}{\widehat{s_{lod}}(CN,CNS)} + \frac{w_{ae}}{\widehat{s_{ae}}(CN,CNS)}} \tag{8.4}$$

An example is the case of candidates that present the desired level of detail and average number of events per trace, but for which the predicted support is close to zero. For this reason, it obtains a low "interestingness" score.

Table 8.5 shows the default parameters used when running the case notion discovery and recommendation step in Command 8.5. The *mode*, *max*, and *min* parameters for each metric (*sp*, *lod*, and *ae*) are used to estimate the β and α parameters of the beta probability distribution function (beta pdf) that will give us a score for the metric. According to the chosen values, we prefer case notions for which the average number of activities per trace is between 3 and 10, and preferably close to 7. Also, case notions with an average number of events per trace above 3000 will score lower. With respect to the support, we will prefer case notions with as many traces as possible. The weights (w_{sp}, w_{lod}, and w_{ae}) are used when computing the global score per case notion, combining the individual scores for each metric (Equation 8.4).

Command 8.5: Command to discover case notions and build the top 3 event logs based on the ranking of "interestingness".

```
$ eddytools cases ./event_disc_dir/mm-events.slexmm ./cn_disc_dir --build-logs --topk=3
```

The execution of Command 8.5 generated a ranking of case notions, sorted based on their predicted "interestingness". Table 8.6 shows the total number of generated case notions, as well as information on for how many of them the predicted *support*

Table 8.5: Parameters used during the case notion discovery and recommendation step.

Parameter	Value	Description
$mode_{sp}$	-	Mode of the beta pdf used to score the *support* (number of cases). Default is null, since we try to maximize *sp*.
max_{sp}	∞	Highest value of the desired range used to score the *support* value.
min_{sp}	0	Highest value of the desired range used to score the support value.
$mode_{lod}$	7	Mode of the beta pdf used to score the *lod* (level of detail) value.
max_{lod}	10	Highest value of the desired range used to score the *lod* value.
min_{lod}	3	Lowest value of the desired range used to score the *lod* value.
$mode_{ae}$	1500	Mode of the beta pdf used to score the *ae* (average number of events per trace) value.
max_{ae}	3000	Highest value of the desired range used to score the *ae* value.
min_{ae}	0	Lowest value of the desired range used to score the *ae* value.
w_{sp}	0.33	Weight of the *support* score on the final global score.
w_{lod}	0.33	Weight of the *lod* score on the final global score.
w_{ae}	0.33	Weight of the *ae* score on the final global score.

Table 8.6: Case notions discovered for each cluster.

Cluster	Classes	Case Notions	Non-empty ($\widehat{SP} > 0$)
OK	102	1277	455
AGENDA	95	1881	790
PATIENT	46	131	64

(number of cases) was higher than zero. We gathered the details of the three top ranked case notions from the individual ranking of each cluster (*OK*, *AGENDA*, and *PATIENT*). The resulting case notions can be observed in Figures 8.1 to 8.9. The notation is the same as in Figure 5.4 of Chapter 5, where squares represent classes (tables), a square with a bold border represents the root class of the case notion, solid arrows indicate the hierarchy relation within the case notion, and dashed arrows show the used relationships (foreign keys) between tables to correlate events.

The three top-ranked case notions in the *OK* cluster are the ones depicted in Figures 8.1-8.3. We see that the three of them represent a similar process, given that the same tables are present in all of them: *dbo.OK_OKINFO*, *dbo.OK_OKANNULE*, and *dbo.OK_OKROUTE*. The difference between the three case notions is the class that acts as root. This will determine how traces are built, since at least an instance of that class will be necessary in order to create a new trace. This produces slightly different results for each case notion, even though the three of them represent a similar process albeit from different perspectives.

Domain Knowledge

The case notion in Figure 8.1 reflects the process of surgery room appointments from the point of view of the central surgery appointment class (*dbo.OK_OK-*

INFO).
The case notion in Figure 8.2 represents the same process, but from the point of view of the execution of the surgery appointments (*dbo.OK_OKROUTE*). The third case notion (Figure 8.3) looks at the process from the point of view of the cancellation of surgery appointments.

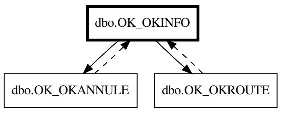


Figure 8.1: First case notion in the ranking for the *OK* cluster, with *dbo.OK_OKINFO* as root class. It reflects the process of surgery room appointments.

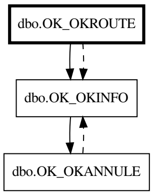


Figure 8.2: Second case notion in the ranking for the *OK* cluster, with *dbo.OK_OKROUTE* as root class. It reflects the process of surgery room appointments.

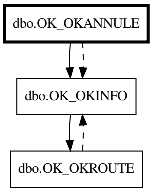


Figure 8.3: Third case notion in the ranking for the *OK* cluster, with *dbo.OK_-OKANNULE* as root class. It reflects the process of surgery room appointment cancellation.

The most relevant case notions discovered in the *AGENDA* cluster are related to the appointments between patients and doctors. In this case, the three case notions share some common classes (*dbo.AGENDA_AFSPRAAK*, *dbo.AGENDA_AGENDA*, and *dbo.AGENDA_WACHTTYD*). Also, different additional and root classes are considered for each case notion.

Domain Knowledge

Figure 8.4 shows that, for the first case notion, *dbo.AGENDA_AFSPRAAK* is the root class, which collects the agenda appointments (*afspraak* in Dutch) between patients and doctors. It is also linked to the central agenda class *dbo.AGENDA_AGENDA* and the waiting list class *dbo.AGENDA_WACHT-TYD*, among others.
For the second case notion (Figure 8.5), the root class is *dbo.AGENDA_BEPAL-ING*, which seems to be related to some sort of assessment or evaluation, being linked to other classes related to a waiting list (*dbo.AGENDA_WACHTTYD*, and *dbo.AGENDA_AGNWACHT*) and the main agenda appointment classes (*dbo.AGENDA_AGENDA* and *dbo.AGENDA_AFSPRAAK*).
The third case notion (Figure 8.6) takes the central agenda class *dbo.AGENDA_-AGENDA* as root, and combines the appointment class *dbo.AGENDA_AF-SPRAAK*, the waiting list class *dbo.AGENDA_WACHTTYD*, and two more classes that have some relation to locations (*dbo.AGENDA_AFSPVERROPT* and *dbo.AGENDA_SUBAGLOKMAP*).

We see that the three case notions are quite different in terms of root classes and additional classes involved. Still, they maintain some common factors, all notably related to the appointment classes.

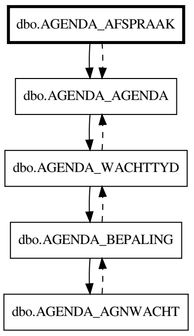


Figure 8.4: First case notion in the ranking for the *AGENDA* cluster, with *dbo.AGENDA_AFSPRAAK* as root class. It reflects the process of agenda appointments between patients and doctors.

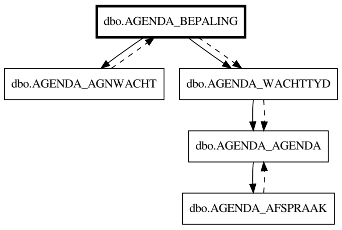


Figure 8.5: Second case notion in the ranking for the *AGENDA* cluster, with *dbo.AGENDA_BEPALING* as root class. It reflects the process of agenda appointments between patients and doctors combined with information about waiting lists.

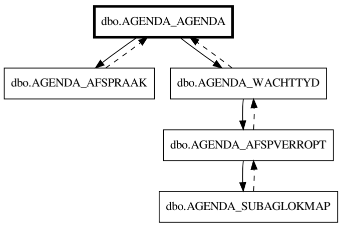


Figure 8.6: Third case notion in the ranking for the *AGENDA* cluster, with *dbo.OK_AGENDA* as root class. It reflects the process of agenda appointments between patients and doctors combined with information about waiting lists and locations.

Finally, we analyze the case notions discovered in the *PATIENT* cluster. The top three case notions are related to the combination of the *dbo.PATIENT_MERGELOG* and the *dbo.PATIENT_PATLAST* classes. Each case notion defines one of these two classes as root. Only the third one takes a different class into account, *dbo.PATIENT_PATLAST_OLD*, which seems to be a copy of *dbo.PATIENT_PATLAST*. We see that the case notions that we obtained are much less complex and descriptive than the ones found for other clusters. One reason is the scarcity of case notions compared to other clusters (only 64 non-empty case notions discovered). Also, we observed that the discovered events are concentrated on very few classes (Table 8.7). This means that most of the discovered case notions related to classes with little events will score very low in terms of support, which penalizes their global score.

In the next section, we will generate the corresponding event logs for the case notions that we have presented. Also, we will compare the predicted and actual values for each of the metrics considered to compute the case notion score.

8.2.6 Event Log Building

Event log building is the last step of our approach. It is also one of the most mechanical parts of the process, since all the necessary pieces of information have been discovered and extracted in the previous steps. We already discovered a data schema, extracted all the relevant data from the source database, discovered and extracted events, and

Table 8.7: Distribution of discovered activities and events per class for the *PATIENT* cluster.

Class	Activities	Extracted events
dbo.PATIENT__PATIENT	9	2474660
dbo.PATIENT__POLIS	3	2044415
dbo.PATIENT__PATVERZ	2	464185
dbo.PATIENT__MERGELOG	2	42272
dbo.PATIENT__PATLAST	1	16219
dbo.PATIENT__PATLAST__OLD	1	16207
dbo.PATIENT__MRGLOG	1	7001
dbo.PATIENT__COVSTBAT	3	465
dbo.PATIENT__PATIENT__EXT	2	365
dbo.PATIENT__PATLINK	2	247
dbo.PATIENT__PATHIST	4	22
dbo.PATIENT__HINDER	3	3

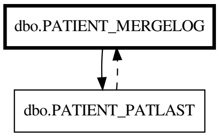


Figure 8.7: First case notion in the ranking for the *PATIENT* cluster, with *dbo.PATIENT__MERGELOG* as root class.

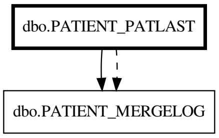


Figure 8.8: Second case notion in the ranking for the *PATIENT* cluster, with *dbo.PATIENT__PATLAST* as root class.

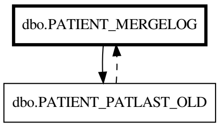


Figure 8.9: Third case notion in the ranking for the *PATIENT* cluster, with *dbo.PATIENT__MERGELOG* as root class.

computed a ranking of case notions. During this step we will build one event log for each selected case notion. We do so by combining the extracted events by means of the relations we computed between the objects obtained from the database. This is the phase when the application of our method materializes into a collection of event logs ready to be analyzed.

We will compute one event log for each of the three top-ranked case notions per cluster presented in the previous section. When executing Command 8.5, we already indicated that we wanted to build the event logs for the top three case notions. That means that we only need to export the computed event logs to a format that we can use for analysis. Command 8.6 shows how to do this.

Command 8.6: Command to export previously computed event logs to csv format.

```
$ eddytools logs ./cn_disc_dir/mm-logs.slexmm --export_log=1 --o=log_1.csv
```

Next, we compute the actual values for each metric (*support*, *lod*, and *ae*) to compare them to the ones that we predicted. Table 8.8 shows the results. In general, we see that all the computed event logs contain a large amount of traces (*SP* column), and an acceptable number of activities per trace (*LoD* column). Also, the average number of events per trace is equal to the average number of activities per trace. The reason for this is the way we correlate events together. Also, because the activity name of such events corresponds to the timestamp column name, no more than one event per activity will belong to each trace. If we considered case notions with *converging* classes (as described in Chapter 5), the values for *ae* and *lod* would differ significantly.

We can compare the predicted versus actual values for each metric. In the case of *support*, it is obvious that the predicted value is several orders of magnitude higher than the actual one. This is due to the fact that the prediction is based on an average between the lower and upper bound of the metric. In the case of *support*, the upper bound is always very high. A better strategy to improve the accuracy would be to consider the lower bound as a prediction. Also, defining a heuristic that takes into account other factors of the case notion to provide a prediction between the lower and upper bound would help to increase the accuracy. When it comes to *lod* and *ae*, the prediction is much more accurate.

Table 8.8: Details on the event logs computed for each of the three top-ranked case notions per cluster. Predicted vs. actual values per metric are displayed.

Cluster / Case Notion	Rank	Score	$\widehat{SP}$	SP	$\widehat{LoD}$	LoD	$\widehat{AE}$	AE
OK								
cn_1059	1	2.5×10^{-2}	3.8×10^{14}	2.2×10^{5}	6.50	6.59	6.50	6.59
cn_1061	2	2.3×10^{-2}	3.8×10^{14}	1.1×10^{5}	6.00	7.96	6.00	7.96
cn_1060	3	2.3×10^{-2}	3.8×10^{14}	1.1×10^{5}	6.00	7.50	6.00	7.50
AGENDA								
cn_292	1	1.9×10^{-2}	1.1×10^{19}	3.5×10^{6}	5.50	4.05	5.50	4.05
cn_291	2	1.9×10^{-2}	1.1×10^{19}	2.6×10^{3}	5.50	2.00	5.50	2.00
cn_57	3	1.6×10^{-2}	2.6×10^{20}	5.0×10^{5}	4.00	4.57	4.00	4.57
PATIENT								
cn_65	1	6.8×10^{-4}	3.3×10^{8}	4.1×10^{4}	2.00	1.43	2.00	1.43
cn_64	2	6.8×10^{-4}	3.3×10^{8}	1.6×10^{4}	2.00	2.03	2.00	2.03
cn_63	3	6.8×10^{-4}	3.3×10^{8}	4.1×10^{4}	2.00	1.43	2.00	1.43

This was the last step of our method for event log generation. So far, all the steps were performed trying to reduce the amount of manual work required. Additionally, we used some domain knowledge, e.g., information about table prefixes in order to divide the dataset in clusters of tables. This allowed us to reduce the complexity analyzing each cluster separately. Also, the intermediate results were not corrected in any way. In order to get a better understanding of the underlying processes interacting with the database at hand, we need to apply other process mining techniques. In the next section we will discover process models based on the computed event logs.

8.3 Results

In the previous section, we followed a semi-automated method to discover and obtain event logs with minimal manual work. Once we have one or many event logs, we can go on with the process mining analysis in order to get insights. In this section, we show the results of running the Inductive Visual Miner [65] on each log to obtain a process model.

We start with the first event log in the ranking of case notions for the cluster *OK* (cn_1059). This event log was computed according to the case notion in Figure 8.1, which represents the process of operation room appointments with cancellations.

Domain Knowledge

The event log corresponding to the case notion in Figure 8.1 combines events belonging to the following three tables:

- *dbo.OK_INFO*: This is the central table about surgical operations. It holds information about appointments that require an operation room.
- *dbo.OK_OKANNULE*: Cancellations of operation room appointments.
- *dbo.OK_OKROUTE*: Information related to what happens inside of the operation rooms for a specific appointment.

Figure 8.10 shows the process model obtained when mining the event log with the Inductive Miner.

Domain Knowledge

These are the most important activities within the process in Figure 8.10:

- *dbo.OK_OKINFO.INVOER_D*: indicates the moment when something was inserted in the appointment record, normally by a human operator,
- *dbo.OK_OKINFO.AANVRAAG_D*: something was requested, either at the appointment level, or even during the operation,
- *dbo.OK_OKINFO.AANMAAK_D*: the appointment was created in the system,
- *dbo.OK_OKANNULE.ANNUDATUM*: the appointment has been canceled,
- *dbo.OK_OKANNULE.GEPLANDEDA*: the date in which the canceled appointment was planned,
- *dbo.OK_OKROUTE.STARTDATE*: the moment when the operation starts,
- *dbo.OK_OKROUTE.VERTREKDAT*: the moment when the personnel leaves the operation room,
- *dbo.OK_OKINFO.OPERATIE_D*: planned date for the operation,

The discovered model in Figure 8.10 shows a nicely structured process, with some parallelism on the appointment creation (at the top) and a logical structure when

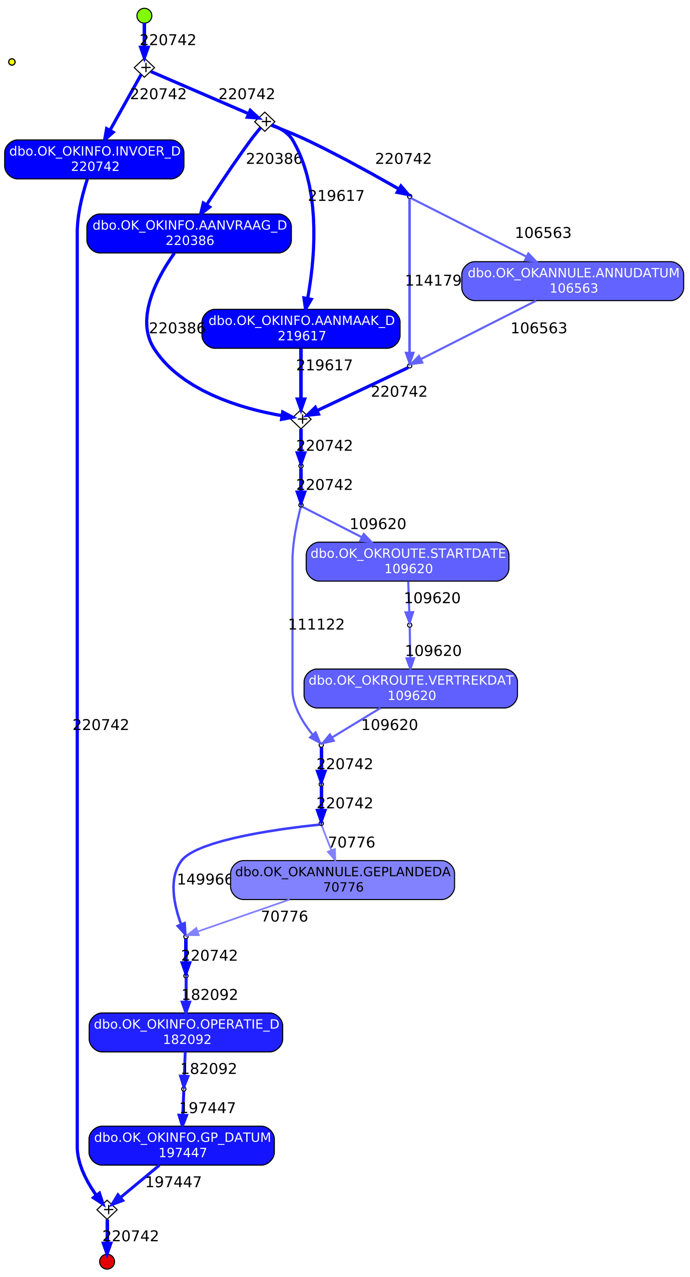


Figure 8.10: Process model mined with the Inductive Miner based on the event log of cluster OK corresponding to the case notion in Figure 8.1.

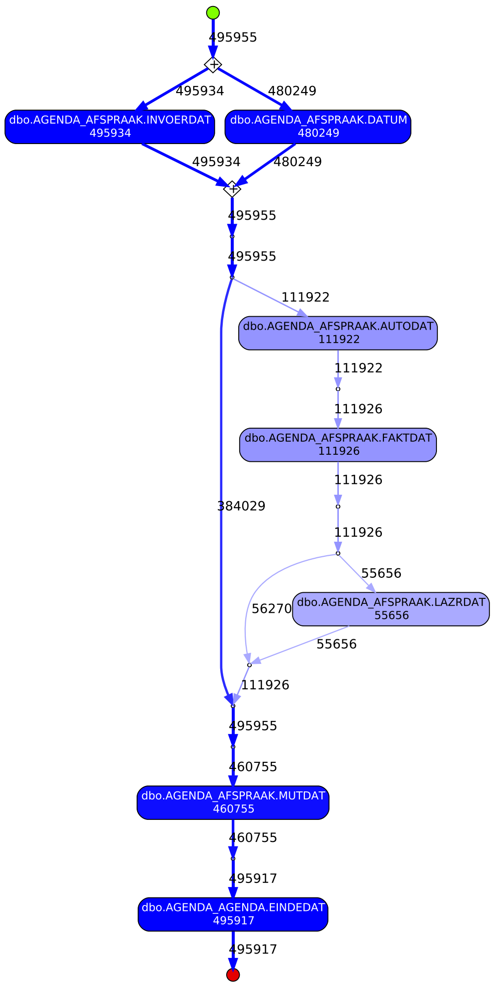

495955
495934
480249
dbo.AGENDA_AFSPRAAK.INVOERDAT 495934
dbo.AGENDA_AFSPRAAK.DATUM 480249
495934
480249
495955
495955
111922
dbo.AGENDA_AFSPRAAK.AUTODAT 111922
111922
111926
dbo.AGENDA_AFSPRAAK.FAKTDAT 111926
111926
384029
111926
55656
56270
dbo.AGENDA_AFSPRAAK.LAZRDAT 55656
55656
111926
495955
460755
dbo.AGENDA_AFSPRAAK.MUTDAT 460755
460755
495917
dbo.AGENDA_AGENDA.EINDEDAT 495917
495917

Figure 8.11: Process model mined with the Inductive Miner based on the event log of cluster AGENDA corresponding to the case notion in Figure 8.5.

performing the operations (in the middle) with the sequence of activities that indicate the start and end of the operation. Activities belonging to canceled appointments are included in this process, which shows at which point these cancellations happen. Normally, they occur in parallel with the appointment creation. Something to notice is the choice between the execution of the operation, or the skip arc at the center of the model. This can be due to the existence of incomplete cases, i.e., for which an appointment has been planned but has not been completed yet.

The next model we obtained (Figure 8.11) was based on the event log corresponding to the case notion in Figure 8.5. This case notion represents the process of appointments between doctors and patients.

Domain Knowledge

The process in Figure 8.11 handles the appointments between doctors and patients. The main activities involved are:

- *dbo.AGENDA_AFSPRAAK.INVOERDAT*: date of creation of the appointment,
- *dbo.AGENDA_AFSPRAAK.DATUM*: date of the appointment,
- *dbo.AGENDA_AFSPRAAK.FAKTDAT*: date of the invoicing,
- *dbo.AGENDA_AFSPRAAK.MUTDAT*: date of modification of the appointment,
- *dbo.AGENDA_AGENDA.EINDEDAT*: end of the agenda process.

The process shows a rather sequential structure. It has two activities happening in parallel at the start (*dbo.AGENDA_AFSPRAAK.INVOERDAT* and *dbo.AGENDA_AFSPRAAK.DATUM*) and a choice block in the middle, which can be skipped. The initial parallel part corresponds to the appointment creation. We see that the appointment creation can happen in parallel with the appointment date itself. This means that many of the appointments take place on the same day they are created. Then, the optional block seems to be related to the invoicing process. It only happens in roughly 1/5 of the cases. At the end, some appointments are modified and the process ends.

Finally, we discovered a process model using the event log generated according to the case notion in Figure 8.7. This case notion combines events from the classes *dbo.PATIENT_MERGELOG* and *dbo.PATIENT_PATLAST*, representing the process of how doctors access patient records.

Domain Knowledge

The class *MERGELOG* combines patient events from different sources. *PATLAST* stores the last time that a doctor accessed the record of a patient.

This process is rather simple, with only three activities and a very sequential structure in which most of the times its activities are skipped. This is an example of a discovered case notion for which the event log does not represent a very meaningful process. Either because of the choice of activity names or because of the simplicity of the process, the result does not provide us with interesting insights, even though the process might match the desired criteria to consider an event log as interesting. This is an example to consider in order to improve the technique further.

In the next section we will show an example of data querying on this dataset, and we will discover a process model from the resulting event log.

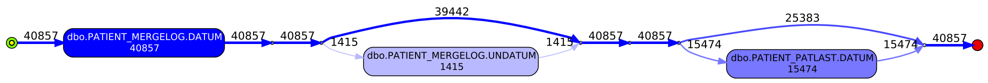

40857
dbo.PATIENT_MERGELOG.DATUM
40857
40857
40857
39442
1415
dbo.PATIENT_MERGELOG.UNDATUM
1415
1415
40857
40857
25383
15474
dbo.PATIENT_PATLAST.DATUM
15474
15474
40857

Figure 8.12: Process model mined with the Inductive Miner based on the event log of cluster PATIENT corresponding to the case notion in Figure 8.7.

8.4 Data Querying

It is often the case that, when preparing data for process mining, the obtained event logs are too detailed. Also, sometimes we need to focus on a part of the data to answer a specific business question. In this section, we leverage domain knowledge to show an example of data querying. We want to obtain a filtered event log based on the extracted data from the *OK* cluster previously presented. We will use DAPOQ-Lang, the data querying language proposed in Chapter 7.

From the case notion displayed in Figure 8.1, we learned that the combination of surgery appointments and cancellations could provide an interesting view on the underlying processes interacting with the database. One business question (BQ) to answer on such a process can relate to the appointments conducted with a specific type of anesthesia, in a certain period of time:

> **BQ**: What was the process followed when applying local anesthesia between March 2010 and July 2013 when handling surgery appointments?

In order to provide an answer to this business question, first we need to correlate cases to types of anesthetics. Next, we must filter these cases based on the time of occurrence. A general query (GQ) for this business question would look like this:

> **GQ**: Which cases of log LOG are related to objects of class CL for which the value of attribute AT is X, and the cases started and ended within the period T1 and T2?

We need to translate this general query to DAPOQ-Lang adapting it to the *OK* dataset. We require certain domain knowledge.

Domain Knowledge

In order to translate the generic query *GQ* into a DAPOQ-Lang query that we could execute, we require certain details that come from domain knowledge:

- LOG: this is the id of the event log that contains traces related to surgery appointments. From domain knowledge we know that the event log corresponding to the case notion in Figure 8.1 represents the desired process. The generated event log in our dataset has the id “1”,

- CL: this is the class that contains information about the surgical procedures. *dbo.OK_OKINFO* is the corresponding table within the *OK* cluster,

- AT: this is the attribute of the CL class that reflects the type of anesthesia applied during the procedure. The attribute *ANAESTTECH* belonging to the table *dbo.OK_OKINFO* is the one we need,
- X: this is the value of the attribute AT that indicates that local anesthesia was used. For the attribute *ANAESTTECH* we observed that the procedures in which local anesthesia was used contain the value “LOK”.

Query 8.7 shows the translated query. First, we select the log with id “1” (line 1). This is the event log corresponding to the case notion in Figure 8.1. Next, we select the object versions of *dbo.OK_OKINFO* for which the attribute *ANAESTTECH* has the value “LOK”, which means that local anesthesia was applied (lines 3-5). After that, we select the cases from log 1 related to an object version in the set of surgery appointments in which local anesthesia was used (lines 7-9). Now we can compute the periods on this set of cases, which will be cached. This could also be done per case within the *where* clause (line 16), but it is more efficient to compute it on the whole batch of cases at once. Next, we define the period of time in which we are interested (line 11) and we filter the selected cases that happened during this period (lines 13-15). Finally, we export an event log with the final filtered cases (line 17).

Query 8.7: Export the cases of log 1 where local anesthesia was applied and happened between March 2010 and July 2013.

```
 1 log = allLogs().where{id==1}
 2 
 3 cl = allClasses().where{name=="dbo.OK_OKINFO"}
 4 
 5 anaesthesic = versionsOf(objectsOf(cl)).where{at.ANAESTTECH == "LOK"}
 6 
 7 cases = casesOf(log)
 8 
 9 filtered_cases = cases.intersection(casesOf(anaesthesic))
10 
11 periodsOf(filtered_cases)
12 
13 def period = createPeriod("2010/03/01 00:00","2013/07/01 00:00","yyyy/MM/dd HH:mm")
14 
15 filtered_cases_time = filtered_cases.where{
16   during(globalPeriodOf(it),period)
17 }
18 
19 exportXLogsOf(filtered_cases_time)
```

The execution of Query 8.7 yields an event log with 4121 cases, 20234 events, and an average of 5 activities per case. This event log should contain the subset of cases in which local anesthesia was applied, which occurred between March 2010 and July 2013.

Figure 8.13 shows the process model discovered with the Inductive Miner. The first thing we notice is the lack of an apparent structure when compared to the model in Figure 8.10. In this case, all the appointment management activities are represented at the top part of the model, happening mostly in parallel. Something unusual is the low rate of executed operations (36) with respect to the total number of appointments (4121). Also, the sum of executed operations and the number of

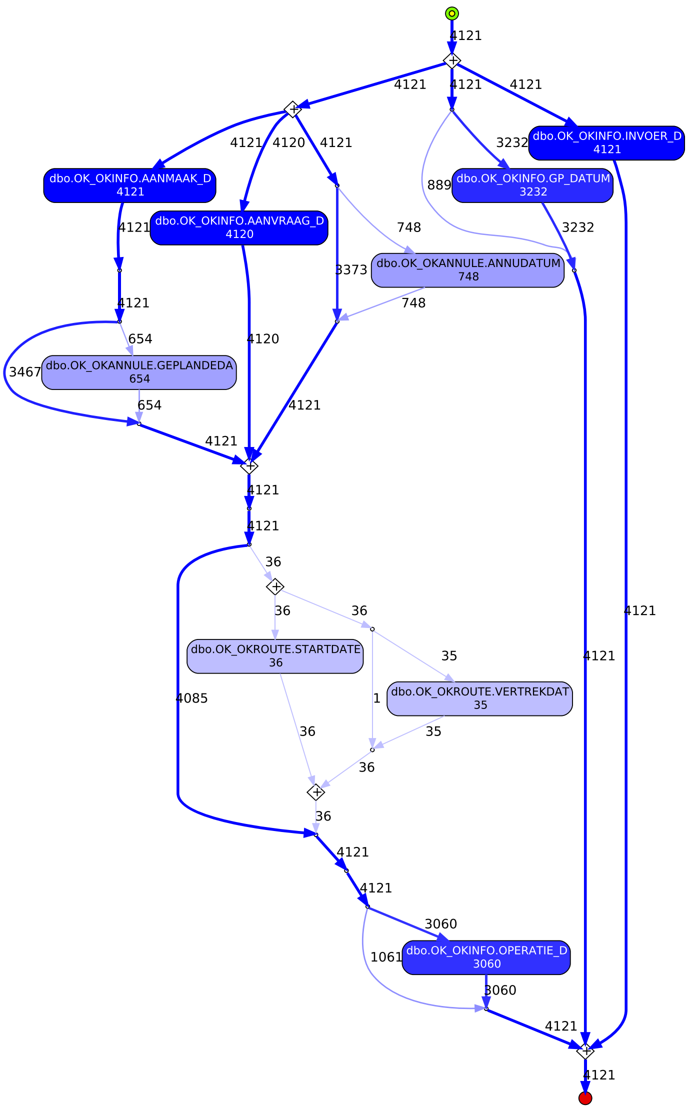

4121
4121
4121
4121
dbo.OK_OKINFO.INVOER_D 4121
4121
4120
4121
3232
dbo.OK_OKINFO.AANMAAK_D 4121
889
dbo.OK_OKINFO.GP_DATUM 3232
dbo.OK_OKINFO.AANVRAAG_D 4120
4121
748
3232
dbo.OK_OKANNULE.ANNUDATUM 748
3373
4121
748
654
4120
3467
dbo.OK_OKANNULE.GEPLANDEDA 654
654
4121
4121
4121
4121
36
36
36
4121
dbo.OK_OKROUTE.STARTDATE 36
35
4121
4085
1
dbo.OK_OKROUTE.VERTREKDAT 35
36
35
36
36
4121
4121
3060
1061
dbo.OK_OKINFO.OPERATIE_D 3060
3060
4121
4121

Figure 8.13: Process model mined with the Inductive Miner based on the event log of cluster OK corresponding to the result of Query 8.7.

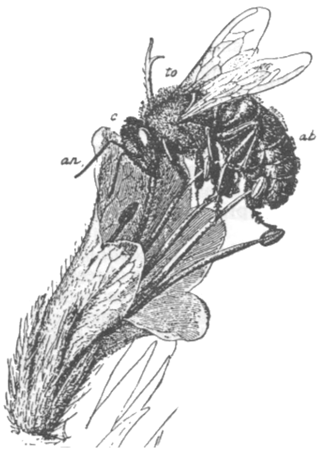
canceled appointment does not add up to the total of cases. Given the period used to filter the cases (2010 to 2013) it is unlikely that this difference is due to incomplete cases, i.e., all the cases that occurred during that period should have ended already and therefore be contained within the dataset. The most plausible reason for the lack of operation executions is that they were either not recorded or that the cancellation was not registered in the system. Nevertheless, this is something worth reporting to the business owners.

To summarize, we conclude this section with a reflection on the lessons learned. This case study demonstrates that our methodology has the potential to transform the way data extraction and preparation is approached by process mining professionals. Our techniques covered all the steps necessary to obtain event logs from databases. The results were meaningful in most of the cases and domain experts concluded that the obtained event logs represent existing processes that run within the HIS under study. The resulting event logs represent a good start for any process mining project, allowing the analysts to get a feeling on the kind of processes interacting with the database at hand. Also, we provide the means for the user to correct or modify the intermediate results at each step, e.g., correcting the discovered data schema, adding non-detected event definitions, etc. Also, we showed how our query language (DAPOQ-Lang) makes it possible to refine the resulting event logs to cover the user's area of interest. We do not aim at replacing completely manual approaches, but to automate as much as possible. This reduces the workload and the possibility of introducing errors while building event logs.

We also have identified several limitations of our techniques. First, discovering the data schema of a database is a complex task. Further performance improvements to reduce the search space are required in order to be able to apply the technique at scale for keys of higher length. Also, reducing the number of false positives, e.g., not meaningful foreign keys, would help to reduce the complexity of the discovered data schema. Another step where improvements can be made is in the discovery of case notions. Improving the accuracy of the predicted log metrics would inherently improve the quality of the case notions ranking. Finally, adding more metrics to the log "interestingness" assessment would be helpful to make a better decision on which case notions would actually be meaningful.

8.5 Chapter Summary

In this chapter, we have demonstrated the use of the techniques proposed in this thesis in a real-life environment. We applied a semi-automatic methodology to cover all the steps that need to be followed from the first access to a database, until the moment we obtain an event log. All the steps were executed trying to reduce the amount of domain knowledge needed. We showed that following this methodology enabled us to obtain insights on the underlying processes that interact with the database reducing time and resources spent interviewing domain experts or reading extensive documentation. In fact, this case study was executed in a particularly challenging environment due to the complexity and size of the dataset and the lack of documentation. Also, the privacy constraints given the nature of the data forced us to adapt our implementation in order to be executed in a resource-limited environment. As a result, we were able to obtain meaningful event logs and mine them during the analysis phase. Finally, we identified some limitations of our techniques, and the opportunities for improvement.



Worker honeybee on a flower of Echium maritimum.

"Cours complet d'apiculture", Georges de Layens and Gaston Bonnier, 1897

9 Conclusion

The present chapter concludes the thesis. First, the contributions presented in all previous chapters are summarized in Section 9.1. Next, we discuss the limitations of the results in Section 9.2. In Section 9.3, we propose improvements and opportunities for future work. Finally, in Section 9.4 we reflect on the overall thesis.

9.1 Contributions

In Chapter 1 of this thesis, we introduced the field of process mining and we discussed the research focus: *to research methods to extract event data and support the user in the process of generating event logs suitable for process mining analysis.* Additionally, we listed a set of challenges related to our main goal:

- Challenge 1: Finding, merging, and cleaning event data,
- Challenge 2: Dealing with complex event logs having diverse characteristics,
- Challenge 3: Cross-organizational mining,
- Challenge 4: Multi-perspective event log building,
- Challenge 5: Improve usability for non-experts,
- Challenge 6: Fill the domain knowledge gap in event log extraction,
- Challenge 7: Question-driven log extraction.

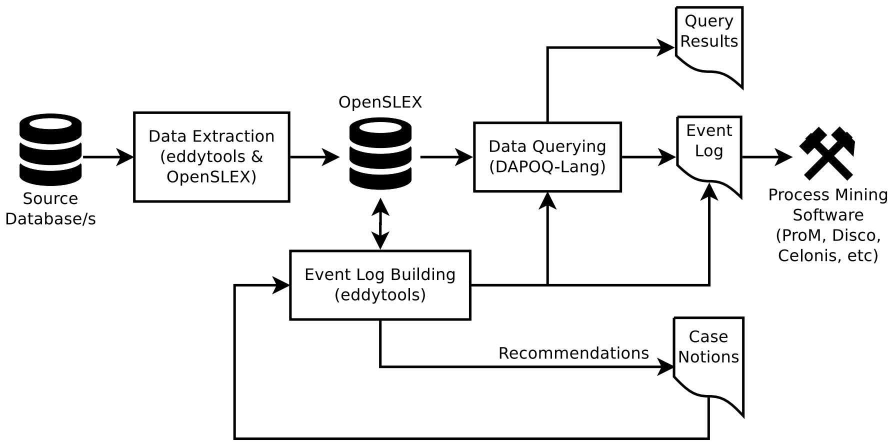

Query Results
Source Database/s
Data Extraction (eddytools & OpenSLEX)
OpenSLEX
Data Querying (DAPOQ-Lang)
Event Log
Process Mining Software (ProM, Disco, Celonis, etc)
Event Log Building (eddytools)
Recommendations
Case Notions

Figure 9.1: Overview of the contributions of this thesis, in the context of a pipeline connecting databases with existing process mining techniques.

In this section, we describe the contributions made in this thesis and the relation to the challenges mentioned above. The contributions are divided into three blocks, according to the phase of a process mining analysis project to which they relate: (1) data extraction, (2) event log building, and (3) data querying.

9.1.1 Data Extraction

The data extraction phase on a process mining project consists of the process of accessing, extracting, and transforming the data to a standard format. In the context of this thesis, we contributed to the data extraction phase by defining a meta-model for process mining on databases (OpenSLEX) (Chapter 3). This meta-model provides a standardization layer decoupling data extraction from event log building, data query, and process mining analysis. Also, data extraction was supported by means of several adapters (Chapter 4). These adapters are able to extract and transform data from different environments (SAP, Oracle redo-logs, etc.) to the OpenSLEX format.

The contributions made in this thesis with respect to data extraction tackle some of the aspects highlighted by challenges 1, 2 and 3. We believe that the definition of a standard meta-model for event data extraction contributes to alleviating the problems found when searching and merging event data. Compared to alternatives like XES, OpenSLEX becomes especially useful when dealing with multidimensional and complex data given its ability to provide a standard representation without flattening data into an event log format. Also, the task of merging data obtained from different systems and organizations becomes easier when a standard target format is defined.

9.1.2 Event Log Building

The goal of the event log building phase is to obtain event logs from the data extracted from the original data source. This thesis presents several contributions that tackle some of the challenges in event log building. In Chapter 5, we provided techniques to find and select different case notions together with a method to correlate events into cases in the context of databases. The event log building method tackles Challenge 4, since each case notion represents an alternative view or perspective on the underlying data. The presented method for event log building has been implemented in the tool *eddytools*. This helps to alleviate the technical complexity of the task of event log building for non-experts, which addresses Challenge 5.

Also, in Chapter 5 we proposed a framework for case notion discovery and recommendation. This framework provides a method to generate candidate case notions based on the data schema of the original dataset. Also, it makes it possible to build event logs using the aforementioned event log building method. Additionally, the concept of log “interestingness” is introduced, together with a score function. This makes it possible to rank candidate case notions on a dataset based on a predicted “interestingness” score. This framework tackles several relevant challenges. First, it provides the means for multi-perspective event log building (Challenge 4). Also, it automates the process as much as possible in order to improve usability (Challenge 5). Finally, the framework makes it possible to provide suggestions about interesting views on the data as a way to support the user when dealing with large and complex datasets (Challenge 6).

9.1.3 Data Querying

Data querying languages are tools that can prove very valuable at many stages during a process mining project. One of the contributions of this thesis is the data query language DAPOQ-Lang (Chapter 7). This query language, based on the structure proposed in the OpenSLEX meta-model, has been designed to facilitate the task of data querying in the context of process mining. The language provides constructs that enable the user to write complex queries. Such queries would otherwise be difficult to write and interpret when using other query languages such as SQL. DAPOQ-Lang aims at supporting the user when querying event data by means of constructs and functions to express time constraints, data relations, and event types, among others. DAPOQ-Lang can be used as a filtering technique, allowing the user to obtain refined event logs that focus on the parts of the data that are relevant to answer the business question at hand. The combination of our contributions in event log building with the querying capabilities of DAPOQ-Lang enables the user to carry out event log extraction in a question-driven fashion, which directly relates to Challenge 7.

9.2 Limitations and Open Issues

The contributions made in this thesis can be seen as important first steps to speed up the preprocessing of data for process mining. In this section, we acknowledge that additional work is needed and we present the main limitations and open issues of our work.

9.2.1 Data Extraction

In the first part of this thesis, we presented several contributions related to the data extraction phase. The main identified limitations of our work in this area are described below:

- The definition of a target meta-model for data extraction (OpenSLEX) presents many benefits for the analysis phase, but does not solve the data extraction problem. The variety of formats in which data can be found in real-life makes it challenging to design a solution that fits every situation. Although we provided examples of adapters for some wide-spread business information systems, the challenge of automatic data extraction remains open when a new raw-data representation is faced.
- The data extraction method proposed in this thesis assumes the existence of a complete data model, i.e., the relationships between data classes, usually represented by foreign keys in most database systems, must be explicitly defined. However, such relationships are not always present, as observed in Chapter 8. Despite the existence of techniques in the literature that attempt to solve the schema discovery problem, a reliable and efficient solution able to handle complex environments is not available yet.
- The proposed OpenSLEX meta-model has been designed with the purpose of capturing aspects of data typically found in databases and event data stores. Aspects such as data models, classes, objects, attributes, and relationships can be directly mapped to database concepts such as schemas, tables, rows, columns, and foreign keys. However, this can limit the applicability of the proposed solution in scenarios where this mapping is not so straightforward. This is the case for event streams, where the notions of object and class become fuzzy or do not exist.

9.2.2 Event Log Building

The second part of this thesis presents contributions related to the event log building phase. These are the main limitations of the methods and techniques proposed:

- When building event logs from databases, it is rare to find event data readily available. Usually, event definitions need to be discovered or manually defined. The approach proposed in this thesis looks into the data type of object attributes to find event timestamps. Next, it transforms instances of timestamp values into events using the name of the timestamp attribute as the activity name. That is a rather naive approach, since it ignores other attribute values that could represent a much more meaningful activity name, e.g., task name or status.

Moreover, not all timestamp attributes represent the occurrence of an event in the context of business processes, e.g., customer birthday, which introduces noise in the discovered event logs.

- The notion of log “interestingness” proposed in this thesis is somewhat superficial. Only certain structural properties of the log (*level of detail*, *support*, *average number of events per trace*) are taken into account when evaluating event logs. The current notion of log “interestingness” ignores other important aspects such as the relevance of the log semantics at the business level, how meaningful the activities are with respect to the process, as well as the homogeneity of behavior captured in the event log.
- In this thesis, we proposed certain predictors for the event log metrics used to assess log “interestingness”. It has been shown that the resulting ranking based on predicted scores resembles, at an acceptable level of accuracy, the ranking based on the actual metrics. However, the individual predictions for each log metric lack accuracy. Relative assumptions can still be made, e.g., log A has higher support than log B. However, accurate predictions would make the technique more robust to outliers, and benefit the overall quality of the log “interestingness” assessment.

9.2.3 Data Querying

In the third part of this thesis, we presented contributions in the area of data querying applied to process mining. When it comes to querying, we are aware of the following limitations and open issues:

- The query language proposed in this thesis, DAPOQ-Lang, was designed with a focus on event data querying for process mining analysis. The design choices were made with a focus on functionality rather than performance. However, in many real-life scenarios analysts need to handle large and complex datasets. In such cases, the applicability of DAPOQ-Lang could be limited by a lack of performance and scalability in terms of memory and CPU use.
- DAPOQ-Lang is a query language that assumes that the data to query has been transformed into a suitable format, i.e., an SQL store complying with the OpenSLEX data schema. This means that DAPOQ-Lang cannot be used to query data from the original source, but only after extraction and transformation. This can be an issue when dealing with large datasets or when querying data that changes frequently.
- Even though DAPOQ-Lang was designed trying to ease the data querying tasks of analysts, the querying language itself has a steep learning curve, especially for those not familiar with procedural programming languages. The dialect is composed of many functions (57) defined to deal with the elements of the OpenSLEX meta-model, in addition to the general functions and expressions belonging to the host language (Groovy). This means that, even though many

data operations will be easy to perform thanks to DAPOQ-Lang's domain specific functions, making the most our of this query language requires some study and practice.

9.3 Future Work

We list the most promising directions for future work that were identified while carrying out the work presented in this thesis. Again we consider data extraction, event log building, and data querying separately. Also, we propose new directions that do not fit the aforementioned areas.

9.3.1 Data Extraction

With respect to the data extraction phase, we would like to extend our contributions in the following directions:

- One of the limitations of the proposed data extraction method is the need for adapters when dealing with new source data formats and structures. A way to alleviate this issue would be to provide additional adapters, one for each of the most common types of data sources. This would allow the coverage of a large portion of the cases. Ideally, we would like to develop a more general approach. For this, it would be interesting to carry out an extensive study to identify how data are structured in existing information systems. Providing a classification of data source structures would make it possible to develop a collection of generic adapters. This would allow extracting data from new systems by simply choosing the adapter that corresponds to the corresponding underlying data structure.

- The lack of a reliable data schema discovery method is one of the impediments for a fully automated data extraction solution in environments where the data schema is not explicitly defined. Despite the efforts performed by the data-profiling community, further work should be done to reduce both the complexity and the number of false positives when executing the primary and foreign key discovery tasks. It is necessary to provide a reliable and scalable data schema discovery method that can be applied to real-life datasets.

- The OpenSLEX meta-model was designed to be applied in data extraction from structured data sources where notions such as data model, class, and relationship are well defined. However, extraction from less structured sources such as data streams and document repositories can be challenging. It would be of interest to evaluate the suitability of the meta-model to deal with data from non-structured sources.

9.3.2 Event Log Building

We would like to overcome some of the limitations that affect our contributions to event log building by continuing our research as described below:

- One of the biggest challenges in automated event log building is the discovery of event data. Current solutions focus on the presence of timestamps to identify event occurrences. Most of the times they use the name of the timestamp field as the activity name. Further work must be done to automatically identify more meaningful activity names present in the data and to rule out false positives. Statistics about the number of unique values and their relation to the presence of timestamps within the data could be used to identify valid activity names. Also, the use of natural language processing could prove useful to assess if text values in a data attribute represent activities, resources, or life-cycle phase names that could be used to enrich the extracted events and to construct meaningful activity names.

- The notion of log "interestingness" proposed in this thesis is a first attempt at providing an objective score to rank event logs. However, the relation of the proposed "interestingness" metric with respect to a subjective interestingness score provided by users has not been evaluated. A study should be carried out involving real business analysts and domain experts to evaluate the suitability of the metric when applied to different datasets and contexts. Also, this study would be valuable to identify additional measurable aspects that contribute to the notion of log "interestingness" and have not been considered by our definition.

- The accuracy of the proposed predictors for log metrics (*support*, *level of detail*, and *average number of events per trace*) is far from accurate. Finding stricter upper and lower bounds and designing more accurate predictors for each log metric would help to improve the quality of event log "interestingness" rankings and provide better recommendations to the analyst. This could be combined with sampling techniques that combine predicted scores on candidate case notions with actual scores on computed event logs. This would allow to compute event logs only for a limited number of case notions, while increasing ranking quality introducing some certainty in the scores.

- Other approaches propose to avoid data extraction and, instead, to map data sources to a higher level meta-model. The OnProm [20] approach supports the explicit modeling of a domain ontology model that represents the data contained in the relational database at a level of abstraction that is closer to that of the end user. It would be very interesting to explore how OnProm and OpenSLEX can be combined in a way that different data sources can be mapped to the OpenSLEX meta-model. The combination would allow us to benefit from both the generality of onprom, and the semi-automated approaches for case notion discovery and recommendation proposed in this thesis.

9.3.3 Data Querying

We have identified several ways to improve our contributions on data querying by focusing on these aspects:

- Query languages need to keep expanding and evolving as user needs change in order to remain useful and relevant. The case of DAPOQ-Lang is not different. Additional language constructs and functionality can be added to the language to support more process mining scenarios that general query languages do not cover. The decision on which constructs to add to the language should be guided by further user studies.

- An empirical evaluation of the query language by real users is necessary. Such a study should allow as to assess the actual suitability and ease of use of DAPOQ-Lang within the process mining domain. Also, it should help to identify aspects to improve any lacking features that may be critical for its application in real scenarios.

- An important aspect to ensure that a query language is actually useful is its ability to execute complex queries efficiently. Also, improving its scalability is critical in order to handle the ever-growing size of datasets in modern organizations. Efforts should be made to redesign the query planning and execution strategies and the internal memory structures to tackle big data challenges, with the possibility to be integrated within large-scale data processing frameworks such as Apache Spark[1].

9.3.4 Beyond Data Preprocessing

We have identified interesting research lines that deviate from the data preprocessing phase. Some of these options are described below:

- In this thesis, we proposed methods to extract event logs from different data sources. Obtaining event logs became our goal, and we used existing process mining techniques as a tool to analyze the results of our techniques. However, the OpenSLEX meta-model is able to capture more perspectives on data than regular flat event logs. We see a potential benefit, in terms of quality of insights, in the development of process discovery techniques able to analyze event data directly from the OpenSLEX format. New types of analysis would become possible when the data and process perspectives are integrated within one structure.

- In the same line as the previous item, performing process mining analysis directly on the OpenSLEX structure would allow for interesting complex analyses. Transforming event data from databases into regular event logs is a lossy process in which data must be flattened into event, trace, and log attributes. However,

[1]`https://spark.apache.org/`

original data often present many dimensions and transitive relations between different data objects. One possibility would be the verification of complex data compliance rules in the database context. The data perspective integrated within OpenSLEX makes it possible to explore data related to events in a structure that is close to its original form.

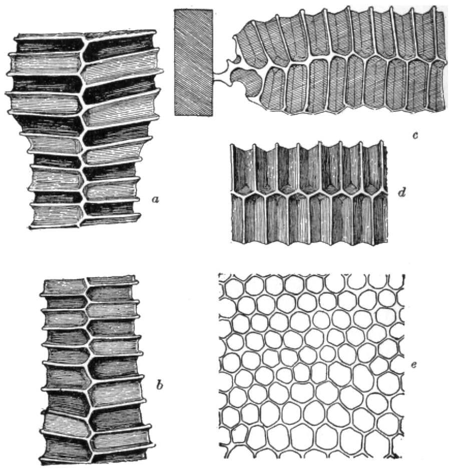
9.4 Reflection

We conclude this thesis by reflecting on our contributions in a broader context. When carrying out a process mining project, a large amount of time and resources are spent during the data extraction and preparation phase. The growing trend in terms of data size and complexity of systems indicate that this problem will remain relevant in the future.

The main motivation to develop the methods and techniques described in the contributions of this thesis is the unavoidable need for automated data extraction and preparation. We believe that efforts must be made in order to shift the time dedicated to process mining projects from the data extraction and preparation phases to focus on the important parts of the analysis. Automating tedious steps as much as possible seems crucial to save time, effort, and cost.

However, the data available in enterprise information systems do not capture the whole picture of an organization. A lot of domain knowledge is still required to make sense of the data and to obtain insights from the results provided by the process mining techniques. One of the aims of process mining is to provide insights to support better decision making. We cannot rely on automated systems to make assumptions and decisions if we provide them with incomplete data. Our contributions do not aim at replacing the role of the business analyst. Instead, they provide tool support to speed up his work and to explore data in innovative ways that were out of reach before, mainly due to time and cost constraints.

We believe that the automation of the different steps involved in a process mining project can only be achieved to a certain extent. Domain knowledge will always be necessary to focus the analysis on relevant aspects and to interpret the results in order to obtain meaningful insights. The goal is to achieve a fluid level of interplay between support tools and domain experts so that data-backed insights can be used to improve the decision making in modern organizations.



Structure of a comb: a, vertical section at top of comb; b, vertical section showing transition from worker to drone cells; c, horizontal section at side of comb showing end-bar frame; d, horizontal section of worker brood cells; e, diagram showing transition cells.

"Beekeeping: a discussion of the life of the honeybee and of the production of honey", Everett Franklin Phillips, 1923

A Mapping from Data Sources to OpenSLEX

In this appendix, we provide extended formalizations of the mapping between the three environments presented in Chapter 4 and the meta-model proposed in Chapter 3.

Section 5.6 presented three environments in which data were extracted from their corresponding databases and transformed to adapt to the structure of our meta-model. Next, the transformed datasets where analyzed in Section 4.3. This section presents a formal description of these three environments, as well as their mapping to the meta-model.

In Section A.1 we formalize the common aspects of any relational database. Also, we provide a definition of how events can be found in the database. The purpose of the section is to provide a common ground. Sections A.2, A.3, and A.4 formalize the aspects that are different for each of the three environments: redo-logs, in-table versioning, and SAP change logs. The next step is to define a mapping from the source definitions of these environments to the OpenSLEX meta-model. The mapped definitions in Section A.5 demonstrate how to transform data from the source database to a format that complies with the OpenSLEX structure provided in Chapter 3.

Figure A.1 shows a diagram of the mapping between the source data and the meta-model elements. We see that some of the source elements need a special mapping depending on the environment. For instance, the source object model (*Valid Source Object Model*) of redo-log and SAP environments share the same structure, while it requires a special mapping in the case of in-table versioning. Each of the blocks depicted in the diagram corresponds to a definition in the coming subsections.

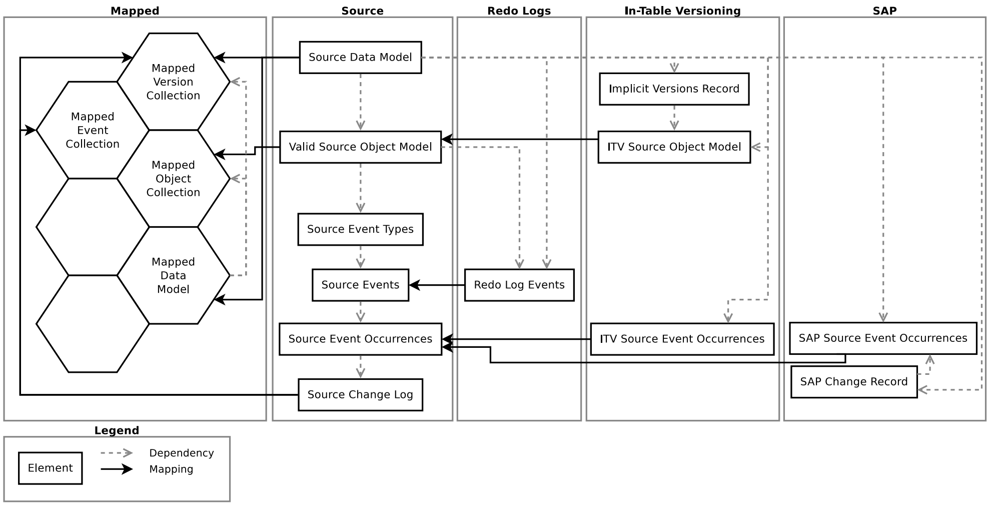


Figure A.1: Mapping of the elements of each of the three environments to the OpenSLEX meta-model.

A.1 Common Definitions to the three Environments

The three environments we are dealing with use a relational database to store all the relevant information. Therefore, the three of them share some characteristics. Some of these characteristics have been introduced in Chapter 2, specifically in Section 2.2. The concepts of *Source Data Model*, *Source Object Model*, and *Valid Source Object Model* are described in definitions 7, 9, and 10 respectively. In this subsection, additional formalizations common to the three environments are presented.

In one form or the other, we find events when extracting our data. Something in common between the events we find in the three mentioned environments is that three types can be distinguished: additions, updates, and deletions. Addition events correspond to row insertions in a table, like *INSERT* operations in an SQL statement executed in a database. Update events represent *UPDATE* operations, where values in a table row are modified. Deletion events correspond to the removal of rows in a table, e.g., the execution of a *DELETE* statement in SQL. For our purposes, we define these three types for each class in the source database. We call these types *source event types*, as explained in Definition 30.

Definition 30 (Source Event Types) *Let* $SDM = (C, A, classAttr, val, PK, FK, keyClass, keyRel, keyAttr, refAttr)$ *be a source data model and VSOM the set of valid source object models.* $SET = ET_{add} \cup ET_{upd} \cup ET_{del}$ *is the set of source event types composed of the following pairwise disjoint sets:*

- $ET_{add} = \{(\oplus, c) \mid c \in C\}$ *are the event types for adding objects,*
- $ET_{upd} = \{(\oslash, c) \mid c \in C\}$ *are the event types for updating objects,*

- $ET_{del} = \{(\ominus, c) \mid c \in C\}$ *are the event types for deleting objects.*

Each of the three environments we are trying to formalize present different characteristics when recording execution events. In general, we say that an event in a database context is represented by an event type (operation performed), a map of old values (old values in the columns of the row affected by the operation), and a map of new values (new values in the same row after the operation). For instance, an event can be represented by an update on the values of a row in a table, capturing the row before and after the change. Definition 31 provides a common, base concept of *source events.* However, in further sections we will see what the characteristics of this concept are in each of the environments.

Definition 31 (Source Events) *Let* $SDM = (C, A, classAttr, val, PK, FK, keyClass, keyRel, keyAttr, refAttr)$ *be a source data model, VSOM the set of valid source object models, and SET the set of source event types. SE is the set of source events such that* $\forall e \in SE : \exists et \in SET : e = (et, map_{old}, map_{new})$.

Finally, something needed for the mapping between these systems and our meta-model is the existence of a concept of *source event occurrence* and *source change log.* Definition 32 provides a description of these concepts. These concepts are an abstraction on different ways to store event occurrences and change logs as observed in each environment. The coming sections discuss these differences and how a change log can be inferred in each case.

Definition 32 (Source Event Occurrence and Source Change Log) *Let* $SDM = (C, A, classAttr, val, PK, FK, keyClass, keyRel, keyAttr, refAttr)$ *be a source data model, VSOM the set of valid source object models and SE the set of source events. Assume some universe of timestamps TS.* $eo = (e, ts) \in SE \times TS$ *is a source event occurrence.* $SEO(SDM, SE) = SE \times TS$ *is the set of all possible source event occurrences. A source change log* $SCL = \langle eo_1, eo_2, ..., eo_n \rangle$ *is a sequence of source event occurrences such that time is non-decreasing, i.e.,* $SCL = \langle eo_1, eo_2, ..., eo_n \rangle \in (SEO(SDM, SE))^*$ *and* $ts_i \leq ts_j$ *for any* $eo_i = (e_i, ts_i)$ *and* $eo_j = (e_j, ts_j)$ *with* $1 \leq i < j \leq n$.

Finally, Definition 33 establishes some useful notations for further definitions, with respect to source event occurrences and source object ids.

Definition 33 (Notation) *Assume a universe of timestamps TS, and a source data model SDM. We define the following shorthand:* $objectId(c, map) = \{(a, v) \in A \times V \mid a \in keyAttr(PK_c) \land map(a) = v\}$, *i.e., it returns a set of pairs* $(attribute, value)$ *for a mapping* map *according to the attributes of the primary key of such class c in the source data model.*

Now, all the common elements of the three environments have been formalized. These represent the common ground needed in order to make the mapping to our meta-model. The three following sections (Section A.2, Section A.3, and Section A.4) formalize some of the concepts that differ between the three environments. Finally, Section A.5 proposes a mapping to our meta-model.

A.2 Database Redo-Logs: Formalization

The redo-log environment presents some particularities with respect to how the events are represented. Definition 34 formalizes this concept while maintaining compatibility with the common description of source events in Definition 31.

Definition 34 (Redo-Log Events) *Let $SDM = (C, A, classAttr, val, PK, FK, keyClass, keyRel, keyAttr, refAttr)$ be a source data model, VSOM the set of valid source object models and $\perp$ the null value. $SE = E_{add} \cup E_{upd} \cup E_{del}$ is the set of source events composed of the following pairwise disjoint sets:*

- $E_{add} = \{((\oplus, c), map_{old}, map_{new})) \mid (c, map_{new}) \in O^{SDM} \wedge \ map_{old} = \perp\}$
- $E_{upd} = \{((\oslash, c), map_{old}, map_{new})) \mid (c, map_{old}) \in O^{SDM} \wedge (c, map_{new}) \in O^{SDM}\}$
- $E_{del} = \{((\ominus, c), map_{old}, map_{new})) \mid (c, map_{old}) \in O^{SDM} \wedge \ map_{new} = \perp\}$

Mainly, this is the only difference between the redo-log environment and the common description provided in the previous section. However, for the other two environments, some additional details need to be taken into account.

A.3 In-Table Versioning: Formalization

In the case of in-table versioning environments, change logs are not explicitly recorded. However, we find object versions implicitly recorded within the tables of the database. Definition 35 formalizes this structure.

Definition 35 (Implicit Versions Record) *Let $SDM = (C, A, classAttr, val, PK, FK, keyClass, keyRel, keyAttr, refAttr)$ be a source data model. An implicit versions record is a tuple $IVR = (OBJECTID, TIMESTAMP, VERID, FNAMES, value)$ such that:*

- *$OBJECTID \subseteq \mathbb{N}$ is a set of object identifiers,*
- *$TIMESTAMP$ is a set of timestamps of versions,*
- *$VERID \subseteq C \times OBJECTID \times TIMESTAMP$ is a set of unique version identifiers formed by the combination of a table name, an object id, and a time stamp,*
- *$FNAMES \subseteq A$ is the set of attributes of the object version,*
- *$value \in (VERID \times A) \not\rightarrow V$ is a mapping between a pair $(versionId, attributeName)$ and its new value after the change.*

Now that we know how object versions are being defined in this specific environment, we can show how it affects our previous definition of *source object model.* Definition 36 shows the compatibility in this particular case.

Definition 36 (ITV Source Object Model) *Given a source data model* $SDM = (C, A, classAttr, val, PK, FK, keyClass, keyRel, keyAttr, refAttr)$ *and an implicit versions record IVR, an ITV source object model SOM is a set of objects such that* $SOM \subseteq O^{SDM}$ *such that:* $\forall o = (c, map) \in SOM : c \in C \wedge \forall a \in domain(map) : a \in FNAMES \wedge \exists v = (tab, ob, ts) \in VERID : map(a) = value(v, a) \wedge \nexists v' = (tab, ob, ts') \in VERID : ts' > ts$, *i.e., the ITV source object model SOM is formed by all the most recent object versions for each object id.*

Definition 37 (Notation) *Let* $SDM = (C, A, classAttr, val, PK, FK, keyClass, keyRel, keyAttr, refAttr)$ *be a source data model, and* $IVR = (OBJECTID, TIMESTAMP, VERID, FNAMES, value)$ *an implicit versions record. Given a version id* $vid = (tab, ob, ts) \in VERID$*, we define the following shorthands:*

- $itvEvType(id)$
- $itvEvType \in VERID \rightarrow \{\oplus, \oslash\}$ *is a function mapping object version ids to types of change, such that* $itvEvType(vid) = \oplus \iff \nexists vid' = (tab', ob', ts') \in VERID : tab' = tab \wedge ob' = ob \wedge ts' < ts$*, i.e., if a previous version of the same object does not exist it is considered to be an addition change. Otherwise,* $itvEvType(vid) = \oslash$*, i.e., it is considered to be an update,*
- $itvTs \in VERID \rightarrow TIMESTAMP$ *is a function mapping object version ids to timestamps such that,* $itvTs(vid) = ts$,
- $itvTabName \in VERID \rightarrow C$ *is a function mapping object version ids to table names such that,* $itvTabName(vid) = tab$,
- $itvObjId \in VERID \rightarrow OBJECTID$ *is a function mapping object version ids to object ids such that,* $itvObjId(vid) = ob$,
- $itvPrevVerId \in VERID \rightarrow (VERID \cup \perp)$ *is a function mapping object version ids to their predecessor object version id (or null id if a predecessor does not exist) such that,* $itvPrevVerId(vid) = vid' \wedge (vid' = (tab', ob', ts') \wedge tab = tab' \wedge ob = ob' \wedge ts > ts' \wedge \nexists (tab', ob', ts'') \in VERID : ts > ts'' > ts') \vee (itvEvType(vid) = \oplus \wedge vid' = \perp)$,

Finally, the only building block left to define in this case is the correlation between the previous definition of source event occurrences (Definition 32) with this environment. Definition 38 shows that equivalence and provides the key to translate implicit object versions into source event occurrences.

Definition 38 (ITV Source Event Occurrences) *Let TS be a universe of timestamps, V a universe of values and SDM a source data model* $SDM = (C, A, classAttr, val, PK, FK, keyClass, keyRel, keyAttr, refAttr)$*. We say that* $SEO(SDM, IVR) = SE \times TS \times V \times V$ *is a set of ITV source event occurrences such that:*
$\forall eo = (((evT, c), map_{old}, map_{new}), ts) \in SEO(SDM, IVR) : \exists id \in VERID:$

- $evT = itvEvType(id) \wedge$
- $c = itvTabName(id) \wedge$

- $(\{itvPrevVerId(id)\} \times domain(map_{old})) \subseteq domain(value) \land$
- $(\{id\} \times domain(map_{new})) \subseteq domain(value) \land$
- $\forall a \in domain(map_{old}) : map_{old}(a) = value(itvPrevVerId(id), a) \land$
- $\forall a \in domain(map_{new}) : map_{new}(a) = value(id, a) \land$
- $ts = itvTs(id) \land$
- $itvObjId(id) = objectId(c, map_{new})$.

That is, for each source event occurrence in $SEO(SDM, IVR)$ exists an implicit version record in IVR that shares values for all its properties.

A.4 SAP-style Change Table: Formalization

SAP systems are a different kind of environment. They are closely related to the redo log environments, with the difference that the change record contains the relevant information in a slightly different way. Definition 39 provides details on this *SAP change record.*

Definition 39 (SAP Change Record) *Assume a universe of values V, a universe of date values DATE and a universe of time values TIME. Given a source data model $SDM = (C, A, classAttr, val, PK, FK, keyClass, keyRel, keyAttr, refAttr)$, a SAP change record is a tuple $SCR = (CHNR, OBJECTID, CHNID, uName, uDate, uTime, change\_ind, tabName, tabKey, fName, value\_new, value\_old)$ such that:*

- *$CHNR \subseteq \mathbb{N}$ is a set of SAP change numbers,*
- *$OBJECTID \subseteq \mathbb{N}$ is a set of SAP object ids,*
- *$CHNID \subseteq CHNR \times C \times OBJECTID$ is a set of unique change identifiers formed by the combination of a change number (CHNR), an object class (C) and an object id (OBJECTID),*
- *$uName \in CHNID \rightarrow V$ is a mapping between change ids and user name strings,*
- *$uDate \in CHNID \rightarrow DATE$ is a mapping between change ids and date values,*
- *$uTime \in CHNID \rightarrow TIME$ is a mapping between change ids and time values,*
- *$change\_ind \in CHNID \rightarrow \{\oplus, \oslash, \ominus\}$ is a mapping between change ids and a change type,*
- *$tabName \in CHNID \rightarrow C$ is a mapping between change ids and a class name,*
- *$tabKey \in CHNID \rightarrow V$ is a mapping between change ids and the primary key of the modified object,*

- $fName \in CHNID \rightarrow \mathscr{P}(A)$ *is a mapping between change ids and a set of changed attributes,*
- $value\_new \in (CHNID \times A) \not\rightarrow V$ *is a mapping between a pair (change id,attribute name) and its new value after the change,*
- $value\_old \in (CHNID \times A) \not\rightarrow V$ *is a mapping between a pair (change id,attribute name) and its old value before the change.*

The previous definition gives us the ground to build the mapping to source event occurrences. Definition 40 describes how to obtain the source event occurrences previously introduced from the SAP change record. This allows inferring the change log necessary to build our meta-model.

Definition 40 (SAP Source Event Occurrences) *Assume a universe of timestamps* TS*, a universe of values* V*, and a function* $convertDateTime \in (DATE \times TIME) \rightarrow TS$ *that maps pairs of date and time values into a timestamp. Given a source data model* $SDM = (C, A, classAttr, val, PK, FK, keyClass, keyRel, keyAttr, refAttr)$ *and a SAP change record* $SCR = (CHNR, OBJECTID, CHNID, uName, uDate, uTime, change\_ind, tabName, tabKey, fName, value\_new, value\_old)$*, we define* $SEO(SDM, SCR) = SE \times TS \times V \times V$ *as a set of source event occurrences such that* $\forall eo = (((evT, c), map_{old}, map_{new}), ts) \in SEO(SDM, SCR) : \exists id \in CHNID$*:*

- $evT = change\_ind(id) \wedge$
- $c = tabName(id) \wedge$
- $map_{old}(fName(id)) = value\_old(fName(id)) \wedge$
- $map_{new}(fName(id)) = value\_new(fName(id)) \wedge$
- $ts = convertDateTime(uDate(id), uTime(id)) \wedge$
- $tabKey(id) = objectId(c, map_{new})$.

That is, for each event occurrence in $SEO(SDM, SCR)$ *exists an event record in* SCR *that shares values for all its properties.*

A.5 Common Meta-Model Mapping for the three Environments

In the previous sections we have defined the common aspects of the three environments under study, together with the characteristics of each of them. At this point, we have defined the necessary notions to map each concept from the original sources to our meta-model. In this section we will define this mapping for each of the main elements of the meta-model, except for the *cases* and *process models* sectors. *These are* *independent from the source data and can be inferred from the extracted information once it has been already mapped to our meta-model.* We will start with the mapping of the *source data model* in Definition 41.

Definition 41 (Mapped Data Model) *Given a source data model* $SDM = (C, A, \mathit{classAttr}, \mathit{val}, \mathit{PK}, \mathit{FK}, \mathit{keyClass}, \mathit{keyRel}, \mathit{keyAttr}, \mathit{refAttr})$*, a mapped data model is a tuple* $MDM = (CL, AT, \mathit{classOfAttribute}, RS, \mathit{sourceClass}, \mathit{targetClass})$ *such that:*

- $CL = C$ *is a set of class names,*
- $AT = A$ *is a set of attribute names,*
- $\mathit{classOfAttribute} \in AT \rightarrow CL$ *is a function that maps each attribute to a class, such that* $\forall at \in AT : at \in \mathit{classAttr}(\mathit{classOfAttribute}(at))$*,*
- $RS = FK$ *is a set of relationship names,*
- $\mathit{sourceClass} \in RS \rightarrow CL$ *is a function that maps each relationship to its source class, such that* $\forall rs \in RS : \mathit{sourceClass}(rs) = \mathit{keyClass}(rs)$*,*
- $\mathit{targetClass} \in RS \rightarrow CL$ *is a function that maps each relationship to its target class, such that* $\forall rs \in RS : \mathit{targetClass}(rs) = \mathit{keyClass}(\mathit{keyRel}(rs))$*.*

The same can be done with the *source object model*. Its mapping is formalized in Definition 42. Also, in Definition 43 we redefine the concept of *timestamps* to include the situations in which the beginning or end of a period are unknown. Next, some useful notations for further definitions are expressed in Definition 44.

Definition 42 (Mapped Object Collection) *Assume* $SOM \in VSOM$ *to be a valid source object model,* MDM *a mapped data model, and* OBJ *the set of all possible objects for* MDM*. A mapped object collection* MOC *is a set of objects such that* $MOC \subseteq OBJ$*, and* $\mathit{mappedObj} \in MOC \leftrightarrow SOM$ *is a bijective function that maps every mapped object to a source object and vice versa.*

Definition 43 (Universe of Global Timestamps) *Assume* TS *to be a universe of timestamps. A universe of global timestamps is a set* GTS *such that* $GTS = TS \cup \{-\infty\} \cup \{+\infty\}$*, where* $-\infty$ *represents an undefined timestamp in the past, and* $+\infty$ *an undefined timestamp in the future. These timestamps fulfill the following condition:* $\forall ts \in TS : -\infty < ts < +\infty$*.*

Definition 44 (Notation) *Assume a universe of timestamps* TS*. Given a source change log* SCL *and a mapped data model* $MDM = (CL, AT, \mathit{classOfAttribute}, RS, \mathit{sourceClass}, \mathit{targetClass})$*, we define the following shorthands:*

- $SCL_a^b(c, oid) = \{eo_i \in SCL \mid eo_i = ((evT, c), map_{old}, map_{new}, t) \wedge b > i > a \wedge \mathit{objectId}(c, map_{new}) = oid\}$*, is the set of event occurrences in the source change log such that all of them occurred after the a-th and before the b-th element of the sequence, and they correspond to objects of class c and with the object id* oid*,*

- $tseo \in \mathscr{P}(SCL) \rightarrow \mathscr{P}(TS)$ *is a function that returns a set of timestamps corresponding to the provided set of event occurrences.*

One of the key elements of this mapping is the version collection. In Definition 45 we use the source change log and object models to infer the content of the versions part of our meta-model.

Definition 45 (Mapped Version Collection) *Assume a universe of global timestamps GTS, a mapped object collection MOC, a source data model* $SDM = (C, A, classAttr, val, PK, FK, keyClass, keyRel, keyAttr, refAttr)$, *a mapped data model* $MDM = (CL, classOfAttribute, AT, RS, sourceClass, targetClass)$, *and a source change log* $SCL = < eo_1, eo_2, ..., eo_n >$.
A mapped object version collection is a tuple $MOVC = (OV, attValue, startTimestamp, endTimestamp, REL)$ *such that:*

- $OV = \{(c, map, t_s, t_e)\}$ *is a set of object versions for which the following holds:*
  - $\forall o = (c, map) \in MOC : \exists ov \in OV : ov = (c, map, t_s, t_e) \wedge t_e = +\phi \wedge t_s = max(tseo(SCL_1^n(c, id)), -\phi)$, *i.e., for every object in the mapped object collection exists a version for which the start timestamp is either unknown or the timestamp of the last source event occurrence that affected that object, and the end timestamp is unknown,*
  - $\forall i \in 1..n : eo_i = ((evT, c), map_{old}, map_{new}, t_{ev}) \in SCL : \exists ov \in OV : ov = (c, map, t_s, t_e) \wedge (((evT = \oplus \vee evT = \oslash) \wedge map_{new} = map \wedge t_s = t_{ev} \wedge t_e = min(ts(SCL_{i+1}^n(c, id)), +\phi)) \vee (evT = \ominus \wedge map_{old} = map \wedge t_e = t_{ev} \wedge t_s = max(ts(SCL_1^{i-1}(c, id)), -\phi)))$, *i.e., for every source event occurrence that represents an addition or a modification, there is an object version with the values after the event occurrence, start timestamp equal to the event occurrence timestamp, and end timestamp equal to the one of the next event occurrence that affected the same object. In case it is the first version of the object, the start timestamp will be unknown. If it is the last version of the object, the end timestamp will be unknown instead.*

- $attValue \in (AT \times OV) \not\rightarrow V$ *is a function that maps values to pairs of attributes and object versions such that, given an attribute* $at \in AT$ *and an object version* $ov = (c, map, t_s, t_e)$, $attValue(at, ov) = map(at)$,

- $startTimestamp \in OV \rightarrow GTS$ *is a function that returns the start timestamp of an object version such that, given an obj. version* $ov = (c, map, t_s, t_e)$, $startTimestamp(ov) = t_s$,

- $endTimestamp \in OV \rightarrow GTS$ *is a function that returns the end timestamp of an object version such that, given an object version* $ov = (c, map, t_s, t_e)$, $endTimestamp(ov) = t_e$,

- $REL \subseteq (RS \times OV \times OV)$ *is a set of triples relating pairs of object versions through specific relationships such that, given a relationship* $rs \in RS$*, and two object versions* $ov_a = (c_a, map_a, t_{sa}, t_{ea}) \in OV$ *and* $ov_b = (c_b, map_b, t_{sb}, t_{eb}) \in OV$*, it is always true that* $(rs, ov_a, ov_b) \in REL \iff rs \in FK \land sourceClass(rs) = c_a \land targetClass(rs) = c_b \land (t_{sa} \leq t_{sb} \leq t_{ea} \lor t_{sb} \leq t_{sa} \leq t_{eb}) \land \forall at \in keyAtt(rs) : map_a(at) = map_b(refAttr(rs, at))$*, i.e., two object versions are related through a relationship if they belong to the source and target classes of the relationship respectively, and if there is a mapped foreign key from the source data model such that both version share the same values for the key attributes. In addition, both versions must have coexisted in time.*

Finally, in Definition 46 we describe how the events of the meta-model are mapped to the source change log.

Definition 46 (Mapped Event Collection) *Assume* V *to be some universe of values,* TS *a universe of timestamps and* SCL *a source change log* $SCL = \langle eo_1, eo_2, ..., eo_n \rangle$*. A mapped event collection is a tuple* $MEC = (EV, EVAT, eventAttributeValue, eventTimestamp)$ *such that:*

- EV *is a set of events,*
- $EVAT$ *is a set of event attribute names,*
- $eventAttributeValue \in (EV \times EVAT) \not\rightarrow V$ *is a function that maps a pair of an event and event attribute name to a value,*
- $eventTimestamp \in EV \rightarrow TS$ *is a function that maps each event to a timestamp.*

And for each event in the mapped event collection, there is an event occurrence in the mapped change log that shares timestamp and attribute values, i.e., $\forall ev \in EV : \exists eo \in SCL : eo = (((evT, c), map_{old}, map_{new}), ts) \land ts = eventTimestamp(ev) \land \forall (ev, at) \in domain(eventAttributeValue) : map_{new}(at) = eventAttributeValue(ev, at)$.

The presented mapping specifies how data are represented in each of the considered environments. Also, it shows how this data can be transformed in order to populate with content our meta-model, with the purpose of performing further analysis in a more standardized and automated way.

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, Nov 2016. (Cited on pages 32, 43, and 45.)

[2] J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, Nov. 1983. (Cited on page 142.)

[3] A. A. Andaloussi, A. Burattin, and B. Weber. Toward an automated labeling of event log attributes. In *Enterprise, Business-Process and Information Systems Modeling*, pages 82–96. Springer, 2018. (Cited on page 107.)

[4] M. A. Assaf, Y. Badr, and Y. Amghar. A continuous query language for stream-based artifacts. In *International Conference on Database and Expert Systems Applications*, pages 80–89. Springer, 2017. (Cited on pages 137 and 138.)

[5] M. Backmann, A. Baumgrass, N. Herzberg, A. Meyer, and M. Weske. Model-driven event query generation for business process monitoring. In *Service-Oriented Computing–ICSOC 2013 Workshops*, pages 406–418. Springer, 2013. (Cited on pages 137 and 138.)

[6] A. V. Baquero and O. Molloy. Integration of event data from heterogeneous systems to support business process analysis. In *Knowledge Discovery, Knowledge Engineering and Knowledge Management*, pages 440–454. Springer, 2012. (Cited on pages 137 and 138.)

[7] D. Bayomie, I. M. Helal, A. Awad, E. Ezat, and A. ElBastawissi. Deducing case ids for unlabeled event logs. In *International Conference on Business Process Management*, pages 242–254. Springer, 2015. (Cited on page 107.)

[8] B. Benatallah, H. R. Motahari-Nezhad, S. Sakr, et al. A query language for analyzing business processes execution. In *Business Process Management*, pages 281–297. Springer, 2011. (Cited on pages 137 and 138.)

[9] A. Bolt and W. M. P. van der Aalst. Multidimensional process mining using process cubes. In *Enterprise, Business-Process and Information Systems Modeling - 16th International Conference, BPMDS 2015, 20th International Conference, EMMSAD 2015, Held at CAiSE 2015, Stockholm, Sweden, June 8-9, 2015, Proceedings*, pages 102–116, 2015. (Cited on page 46.)

[10] A. Bottrighi, L. Canensi, G. Leonardi, S. Montani, and P. Terenziani. Trace retrieval for business process operational support. *Expert Systems with Applications*, 55:212–221, 2016. (Cited on pages 137 and 138.)

[11] S. Bowers, T. McPhillips, B. Ludäscher, S. Cohen, and S. B. Davidson. A model for user-oriented data provenance in pipelined scientific workflows. In *Provenance and Annotation of Data*, pages 133–147. Springer, 2006. (Cited on pages 137 and 138.)

[12] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. (Cited on page 108.)

[13] J. C. A. M. Buijs. Mapping data sources to XES in a generic way. Master's thesis, Technische Universiteit Eindhoven, The Netherlands, 2010. (Cited on page 28.)

[14] P. Buneman, S. Khanna, and T. Wang-Chiew. Why and where: A characterization of data provenance. In *International conference on database theory*, pages 316–330. Springer, 2001. (Cited on page 132.)

[15] A. Burattin and R. Vigo. A framework for semi-automated process instance discovery from decorative attributes. In *Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on*, pages 176–183. IEEE, 2011. (Cited on page 107.)

[16] C. J. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *Advances in neural information processing systems*, pages 193–200, 2007. (Cited on page 108.)

[17] C. J. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96. ACM, 2005. (Cited on page 108.)

[18] D. Calvanese, T. E. Kalayci, M. Montali, and A. Santoso. Obda for log extraction in process mining. In *Reasoning Web International Summer School*, pages 292–345. Springer, 2017. (Cited on page 107.)

[19] D. Calvanese, T. E. Kalayci, M. Montali, and A. Santoso. The onprom toolchain for extracting business process logs using ontology-based data access. In *Proceedings of the BPM Demo Track and BPM Dissertation Award*. CEUR-WS. org, 2017. (Cited on page 107.)

[20] D. Calvanese, T. E. Kalayci, M. Montali, A. Santoso, and W. Van Der Aalst. Conceptual schema transformation in ontology-based data access. In *European Knowledge Acquisition Workshop*, pages 50–67. Springer, 2018. (Cited on page 197.)

[21] D. Calvanese, T. E. Kalayci, M. Montali, and S. Tinella. Ontology-based data access for extracting event logs from legacy data: the onprom tool and methodology. In *International Conference on Business Information Systems*, pages 220–236. Springer, 2017. (Cited on page 107.)

[22] F. Chirigati and J. Freire. Towards integrating workflow and database provenance. In *Provenance and Annotation of Data and Processes*, pages 11–23. Springer, 2012. (Cited on pages 137 and 138.)

[23] F. Costa, V. Silva, D. De Oliveira, K. Ocaña, E. Ogasawara, J. Dias, and M. Mattoso. Capturing and querying workflow runtime provenance with prov: a practical approach. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 282–289. ACM, 2013. (Cited on pages 137 and 138.)

[24] V. Cuevas-Vicenttin, S. Dey, M. L. Y. Wang, T. Song, and B. Ludascher. Modeling and querying scientific workflow provenance in the D-OPM. In *High Performance Computing, Networking, Storage and Analysis (SCC)*, pages 119–128. IEEE, 2012. (Cited on pages 137 and 138.)

[25] S. B. Davidson and J. Freire. Provenance and scientific workflows: challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1345–1350. ACM, 2008. (Cited on page 132.)

[26] D. Deutch and T. Milo. Top-k projection queries for probabilistic business processes. In *Proceedings of the 12th International Conference on Database Theory*. ACM, 2009. (Cited on pages 137 and 138.)

[27] E. Domínguez, B. Pérez, Á. L. Rubio, M. A. Zapata, A. Allué, and A. López. Developing provenance-aware query systems: an occurrence-centric approach. *Knowledge and Information Systems*, 50(2):661–688, 2017. (Cited on pages 137 and 138.)

[28] J. Eder, G. E. Olivotto, and W. Gruber. A data warehouse for workflow logs. In *Engineering and Deployment of Cooperative Information Systems*, pages 1–15. Springer, 2002. (Cited on page 46.)

[29] B. Fazzinga, S. Flesca, F. Furfaro, E. Masciari, L. Pontieri, and C. Pulice. A framework supporting the analysis of process logs stored in either relational or NoSQL DBMSs. In *Foundations of Intelligent Systems*, pages 52–58. Springer, 2015. (Cited on pages 137 and 138.)

[30] D. R. Ferreira and D. Gillblad. Discovering process models from unlabelled event logs. In *International Conference on Business Process Management*, pages 143–158. Springer, 2009. (Cited on page 107.)

[31] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of machine learning research*, 4(Nov):933–969, 2003. (Cited on page 108.)

[32] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001. (Cited on page 108.)

[33] L. M. Gadelha Jr, M. Wilde, M. Mattoso, and I. Foster. MTCProv: a practical provenance query framework for many-task scientific computing. *Distributed and Parallel Databases*, 30(5-6):351–370, 2012. (Cited on pages 137 and 138.)

[34] N. Gehrke and N. Mueller-Wickop. Basic principles of financial process mining a journey through financial data in accounting information systems. In *AMCIS*, page 289, 2010. (Cited on page 50.)

[35] A. Gierth and D. Fetter. Cyclic Tag System http://www.webcitation.org/6Db5tYVpi. Technical report, PostgreSQL wiki, 2011. (Cited on page 162.)

[36] E. González-López de Murillas, J. Fabra, P. Álvarez, and J. Ezpeleta. Parallel computation of the reachability graph of petri net models with semantic information. *Software: Practice and Experience*, 2016. (Cited on page 228.)

[37] E. González López de Murillas, E. Helm, H. A. Reijers, and J. Küng. Audit trails in openslex: Paving the road for process mining in healthcare. In *Information Technology in Bio- and Medical Informatics: 8th International Conference, ITBAM 2017, Lyon, France*, 2017. (Cited on page 228.)

[38] E. González López de Murillas, H. A. Reijers, and G. E. Hoogendoorn. Redo log process mining in real life: Data challenges & opportunities. In *Business Process Management Workshops: BPM 2017 International Workshops, Barcelona, Spain, September 11, 2017.* Springer International Publishing, 2017. (Cited on pages 15, 54, and 228.)

[39] E. González López de Murillas, H. A. Reijers, and W. M. P. van der Aalst. Case notion discovery and recommendation: Automated event log building on databases. Under review. (Cited on pages 15, 53, 59, and 228.)

[40] E. González López de Murillas, H. A. Reijers, and W. M. P. van der Aalst. Connecting databases with process mining: A meta model and toolset. In *International Workshop on Business Process Modeling, Development and Support*, pages 231–249. Springer, 2016. (Cited on pages 15 and 229.)

[41] E. González López de Murillas, H. A. Reijers, and W. M. P. van der Aalst. Everything you always wanted to know about your process, but did not know how to ask. In M. Dumas and M. Fantinato, editors, *Business Process Management Workshops*, pages 296–309, Cham, 2016. Springer International Publishing. (Cited on pages 16, 132, and 228.)

[42] E. González López de Murillas, H. A. Reijers, and W. M. P. van der Aalst. Connecting databases with process mining: a meta model and toolset. *Software & Systems Modeling*, Feb 2018. (Cited on pages 15, 86, 138, 152, 163, and 228.)

[43] E. González López de Murillas, H. A. Reijers, and W. M. P. van der Aalst. *Data-Aware Process Oriented Query Language*, chapter Process Querying Methods. Springer International Publishing AG, 2019. Under Review. (Cited on pages 16 and 228.)

[44] E. González López de Murillas, W. M. P. van der Aalst, and H. A. Reijers. Process mining on databases: Unearthing historical data from redo logs. In *Business Process Management*, pages 367–385. Springer, 2015. (Cited on pages 15, 28, 50, 52, 54, 55, 56, 59, and 229.)

[45] V. Gopalkrishnan, Q. Li, and K. Karlapalem. Star/snow-flake schema driven object-relational data warehouse design and query processing strategies. In *DataWarehousing and Knowledge Discovery*, pages 11–22. Springer, 1999. (Cited on page 81.)

[46] N. Guarino. On the semantics of ongoing and future occurrence identifiers. In *International Conference on Conceptual Modeling*, pages 477–490. Springer, 2017. (Cited on page 46.)

[47] N. Guarino and G. Guizzardi. "We need to discuss the relationship": Revisiting relationships as modeling constructs. In *International Conference on Advanced Information Systems Engineering*, pages 279–294. Springer, 2015. (Cited on page 46.)

[48] N. Guarino and G. Guizzardi. Relationships and events: towards a general theory of reification and truthmaking. In *Conference of the Italian Association for Artificial Intelligence*, pages 237–249. Springer, 2016. (Cited on page 46.)

[49] G. Guizzardi, N. Guarino, and J. P. A. Almeida. Ontological considerations about the representation of events and endurants in business models. In *International Conference on Business Process Management*, pages 20–36. Springer, 2016. (Cited on page 46.)

[50] C. Gunther. *Process Mining in Flexible Environments*. PhD thesis, Eindhoven University of Technology, 2009. (Cited on pages 83, 92, and 107.)

[51] N. Herzberg, A. Meyer, and M. Weske. Improving business process intelligence by observing object state transitions. *Data & Knowledge Engineering*, 98:144–164, 2015. (Cited on page 47.)

[52] B. F. Hompes, J. C. A. M. Buijs, and W. M. P. van der Aalst. A generic framework for context-aware process performance analysis. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 300–317. Springer, 2016. (Cited on pages 113 and 128.)

[53] B. F. Hompes, J. C. A. M. Buijs, W. M. P. van der Aalst, P. Dixit, and J. Buurman. Discovering deviating cases and process variants using trace clustering. In *27th Benelux Conference on Artificial Intelligence, 5-6 November 2015, Hasselt, Belgium*, 11 2015. (Cited on pages 113 and 123.)

[54] X. Huang, Z. Bao, S. B. Davidson, T. Milo, and X. Yuan. Answering regular path queries on workflow provenance. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 375–386. IEEE, 2015. (Cited on pages 137 and 138.)

[55] J. E. Ingvaldsen and J. A. Gulla. Preprocessing support for large scale process mining of SAP transactions. In *Business Process Management Workshops*, pages 30–41. Springer, 2008. (Cited on pages 28, 50, and 107.)

[56] M. Jans and P. Soffer. From relational database to event log: Decisions with quality impact. In *Business Process Management Workshops: BPM 2017 International Workshops, Barcelona, Spain, September 11, 2017*. Springer International Publishing, 2017. (Cited on pages 83, 92, and 106.)

[57] R. Kimball and J. Caserta. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. John Wiley & Sons, 2011. (Cited on page 20.)

[58] R. Koenker and K. F. Hallock. Quantile regression. *Journal of economic perspectives*, 15(4):143–156, 2001. (Cited on page 102.)

[59] P. Koksal, S. N. Arpinar, and A. Dogac. Workflow history management. *ACM Sigmod Record*, 27(1):67–75, 1998. (Cited on pages 137 and 138.)

[60] Y. Lederer Antonucci and R. J. Goeke. Identification of appropriate responsibilities and positions for business process management success: Seeking a valid and reliable framework. *Business process management Journal*, 17(1):127–146, 2011. (Cited on page 154.)

[61] M. Leemans, W. M. P. van der Aalst, and M. G. van den Brand. Recursion aware modeling and discovery for hierarchical software event log analysis. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 185–196. IEEE, 2018. (Cited on page 113.)

[62] S. J. J. Leemans. *Robust process mining with guarantees*. PhD thesis, Eindhoven University of Technology, May 2017. (Cited on pages xv, 5, 6, and 7.)

[63] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst. Discovering block-structured process models from event logs-a constructive approach. In *Application and Theory of Petri Nets and Concurrency*, pages 311–329. Springer, 2013. (Cited on page 163.)

[64] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst. Discovering block-structured process models from event logs containing infrequent behaviour. In *International conference on business process management*, pages 66–78. Springer, 2013. (Cited on pages xv, 23, 24, 113, and 116.)

[65] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst. Process and deviation exploration with inductive visual miner. In *BPM Demo Sessions 2014*, volume 1295, pages 46–50. Springer, 2014. (Cited on page 180.)

[66] G. Li, E. González López de Murillas, R. Medeiros de Carvalho, and W. M. P. van der Aalst. Extracting object-centric event logs to support process mining on databases. In *Information Systems in the Big Data Era*, pages 182–199, Cham, 2018. Springer International Publishing. (Cited on page 228.)

[67] C. Lim, S. Lu, A. Chebotko, and F. Fotouhi. Opql: A first opm-level query language for scientific workflow provenance. In *Services Computing (SCC), 2011 IEEE International Conference on*, pages 136–143. IEEE, 2011. (Cited on pages 137 and 138.)

[68] C. Lim, S. Lu, A. Chebotko, and F. Fotouhi. Storing, reasoning, and querying OPM-compliant scientific workflow provenance using relational databases. *Future Generation Computer Systems*, 27(6):781–789, 2011. (Cited on pages 137 and 138.)

[69] D. Liu. XQuery meets Datalog: Data relevance query for workflow trustworthiness. In *Research Challenges in Information Science (RCIS 2010)*, pages 169–174. IEEE, 2010. (Cited on pages 137 and 138.)

[70] D. Liu, C. Pedrinaci, and J. Domingue. Semantic enabled complex event language for business process monitoring. In *Proceedings of the 4th International Workshop on Semantic Business Process Management*, pages 31–34. ACM, 2009. (Cited on pages 137 and 138.)

[71] X. Lu, D. Fahland, and W. M. P. van der Aalst. Interactively exploring logs and mining models with clustering, filtering, and relabeling. *Proceedings of the BPM 2016 Tool Demonstration Track*, 2016. (Cited on page 113.)

[72] X. Lu, M. Nagelkerke, D. van de Wiel, and D. Fahland. Discovering interacting artifacts from ERP systems. *IEEE Trans. Services Computing*, 8(6):861–873, 2015. (Cited on pages 45 and 85.)

[73] E. R. Mahendrawathi, H. M. Astuti, and I. R. K. Wardhani. Material movement analysis for warehouse business process improvement with process mining: A case study. In *Asia Pacific Business Process Management*, pages 115–127. Springer, 2015. (Cited on page 28.)

[74] F. Mannhardt, M. de Leoni, and H. A. Reijers. The multi-perspective process explorer. In *Proceedings of the BPM Demo Session 2015, Co-located with the 13th International Conference on Business Process Management {(BPM} 2015), Innsbruck, Austria, September 2, 2015*, pages 130–134. CEUR Workshop Proceedings, 2015. (Cited on page 127.)

[75] F. Mannhardt, M. De Leoni, H. A. Reijers, and W. M. P. van der Aalst. Balanced multi-perspective checking of process conformance. *Computing*, 98(4):407–437, 2016. (Cited on pages 113 and 127.)

[76] F. Mannhardt, M. de Leoni, H. A. Reijers, and W. M. P. van der Aalst. Data-driven process discovery-revealing conditional infrequent behavior from event logs. In *International Conference on Advanced Information Systems Engineering*, pages 545–560. Springer, 2017. (Cited on pages 23, 113, and 118.)

[77] R. S. Mans, W. M. P. van der Aalst, R. J. B. Vanwersch, and A. J. Moleman. Process mining in healthcare: Data challenges when answering frequently posed questions. In *Process Support and Knowledge Representation in Health Care*, pages 140–153. Springer, 2013. (Cited on page 154.)

[78] T. Metzke, A. Rogge-Solti, A. Baumgrass, J. Mendling, and M. Weske. Enabling semantic complex event processing in the domain of logistics. In *Service-Oriented Computing–ICSOC 2013 Workshops*, pages 419–431. Springer, 2013. (Cited on pages 137 and 138.)

[79] D. Metzler and W. B. Croft. Linear feature-based models for information retrieval. *Information Retrieval*, 10(3):257–274, 2007. (Cited on page 108.)

[80] A. Meyer, L. Pufahl, D. Fahland, and M. Weske. Modeling and enacting complex data dependencies in business processes. In *Proceedings of the 11th international conference on Business Process Management*, pages 171–186. Springer-Verlag, 2013. (Cited on page 47.)

[81] B. Meyer. *Introduction to the Theory of Programming Languages.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990. (Cited on pages 144 and 146.)

[82] M. Momotko and K. Subieta. Process query language: A way to make workflow processes more flexible. In *Advances in Databases and Information Systems*, pages 306–321. Springer, 2004. (Cited on pages 137 and 138.)

[83] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, et al. The open provenance model core specification (v1. 1). *Future generation computer systems*, 27(6):743–756, 2011. (Cited on page 135.)

[84] H. R. Motahari-Nezhad, R. Saint-Paul, F. Casati, and B. Benatallah. Event correlation for process discovery from web service interaction logs. *The VLDB Journal*, 20(3):417–444, June 2011. (Cited on page 107.)

[85] N. Mueller-Wickop and M. Schultz. ERP event log preprocessing: Timestamps vs. accounting logic. In *Design Science at the Intersection of Physical and Virtual Design*, volume 7939 of *Lecture Notes in Computer Science*, pages 105–119. Springer Berlin Heidelberg, 2013. (Cited on page 28.)

[86] O. Müller, T. Schmiedel, E. Gorbacheva, and J. vom Brocke. Towards a typology of business process management professionals: identifying patterns of competences through latent semantic analysis. *Enterprise IS*, 10(1):50–80, 2016. (Cited on pages xxii, 154, and 155.)

[87] T. Neumuth, S. Mansmann, M. H. Scholl, and O. Burgert. Data warehousing technology for surgical workflow analysis. In *Computer-Based Medical Systems, 2008. CBMS'08. 21st IEEE International Symposium on*, pages 230–235. IEEE, 2008. (Cited on page 46.)

[88] A. Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 78–, New York, NY, USA, 2004. ACM. (Cited on page 108.)

[89] L. Niedrite, D. Solodovnikova, M. Treimanis, and A. Niedritis. Goal-driven design of a data warehouse-based business process analysis system. In *Proceedings of the 6th Conference on 6th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases*, pages 243–249, 2007. (Cited on page 46.)

[90] M. J. Panik. *Advanced statistics from an elementary point of view*, volume 9. Academic Press, 2005. (Cited on page 93.)

[91] J. M. Perez-Alvarez, M. T. Gomez-Lopez, L. Parody, and R. M. Gasca. Process instance query language to include process performance indicators in dmn. In *Enterprise Distributed Object Computing Workshop (EDOCW), 2016 IEEE 20th International*, pages 1–8. IEEE, 2016. (Cited on pages 137 and 138.)

[92] V. Popova, D. Fahland, and M. Dumas. Artifact lifecycle discovery. *International Journal of Cooperative Information Systems*, 24(01):1550001, 2015. (Cited on page 45.)

[93] O. Poppe, S. Giessl, E. A. Rundensteiner, and F. Bry. The HIT model: workflow-aware event stream monitoring. In *Transactions on Large-Scale Data- and Knowledge-Centered Systems XI*, pages 26–50. Springer, 2013. (Cited on pages 137 and 138.)

[94] S. Radeschütz, H. Schwarz, and F. Niedermann. Business impact analysis: a framework for a comprehensive analysis and optimization of business processes. *Computer Science-Research and Development*, 30(1):69–86, 2015. (Cited on pages 137 and 138.)

[95] M. Räim, C. Di Ciccio, F. M. Maggi, M. Mecella, and J. Mendling. Log-based understanding of business processes through temporal logic query checking. In *On the Move to Meaningful Internet Systems: OTM 2014 Conferences*, pages 75–92. Springer, 2014. (Cited on pages 137 and 138.)

[96] M. Ray, M. Liu, E. Rundensteiner, D. J. Dougherty, C. Gupta, S. Wang, A. Mehta, and I. Ari. Optimizing complex sequence pattern extraction using caching. In *Data Engineering Workshops (ICDEW), 2011 IEEE 27th International Conference on*, pages 243–248. IEEE, 2011. (Cited on pages 137 and 138.)

[97] A. Roest. A practitioner's guide for process mining on ERP systems : the case of SAP order to cash. Master's thesis, Technische Universiteit Eindhoven, The Netherlands, 2012. (Cited on page 50.)

[98] M. Rosemann and M. Zur Muehlen. Evaluation of workflow management systems-a meta model approach. *Australian Journal of Information Systems*, 6(1), 1998. (Cited on page 45.)

[99] M. A. Sakka and B. Defude. Towards a scalable semantic provenance management system. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems VII*, pages 96–127. Springer, 2012. (Cited on pages 137 and 138.)

[100] I. Segers. Investigating the application of process mining for auditing purposes. Master's thesis, Technische Universiteit Eindhoven, The Netherlands, 2007. (Cited on page 50.)

[101] Y. Sismanis, P. Brown, P. J. Haas, and B. Reinwald. Gordian: efficient and scalable discovery of composite keys. In *Proceedings of the 32nd international conference on Very large data bases*, pages 691–702. VLDB Endowment, 2006. (Cited on pages 52 and 169.)

[102] M. Solanki and C. Brewster. A knowledge driven approach towards the validation of externally acquired traceability datasets in supply chain business processes. In *Knowledge Engineering and Knowledge Management*, pages 503–518. Springer, 2014. (Cited on pages 137 and 138.)

[103] L. Song, J. Wang, L. Wen, W. Wang, S. Tan, and H. Kong. Querying process models based on the temporal relations between tasks. In *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2011 15th IEEE International*, pages 213–222. IEEE, 2011. (Cited on pages 137 and 138.)

[104] J. Štolfa, M. Kopka, S. Štolfa, O. Koběrskỳ, and V. Snášel. An application of process mining to invoice verification process in sap. In *Innovations in Bio-inspired Computing and Applications*, pages 61–74. Springer, 2014. (Cited on page 28.)

[105] Y. Tang, I. Mackey, and J. Su. Querying workflow logs. *Information*, 9(2):25, 2018. (Cited on pages 137 and 138.)

[106] N. Tax, S. Bockting, and D. Hiemstra. A cross-benchmark comparison of 87 learning to rank methods. *Information Processing & Management*, 51(6):757 – 772, 2015. (Cited on page 104.)

[107] W. M. P. van der Aalst. Process cubes: Slicing, dicing, rolling up and drilling down event data for process mining. In *Asia Pacific Business Process Management - First Asia Pacific Conference, AP-BPM 2013, Beijing, China, August 29-30, 2013. Selected Papers*, pages 1–22, 2013. (Cited on page 46.)

[108] W. M. P. van der Aalst. Extracting event data from databases to unleash process mining. In J. vom Brocke and T. Schmiedel, editors, *BPM - Driving Innovation in a Digital World*, Management for Professionals, pages 105–128. Springer International Publishing, 2015. (Cited on page 54.)

[109] W. M. P. van der Aalst. *Process mining: data science in action*. Springer, 2016. (Cited on pages 4, 21, 22, 23, and 111.)

[110] W. M. P. van der Aalst, A. Adriansyah, A. K. A. de Medeiros, F. Arcieri, et al. *Process Mining Manifesto*, pages 169–194. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. (Cited on pages 8, 9, 11, and 12.)

[111] W. M. P. van der Aalst, A. Adriansyah, and B. F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012. (Cited on pages 6, 24, 113, and 125.)

[112] W. M. P. van der Aalst and M. Song. Mining social networks: Uncovering interaction patterns in business processes. In *International conference on business process management*, pages 244–260. Springer, 2004. (Cited on page 113.)

[113] B. F. van Dongen and W. M. P. van der Aalst. A meta model for process mining data. *Proceedings of the CAiSE'05 Workshops (EMOI-INTEROP Workshop)*, 2:309–320, 2005. (Cited on page 45.)

[114] M. L. van Eck, X. Lu, S. J. J. Leemans, and W. M. P. van der Aalst. PM2: A process mining project methodology. In *International Conference on Advanced Information Systems Engineering*, pages 297–313. Springer, 2015. (Cited on pages xv and 2.)

[115] R. J. B. Vanwersch, K. Shahzad, K. Vanhaecht, P. W. P. J. Grefen, L. M. Pintelon, J. Mendling, G. G. Van Merode, and H. A. Reijers. Methodological support for business process redesign in health care: a literature review protocol. *International Journal of Care Pathways*, 15(4):119–126, 2011. (Cited on page 133.)

[116] H. M. W. Verbeek, J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst. XES, XESame, and ProM 6. In *Information Systems Evolution*, pages 60–75. Springer, 2011. (Cited on page 50.)

[117] T. Vogelgesang and H. Appelrath. PMCube: A data-warehouse-based approach for multidimensional process mining. In *Business Process Management Workshops - BPM 2015, 13th International Workshops, Innsbruck, Austria, August 31 - September 3, 2015, Revised Papers*, pages 167–178, 2015. (Cited on page 46.)

[118] T. Vogelgesang and H. Appelrath. A relational data warehouse for multidimensional process mining. In *Proceedings of the 5th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2015), Vienna, Austria, December 9-11, 2015.*, pages 64–78, 2015. (Cited on page 46.)

[119] T. Vogelgesang, G. Kaes, S. Rinderle-Ma, and H. Appelrath. Multidimensional process mining: Questions, requirements, and limitations. In *Proceedings of*

the CAiSE'16 Forum, at the 28th International Conference on Advanced Information Systems Engineering (CAiSE 2016), Ljubljana, Slovenia, June 13-17, 2016., pages 169–176, 2016. (Cited on page 46.)

[120] M. Walicki and D. R. Ferreira. Sequence partitioning for process mining with unlabeled event logs. *Data & Knowledge Engineering*, 70(10):821–841, 2011. (Cited on page 107.)

[121] H. J. Watson and B. H. Wixom. The current state of business intelligence. *Computer*, 40(9):96–99, Sept 2007. (Cited on page 1.)

[122] A. J. M. M. Weijters, W. M. P. van der Aalst, and A. K. Alves De Medeiros. *Process mining with the HeuristicsMiner algorithm.* BETA publicatie : working papers. Technische Universiteit Eindhoven, 2006. (Cited on pages 23 and 118.)

[123] Q. Wu, C. J. Burges, K. M. Svore, and J. Gao. Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3):254–270, 2010. (Cited on page 108.)

[124] J. Xu and H. Li. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 391–398. ACM, 2007. (Cited on page 108.)

[125] K. Yano, Y. Nomura, and T. Kanai. A practical approach to automated business process discovery. In *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2013 17th IEEE International*, pages 53–62, Sept 2013. (Cited on page 50.)

[126] M. Zhang, M. Hadjieleftheriou, B. C. Ooi, C. M. Procopiuc, and D. Srivastava. On multi-column foreign key discovery. *Proceedings of the VLDB Endowment*, 3(1-2):805–814, 2010. (Cited on pages 52 and 169.)

[127] M. Zur Muehlen. Process-driven management information systems combining data warehouses and workflow technology. In *Proceedings of the International Conference on Electronic Commerce Research (ICECR-4)*, pages 550–566, 2001. (Cited on page 46.)

[128] M. zur Muehlen. Workflow-based process controlling-or: What you can measure you can control. *Workflow handbook*, pages 61–77, 2001. (Cited on page 46.)

[129] M. Zur Muehlen and M. Rosemann. Workflow-based process monitoring and controlling-technical and organizational issues. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, pages 10–pp. IEEE, 2000. (Cited on page 46.)

[130] M. zur Muhlen. Evaluation of workflow management systems using meta models. In *Systems Sciences. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference on*, pages 11–pp. IEEE, 1999. (Cited on page 45.)

Summary

Process Mining on Databases: Extracting Event Data from Real-Life Data Sources

Process mining as a discipline is gaining importance in recent years. Its focus is on the analysis of business processes by means of process execution data. It is a common misconception to think that a process mining project starts as soon as the data are available. We tend to underestimate the time, knowledge, and number of iterations needed to gather relevant data and build an event log suitable for analysis. In many cases, the data extraction and preparation phase can take up to 80% of the project duration. Improving this time-consuming initial phase will have a large impact on the whole process mining project in terms of time, cost, and quality of insights.

In this thesis, we take on several challenges related to data extraction, multi-perspective event log building, and event data querying. We consider environments in which many business processes coexist, sharing data both in a centralized and distributed storage. The aim of this work is twofold. First, to ease the path for practitioners to get started with a process mining project in unknown environments when domain knowledge is not always available. Second, to provide the tools needed for event data querying and to build specialized event logs for a more effective process mining analysis. We propose several techniques that, together, constitute a pipeline connecting databases with existing process mining techniques:

- *A meta-model for process mining on databases* (in Chapter 3). This meta-model aims to standardize the way to capture and structure both data and process aspects of an organization.

- *Data extraction adapters for several scenarios* (in Chapter 4). These adapters developed for some mainstream scenarios allow us to extract and transform data from databases to comply with the proposed meta-model structure.

- A technique to *discover and recommend case notions* in order to *build event logs* for process mining (in Chapter 5). This is achieved by exploring possible views on the data, correlating events by means of the underlying relations extracted from the original database, and assessing their "interestingness" with respect to some metrics.

- A data-aware process oriented query language (in Chapter 7). This query language provides a compact and easy way to express relevant queries in the business context by means of constructs that exploit the underlying meta-model.

The techniques presented in this thesis have been evaluated using representative sample datasets that resemble the most relevant challenges that can be found in real-life environments, as well as case studies with real databases. The results show that these techniques succeed at capturing a more comprehensive picture of the systems under study, improving the quality of the generated event logs, and supporting the user in the process of data extraction and log building. Moreover, they are a step ahead towards the goal of automatic data extraction and process mining analysis of complex IT systems. Implementations of our techniques are publicly available and licensed as open source.

Acknowledgements

First of all, I want to thank my first promotor, Hajo Reijers, for his inestimable supervision and support during the whole project. In good and bad times, you were there to reassure me and to guide me in the right direction. Both your commitment and your warmth make you not only a formidable supervisor, but also a good friend. Second in the list of promotors, but not less important, is Wil van der Aalst. Thank you for your invaluable contribution to this thesis, the constructive discussions, your guidance, and your support in moments of uncertainty.

I must not forget those who introduced me to the research world. Thanks to my master's thesis supervisors, Francisco Javier Fabra Caro and Joaquín Ezpeleta from the University of Zaragoza. Without your help and sponsorship, I would have not had the opportunity to pursue this PhD project.

The defense of this PhD thesis would have not been possible without the external PhD committee members: Avidor Gal, Marco Montali, George Fletcher, and Joaquín Ezpeleta. Thank you all for accepting to be part of the committee and for taking the time to read this PhD thesis.

Special thanks to the secretaries, Ine, Riet and José. You all make the life of everyone at the department much easier, and you do it with a smile. Thanks specially to Ine for being such a nice person. You are always willing to help and even to do what is beyond your duties, like proofreading this thesis. Without people like you, the university would collapse.

I must thank my new colleagues at Accha for giving me the opportunity to be part of a great team. Thank you for your understanding during the last stages of my PhD and for letting me take some time to finalize the thesis. I hope we can build nice things together.

A PhD would be almost impossible to bear without its social component. Thanks to all past and present colleagues and friends at the AIS group and at the TU/e in general. You all made the experience so much more pleasant and enjoyable with the innumerable Friday afternoon drinks, barbecues, borrels, trips, and karaoke nights. A special mention goes to my two paranymphs, Bas and Laura. Thank you for your friendship and for accepting the task. I know you will protect me well if things go south during the defense. And of course, special thanks go to Alfredo. During these years we shared a roof, an office and a lasting friendship. All this would have been very different without you. Come back soon, Europe needs you.

Pursuing a PhD can be quite an absorbing experience. Luckily, we have friends to remind us that there is more in life than research. Whenever I went back to Calahorra, Zaragoza, Madrid, or Barcelona, I had friends I could count on. A special mention goes to the wine, liquor, and piña colada lovers, my little family in Eindhoven. Thank you all for your unconditional friendship, that is what matters the most.

It would have been impossible to obtain this degree without the unconditional love and support of my family: my parents Luis and Rosa, my brother Chelis, my sisters María and Pilar, and all the other members of the family. You always backed me up along the course of my life and studies. From you I learned the most valuable lessons. You all were my best teachers. Thanks to the newcomer too, Mateo. You constantly teach us the importance of every little step.

No hubiera sido posible alcanzar esta meta sin el amor y apoyo incondicional de mi familia: mis padres Luis y Rosa, mi hermano Chelis, mis hermanas María y Pilar, y el resto de miembros de la familia. Siempre me habéis apoyado a lo largo de mi vida y estudios. De vosotros aprendí las lecciones más valiosas. Vosotros fuisteis mis mejores profesores. Gracias también al recién llegado, Mateo. Día a día nos demuestras la importancia de cada pequeño paso.

E ultima, ma non per importanza, Simona. Sei instancabile, esattamente come un'ape, sempre impegnata a fare mille cose: lavorare, fare sport, cucinare, viaggiare, e dimenticarti di alcune cose lungo la strada. E, nonostante tutto, riesci sempre a trovare il tempo per noi. Tu hai fatto si che questa esperienza valesse davvero la pena di essere vissuta. Grazie per essere sempre presente, per avermi supportato quando ne ho avuto bisogno, e per avermi mostrato quanto può essere bella la vita insieme. Grazie Amore.

Eduardo González López de Murillas
Eindhoven, February 2019

Curriculum Vitae

Eduardo González López de Murillas was born on 17th September 1986 and grew up in Calahorra, Spain. In 2011 he received the Master of Science in Computer Science and Engineering at University of Zaragoza, in Zaragoza, Spain. During his studies, he also participated in the Erasmus program and spent the academic year 2009-2010 at Tampere University of Technology, in Tampere, Finland. From 2011 until 2014 he worked as a researcher at University of Zaragoza.

In February 2014, he started a PhD project at Eindhoven University of Technology, in Eindhoven, The Netherlands of which this thesis is the result. Since October 2017 he is working as a machine learning engineer at Precedence B.V. in Eindhoven.

List of Publications

E. González López de Murillas has the following publications:

Book Chapters

- E. González López de Murillas, H. A. Reijers, and W. M. P. van der Aalst. *Data-Aware Process Oriented Query Language*, chapter Process Querying Methods. Springer International Publishing AG, 2019. Under Review

Journals

- E. González López de Murillas, H. A. Reijers, and W. M. P. van der Aalst. Case notion discovery and recommendation: Automated event log building on databases. Under review

- E. González López de Murillas, H. A. Reijers, and W. M. P. van der Aalst. Connecting databases with process mining: a meta model and toolset. *Software & Systems Modeling*, Feb 2018

- E. González-López de Murillas, J. Fabra, P. Álvarez, and J. Ezpeleta. Parallel computation of the reachability graph of petri net models with semantic information. *Software: Practice and Experience*, 2016

Proceedings and Workshop Contributions

- G. Li, E. González López de Murillas, R. Medeiros de Carvalho, and W. M. P. van der Aalst. Extracting object-centric event logs to support process mining on databases. In *Information Systems in the Big Data Era*, pages 182–199, Cham, 2018. Springer International Publishing

- E. González López de Murillas, H. A. Reijers, and G. E. Hoogendoorn. Redo log process mining in real life: Data challenges & opportunities. In *Business Process Management Workshops: BPM 2017 International Workshops, Barcelona, Spain, September 11, 2017*. Springer International Publishing, 2017

- E. González López de Murillas, E. Helm, H. A. Reijers, and J. Küng. Audit trails in openslex: Paving the road for process mining in healthcare. In *Information Technology in Bio- and Medical Informatics: 8th International Conference, ITBAM 2017, Lyon, France*, 2017

- E. González López de Murillas, H. A. Reijers, and W. M. P. van der Aalst. Everything you always wanted to know about your process, but did not know how to ask. In M. Dumas and M. Fantinato, editors, *Business Process Management Workshops*, pages 296–309, Cham, 2016. Springer International Publishing

- E. González López de Murillas, H. A. Reijers, and W. M. P. van der Aalst. Connecting databases with process mining: A meta model and toolset. In *International Workshop on Business Process Modeling, Development and Support*, pages 231–249. Springer, 2016
- E. González López de Murillas, W. M. P. van der Aalst, and H. A. Reijers. Process mining on databases: Unearthing historical data from redo logs. In *Business Process Management*, pages 367–385. Springer, 2015

SIKS dissertations

2011

2011-01 Botond Cseke (RUN), *Variational Algorithms for Bayesian Inference in Latent Gaussian Models.*
2011-02 Nick Tinnemeier(UU), *Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language.*
2011-03 Jan Martijn van der Werf (TUE), *Compositional Design and Verification of Component-Based Information Systems.*
2011-04 Hado van Hasselt (UU), *Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference.*
2011-05 Base van der Raadt (VU), *Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline..*
2011-06 Yiwen Wang (TUE), *Semantically-Enhanced Recommendations in Cultural Heritage.*
2011-07 Yujia Cao (UT), *Multimodal Information Presentation for High Load Human Computer Interaction.*
2011-08 Nieske Vergunst (UU), *BDI-based Generation of Robust Task-Oriented Dialogues.*
2011-09 Tim de Jong (OU), *Contextualised Mobile Media for Learning.*
2011-10 Bart Bogaert (UvT), *Cloud Content Contention.*
2011-11 Dhaval Vyas (UT), *Designing for Awareness: An Experience-focused HCI Perspective.*
2011-12 Carmen Bratosin (TUE), *Grid Architecture for Distributed Process Mining.*
2011-13 Xiaoyu Mao (UvT), *Airport under Control. Multiagent Scheduling for Airport Ground Handling.*
2011-14 Milan Lovric (EUR), *Behavioral Finance and Agent-Based Artificial Markets.*
2011-15 Marijn Koolen (UvA), *The Meaning of Structure: the Value of Link Evidence for Information Retrieval.*
2011-16 Maarten Schadd (UM), *Selective Search in Games of Different Complexity.*
2011-17 Jiyin He (UVA), *Exploring Topic Structure: Coherence, Diversity and Relatedness.*
2011-18 Mark Ponsen (UM), *Strategic Decision-Making in complex games.*
2011-19 Ellen Rusman (OU), *The Mind ' s Eye on Personal Profiles.*
2011-20 Qing Gu (VU), *Guiding service-oriented software engineering - A view-based approach.*
2011-21 Linda Terlouw (TUD), *Modularization and Specification of Service-Oriented Systems.*
2011-22 Junte Zhang (UVA), *System Evaluation of Archival Description and Access.*
2011-23 Wouter Weerkamp (UVA), *Finding People and their Utterances in Social Media.*

2011-24 Herwin van Welbergen (UT), *Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior.*
2011-25 Syed Waqar ul Qounain Jaffry (VU)), *Analysis and Validation of Models for Trust Dynamics.*
2011-26 Matthijs Aart Pontier (VU), *Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots.*
2011-27 Aniel Bhulai (VU), *Dynamic website optimization through autonomous management of design patterns.*
2011-28 Rianne Kaptein(UVA), *Effective Focused Retrieval by Exploiting Query Context and Document Structure.*
2011-29 Faisal Kamiran (TUE), *Discrimination-aware Classification.*
2011-30 Egon van den Broek (UT), *Affective Signal Processing (ASP): Unraveling the mystery of emotions.*
2011-31 Ludo Waltman (EUR), *Computational and Game-Theoretic Approaches for Modeling Bounded Rationality.*
2011-32 Nees-Jan van Eck (EUR), *Methodological Advances in Bibliometric Mapping of Science.*
2011-33 Tom van der Weide (UU), *Arguing to Motivate Decisions.*
2011-34 Paolo Turrini (UU), *Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations.*
2011-35 Maaike Harbers (UU), *Explaining Agent Behavior in Virtual Training.*
2011-36 Erik van der Spek (UU), *Experiments in serious game design: a cognitive approach.*
2011-37 Adriana Burlutiu (RUN), *Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference.*
2011-38 Nyree Lemmens (UM), *Bee-inspired Distributed Optimization.*
2011-39 Joost Westra (UU), *Organizing Adaptation using Agents in Serious Games.*
2011-40 Viktor Clerc (VU), *Architectural Knowledge Management in Global Software Development.*
2011-41 Luan Ibraimi (UT), *Cryptographically Enforced Distributed Data Access Control.*
2011-42 Michal Sindlar (UU), *Explaining Behavior through Mental State Attribution.*
2011-43 Henk van der Schuur (UU), *Process Improvement through Software Operation Knowledge.*
2011-44 Boris Reuderink (UT), *Robust Brain-Computer Interfaces.*
2011-45 Herman Stehouwer (UvT), *Statistical Language Models for Alternative Sequence Selection.*
2011-46 Beibei Hu (TUD), *Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work.*
2011-47 Azizi Bin Ab Aziz(VU), *Exploring Computational Models for Intelligent Support of Persons with Depression.*
2011-48 Mark Ter Maat (UT), *Response Selection and Turn-taking for a Sensitive Artificial Listening Agent.*
2011-49 Andreea Niculescu (UT), *Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality.*

2012

2012-01 Terry Kakeeto (UvT), *Relationship Marketing for SMEs in Uganda.*
2012-02 Muhammad Umair(VU), *Adaptivity, emotion, and Rationality in Human and Ambient Agent Models.*

2012-03 Adam Vanya (VU), *Supporting Architecture Evolution by Mining Software Repositories.*

2012-04 Jurriaan Souer (UU), *Development of Content Management System-based Web Applications.*

2012-05 Marijn Plomp (UU), *Maturing Interorganisational Information Systems.*

2012-06 Wolfgang Reinhardt (OU), *Awareness Support for Knowledge Workers in Research Networks.*

2012-07 Rianne van Lambalgen (VU), *When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions.*

2012-08 Gerben de Vries (UVA), *Kernel Methods for Vessel Trajectories.*

2012-09 Ricardo Neisse (UT), *Trust and Privacy Management Support for Context-Aware Service Platforms.*

2012-10 David Smits (TUE), *Towards a Generic Distributed Adaptive Hypermedia Environment.*

2012-11 J.C.B. Rantham Prabhakara (TUE), *Process Mining in the Large: Preprocessing, Discovery, and Diagnostics.*

2012-12 Kees van der Sluijs (TUE), *Model Driven Design and Data Integration in Semantic Web Information Systems.*

2012-13 Suleman Shahid (UvT), *Fun and Face: Exploring non-verbal expressions of emotion during playful interactions.*

2012-14 Evgeny Knutov(TUE), *Generic Adaptation Framework for Unifying Adaptive Web-based Systems.*

2012-15 Natalie van der Wal (VU), *Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes..*

2012-16 Fiemke Both (VU), *Helping people by understanding them - Ambient Agents supporting task execution and depression treatment.*

2012-17 Amal Elgammal (UvT), *Towards a Comprehensive Framework for Business Process Compliance.*

2012-18 Eltjo Poort (VU), *Improving Solution Architecting Practices.*

2012-19 Helen Schonenberg (TUE), *What's Next? Operational Support for Business Process Execution.*

2012-20 Ali Bahramisharif (RUN), *Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing.*

2012-21 Roberto Cornacchia (TUD), *Querying Sparse Matrices for Information Retrieval.*

2012-22 Thijs Vis (UvT), *Intelligence, politie en veiligheidsdienst: verenigbare grootheden?.*

2012-23 Christian Muehl (UT), *Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction.*

2012-24 Laurens van der Werff (UT), *Evaluation of Noisy Transcripts for Spoken Document Retrieval.*

2012-25 Silja Eckartz (UT), *Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application.*

2012-26 Emile de Maat (UVA), *Making Sense of Legal Text.*

2012-27 Hayrettin Gurkok (UT), *Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games.*

2012-28 Nancy Pascall (UvT), *Engendering Technology Empowering Women.*

2012-29 Almer Tigelaar (UT), *Peer-to-Peer Information Retrieval.*

2012-30 Alina Pommeranz (TUD), *Designing Human-Centered Systems for Reflective Decision Making.*

2012-31 Emily Bagarukayo (RUN), *A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure.*

2012-32 Wietske Visser (TUD), *Qualitative multi-criteria preference representation and reasoning.*
2012-33 Rory Sie (OUN), *Coalitions in Cooperation Networks (COCOON).*
2012-34 Pavol Jancura (RUN), *Evolutionary analysis in PPI networks and applications.*
2012-35 Evert Haasdijk (VU), *Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics.*
2012-36 Denis Ssebugwawo (RUN), *Analysis and Evaluation of Collaborative Modeling Processes.*
2012-37 Agnes Nakakawa (RUN), *A Collaboration Process for Enterprise Architecture Creation.*
2012-38 Selmar Smit (VU), *Parameter Tuning and Scientific Testing in Evolutionary Algorithms.*
2012-39 Hassan Fatemi (UT), *Risk-aware design of value and coordination networks.*
2012-40 Agus Gunawan (UvT), *Information Access for SMEs in Indonesia.*
2012-41 Sebastian Kelle (OU), *Game Design Patterns for Learning.*
2012-42 Dominique Verpoorten (OU), *Reflection Amplifiers in self-regulated Learning.*
2012-43 Withdrawn, .
2012-44 Anna Tordai (VU), *On Combining Alignment Techniques.*
2012-45 Benedikt Kratz (UvT), *A Model and Language for Business-aware Transactions.*
2012-46 Simon Carter (UVA), *Exploration and Exploitation of Multilingual Data for Statistical Machine Translation.*
2012-47 Manos Tsagkias (UVA), *Mining Social Media: Tracking Content and Predicting Behavior.*
2012-48 Jorn Bakker (TUE), *Handling Abrupt Changes in Evolving Time-series Data.*
2012-49 Michael Kaisers (UM), *Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions.*
2012-50 Steven van Kervel (TUD), *Ontologogy driven Enterprise Information Systems Engineering.*
2012-51 Jeroen de Jong (TUD), *Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching.*

2013

2013-01 Viorel Milea (EUR), *News Analytics for Financial Decision Support.*
2013-02 Erietta Liarou (CWI), *MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing.*
2013-03 Szymon Klarman (VU), *Reasoning with Contexts in Description Logics.*
2013-04 Chetan Yadati(TUD), *Coordinating autonomous planning and scheduling.*
2013-05 Dulce Pumareja (UT), *Groupware Requirements Evolutions Patterns.*
2013-06 Romulo Goncalves(CWI), *The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience.*
2013-07 Giel van Lankveld (UvT), *Quantifying Individual Player Differences.*
2013-08 Robbert-Jan Merk(VU), *Making enemies: cognitive modeling for opponent agents in fighter pilot simulators.*
2013-09 Fabio Gori (RUN), *Metagenomic Data Analysis: Computational Methods and Applications.*
2013-10 Jeewanie Jayasinghe Arachchige (UvT), *A Unified Modeling Framework for Service Design..*
2013-11 Evangelos Pournaras(TUD), *Multi-level Reconfigurable Self-*

organization in Overlay Services.

2013-12 Marian Razavian(VU), *Knowledge-driven Migration to Services.*

2013-13 Mohammad Safiri(UT), *Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly.*

2013-14 Jafar Tanha (UVA), *Ensemble Approaches to Semi-Supervised Learning Learning.*

2013-15 Daniel Hennes (UM), *Multiagent Learning - Dynamic Games and Applications.*

2013-16 Eric Kok (UU), *Exploring the practical benefits of argumentation in multi-agent deliberation.*

2013-17 Koen Kok (VU), *The PowerMatcher: Smart Coordination for the Smart Electricity Grid.*

2013-18 Jeroen Janssens (UvT), *Outlier Selection and One-Class Classification.*

2013-19 Renze Steenhuizen (TUD), *Coordinated Multi-Agent Planning and Scheduling.*

2013-20 Katja Hofmann (UvA), *Fast and Reliable Online Learning to Rank for Information Retrieval.*

2013-21 Sander Wubben (UvT), *Text-to-text generation by monolingual machine translation.*

2013-22 Tom Claassen (RUN), *Causal Discovery and Logic.*

2013-23 Patricio de Alencar Silva(UvT), *Value Activity Monitoring.*

2013-24 Haitham Bou Ammar (UM), *Automated Transfer in Reinforcement Learning.*

2013-25 Agnieszka Anna Latoszek-Berendsen (UM), *Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System.*

2013-26 Alireza Zarghami (UT), *Architectural Support for Dynamic Homecare Service Provisioning.*

2013-27 Mohammad Huq (UT), *Inference-based Framework Managing Data Provenance.*

2013-28 Frans van der Sluis (UT), *When Complexity becomes Interesting: An Inquiry into the Information eXperience.*

2013-29 Iwan de Kok (UT), *Listening Heads.*

2013-30 Joyce Nakatumba (TUE), *Resource-Aware Business Process Management: Analysis and Support.*

2013-31 Dinh Khoa Nguyen (UvT), *Blueprint Model and Language for Engineering Cloud Applications.*

2013-32 Kamakshi Rajagopal (OUN), *Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development.*

2013-33 Qi Gao (TUD), *User Modeling and Personalization in the Microblogging Sphere.*

2013-34 Kien Tjin-Kam-Jet (UT), *Distributed Deep Web Search.*

2013-35 Abdallah El Ali (UvA), *Minimal Mobile Human Computer Interaction.*

2013-36 Than Lam Hoang (TUe), *Pattern Mining in Data Streams.*

2013-37 Dirk Börner (OUN), *Ambient Learning Displays.*

2013-38 Eelco den Heijer (VU), *Autonomous Evolutionary Art.*

2013-39 Joop de Jong (TUD), *A Method for Enterprise Ontology based Design of Enterprise Information Systems.*

2013-40 Pim Nijssen (UM), *Monte-Carlo Tree Search for Multi-Player Games.*

2013-41 Jochem Liem (UVA), *Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning.*

2013-42 Léon Planken (TUD), *Algorithms for Simple Temporal Reasoning.*

2013-43 Marc Bron (UVA), *Exploration and Contextualization through Interaction and Concepts.*

2014

2014-01 Nicola Barile (UU), *Studies in Learning Monotone Models from Data.*
2014-02 Fiona Tuliyano (RUN), *Combining System Dynamics with a Domain Modeling Method.*
2014-03 Sergio Raul Duarte Torres (UT), *Information Retrieval for Children: Search Behavior and Solutions.*
2014-04 Hanna Jochmann-Mannak (UT), *Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation.*
2014-05 Jurriaan van Reijsen (UU), *Knowledge Perspectives on Advancing Dynamic Capability.*
2014-06 Damian Tamburri (VU), *Supporting Networked Software Development.*
2014-07 Arya Adriansyah (TUE), *Aligning Observed and Modeled Behavior.*
2014-08 Samur Araujo (TUD), *Data Integration over Distributed and Heterogeneous Data Endpoints.*
2014-09 Philip Jackson (UvT), *Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language.*
2014-10 Ivan Salvador Razo Zapata (VU), *Service Value Networks.*
2014-11 Janneke van der Zwaan (TUD), *An Empathic Virtual Buddy for Social Support.*
2014-12 Willem van Willigen (VU), *Look Ma, No Hands: Aspects of Autonomous Vehicle Control.*
2014-13 Arlette van Wissen (VU), *Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains.*
2014-14 Yangyang Shi (TUD), *Language Models With Meta-information.*
2014-15 Natalya Mogles (VU), *Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare.*
2014-16 Krystyna Milian (VU), *Supporting trial recruitment and design by automatically interpreting eligibility criteria.*
2014-17 Kathrin Dentler (VU), *Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability.*
2014-18 Mattijs Ghijsen (VU), *Methods and Models for the Design and Study of Dynamic Agent Organizations.*
2014-19 Vincius Ramos (TUE), *Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support.*
2014-20 Mena Habib (UT), *Named Entity Extraction and Disambiguation for Informal Text: The Missing Link.*
2014-21 Kassidy Clark (TUD), *Negotiation and Monitoring in Open Environments.*
2014-22 Marieke Peeters (UT), *Personalized Educational Games - Developing agent-supported scenario-based training.*
2014-23 Eleftherios Sidirourgos (UvA / CWI), *Space Efficient Indexes for the Big Data Era.*
2014-24 Davide Ceolin (VU), *Trusting Semi-structured Web Data.*
2014-25 Martijn Lappenschaar (RUN), *New network models for the analysis of disease interaction.*
2014-26 Tim Baarslag (TUD), *What to Bid and When to Stop.*
2014-27 Rui Jorge Almeida (EUR), *Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty.*
2014-28 Anna Chmielowiec (VU), *Decentralized k-Clique Matching.*

2014-29 Jaap Kabbedijk (UU), *Variability in Multi-Tenant Enterprise Software.*
2014-30 Peter de Kock Berenschot (UvT), *Anticipating Criminal Behaviour.*
2014-31 Leo van Moergestel (UU), *Agent Technology in Agile Multiparallel Manufacturing and Product Support.*
2014-32 Naser Ayat (UVA), *On Entity Resolution in Probabilistic Data.*
2014-33 Tesfa Tegegne Asfaw (RUN), *Service Discovery in eHealth.*
2014-34 Christina Manteli (VU), *The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.*
2014-35 Joost van Oijen (UU), *Cognitive Agents in Virtual Worlds: A Middleware Design Approach.*
2014-36 Joos Buijs (TUE), *Flexible Evolutionary Algorithms for Mining Structured Process Models.*
2014-37 Maral Dadvar (UT), *Experts and Machines United Against Cyberbullying.*
2014-38 Danny Plass-Oude Bos (UT), *Making brain-computer interfaces better: improving usability through post-processing..*
2014-39 Jasmina Maric (UvT), *Web Communities, Immigration, and Social Capital.*
2014-40 Walter Omona (RUN), *A Framework for Knowledge Management Using ICT in Higher Education.*
2014-41 Frederic Hogenboom (EUR), *Automated Detection of Financial Events in News Text.*
2014-42 Carsten Eijckhof (CWI/TUD), *Contextual Multidimensional Relevance Models.*
2014-43 Kevin Vlaanderen (UU), *Supporting Process Improvement using Method Increments.*
2014-44 Paulien Meesters (UvT), *Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden..*
2014-45 Birgit Schmitz (OUN), *Mobile Games for Learning: A Pattern-Based Approach.*
2014-46 Ke Tao (TUD), *Social Web Data Analytics: Relevance, Redundancy, Diversity.*
2014-47 Shangsong Liang (UVA), *Fusion and Diversification in Information Retrieval.*

2015

2015-01 Niels Netten (UvA), *Machine Learning for Relevance of Information in Crisis Response.*
2015-02 Faiza Bukhsh (UvT), *Smart auditing: Innovative Compliance Checking in Customs Controls.*
2015-03 Twan van Laarhoven (RUN), *Machine learning for network data.*
2015-04 Howard Spoelstra (OUN), *Collaborations in Open Learning Environments.*
2015-05 Christoph Bösch (UT), *Cryptographically Enforced Search Pattern Hiding.*
2015-06 Farideh Heidari (TUD), *Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes.*
2015-07 Maria-Hendrike Peetz (UvA), *Time-Aware Online Reputation Analysis.*
2015-08 Jie Jiang (TUD), *Organizational Compliance: An agent-based model for designing and evaluating organizational interactions.*
2015-09 Randy Klaassen (UT), *HCI Perspectives on Behavior Change Support Systems.*
2015-10 Henry Hermans (OUN), *OpenU: design of an integrated system to support*

lifelong learning.
2015-11 Yongming Luo (TUE), *Designing algorithms for big graph datasets: A study of computing bisimulation and joins.*
2015-12 Julie M. Birkholz (VU), *Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks.*
2015-13 Giuseppe Procaccianti (VU), *Energy-Efficient Software.*
2015-14 Bart van Straalen (UT), *A cognitive approach to modeling bad news conversations.*
2015-15 Klaas Andries de Graaf (VU), *Ontology-based Software Architecture Documentation.*
2015-16 Changyun Wei (UT), *Cognitive Coordination for Cooperative Multi-Robot Teamwork.*
2015-17 André van Cleeff (UT), *Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs.*
2015-18 Holger Pirk (CWI), *Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories.*
2015-19 Bernardo Tabuenca (OUN), *Ubiquitous Technology for Lifelong Learners.*
2015-20 Lois Vanhée (UU), *Using Culture and Values to Support Flexible Coordination.*
2015-21 Sibren Fetter (OUN), *Using Peer-Support to Expand and Stabilize Online Learning.*
2015-22 Zhemin Zhu (UT), *Co-occurrence Rate Networks.*
2015-23 Luit Gazendam (VU), *Cataloguer Support in Cultural Heritage.*
2015-24 Richard Berendsen (UVA), *Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation.*
2015-25 Steven Woudenberg (UU), *Bayesian Tools for Early Disease Detection.*
2015-26 Alexander Hogenboom (EUR), *Sentiment Analysis of Text Guided by Semantics and Structure.*
2015-27 Sándor Héman (CWI), *Updating compressed colomn stores.*
2015-28 Janet Bagorogoza (TiU), *Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO.*
2015-29 Hendrik Baier (UM), *Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains.*
2015-30 Kiavash Bahreini (OU), *Real-time Multimodal Emotion Recognition in E-Learning.*
2015-31 Yakup Koç (TUD), *On the robustness of Power Grids.*
2015-32 Jerome Gard (UL), *Corporate Venture Management in SMEs.*
2015-33 Frederik Schadd (TUD), *Ontology Mapping with Auxiliary Resources.*
2015-34 Victor de Graaf (UT), *Gesocial Recommender Systems.*
2015-35 Jungxao Xu (TUD), *Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction.*

2016

2016-01 Syed Saiden Abbas (RUN), *Recognition of Shapes by Humans and Machines.*
2016-02 Michiel Christiaan Meulendijk (UU), *Optimizing medication reviews through decision support: prescribing a better pill to swallow.*
2016-03 Maya Sappelli (RUN), *Knowledge Work in Context: User Centered Knowledge Worker Support.*
2016-04 Laurens Rietveld (VU), *Publishing and Consuming Linked Data.*
2016-05 Evgeny Sherkhonov (UVA), *Expanded Acyclic Queries: Containment*

and an Application in Explaining Missing Answers.
2016-06 Michel Wilson (TUD), *Robust scheduling in an uncertain environment.*
2016-07 Jeroen de Man (VU), *Measuring and modeling negative emotions for virtual training.*
2016-08 Matje van de Camp (TiU), *A Link to the Past: Constructing Historical Social Networks from Unstructured Data.*
2016-09 Archana Nottamkandath (VU), *Trusting Crowdsourced Information on Cultural Artefacts.*
2016-10 George Karafotias (VUA), *Parameter Control for Evolutionary Algorithms.*
2016-11 Anne Schuth (UVA), *Search Engines that Learn from Their Users.*
2016-12 Max Knobbout (UU), *Logics for Modelling and Verifying Normative Multi-Agent Systems.*
2016-13 Nana Baah Gyan (VU), *The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach.*
2016-14 Ravi Khadka (UU), *Revisiting Legacy Software System Modernization.*
2016-15 Steffen Michels (RUN), *Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments.*
2016-16 Guangliang Li (UVA), *Socially Intelligent Autonomous Agents that Learn from Human Reward.*
2016-17 Berend Weel (VU), *Towards Embodied Evolution of Robot Organisms.*
2016-18 Albert Meroño Peñuela (VU), *Refining Statistical Data on the Web.*
2016-19 Julia Efremova (Tu/e), *Mining Social Structures from Genealogical Data.*
2016-20 Daan Odijk (UVA), *Context & Semantics in News & Web Search.*
2016-21 Alejandro Moreno Célleri (UT), *From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground.*
2016-22 Grace Lewis (VU), *Software Architecture Strategies for Cyber-Foraging Systems.*
2016-23 Fei Cai (UVA), *Query Auto Completion in Information Retrieval.*
2016-24 Brend Wanders (UT), *Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach.*
2016-25 Julia Kiseleva (TU/e), *Using Contextual Information to Understand Searching and Browsing Behavior.*
2016-26 Dilhan Thilakarathne (VU), *In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains.*
2016-27 Wen Li (TUD), *Understanding Geo-spatial Information on Social Media.*
2016-28 Mingxin Zhang (TUD), *Large-scale Agent-based Social Simulation - A study on epidemic prediction and control.*
2016-29 Nicolas Höning (TUD), *Peak reduction in decentralised electricity systems - Markets and prices for flexible planning.*
2016-30 Ruud Mattheij (UvT), *The Eyes Have It.*
2016-31 Mohammad Khelghati (UT), *Deep web content monitoring.*
2016-32 Eelco Vriezekolk (UT), *Assessing Telecommunication Service Availability Risks for Crisis Organisations.*
2016-33 Peter Bloem (UVA), *Single Sample Statistics, exercises in learning from just one example.*
2016-34 Dennis Schunselaar (TUE), *Configurable Process Trees: Elicitation, Analysis, and Enactment.*
2016-35 Zhaochun Ren (UVA), *Monitoring Social Media: Summarization, Classification and Recommendation.*
2016-36 Daphne Karreman (UT), *Beyond*

R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies.

2016-37 Giovanni Sileno (UvA), *Aligning Law and Action - a conceptual and computational inquiry.*

2016-38 Andrea Minuto (UT), *Materials that Matter - Smart Materials meet Art & Interaction Design.*

2016-39 Merijn Bruijnes (UT), *Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect.*

2016-40 Christian Detweiler (TUD), *Accounting for Values in Design.*

2016-41 Thomas King (TUD), *Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance.*

2016-42 Spyros Martzoukos (UVA), *Combinatorial and Compositional Aspects of Bilingual Aligned Corpora.*

2016-43 Saskia Koldijk (RUN), *Context-Aware Support for Stress Self-Management: From Theory to Practice.*

2016-44 Thibault Sellam (UVA), *Automatic Assistants for Database Exploration.*

2016-45 Bram van de Laar (UT), *Experiencing Brain-Computer Interface Control.*

2016-46 Jorge Gallego Perez (UT), *Robots to Make you Happy.*

2016-47 Christina Weber (UL), *Real-time foresight - Preparedness for dynamic innovation networks.*

2016-48 Tanja Buttler (TUD), *Collecting Lessons Learned.*

2016-49 Gleb Polevoy (TUD), *Participation and Interaction in Projects. A Game-Theoretic Analysis.*

2016-50 Yan Wang (UVT), *The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains.*

2017

2017-01 Jan-Jaap Oerlemans (UL), *Investigating Cybercrime.*

2017-02 Sjoerd Timmer (UU), *Designing and Understanding Forensic Bayesian Networks using Argumentation.*

2017-03 Daniël Harold Telgen (UU), *Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines.*

2017-04 Mrunal Gawade (CWI), *Multi-core Parallelism in a Column-store.*

2017-05 Mahdieh Shadi (UVA), *Collaboration Behavior.*

2017-06 Damir Vandic (EUR), *Intelligent Information Systems for Web Product Search.*

2017-07 Roel Bertens (UU), *Insight in Information: from Abstract to Anomaly.*

2017-08 Rob Konijn (VU), *Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery.*

2017-09 Dong Nguyen (UT), *Text as Social and Cultural Data: A Computational Perspective on Variation in Text.*

2017-10 Robby van Delden (UT), *(Steering) Interactive Play Behavior.*

2017-11 Florian Kunneman (RUN), *Modelling patterns of time and emotion in Twitter #anticipointment.*

2017-12 Sander Leemans (TUE), *Robust Process Mining with Guarantees.*

2017-13 Gijs Huisman (UT), *Social Touch Technology - Extending the reach of social touch through haptic technology.*

2017-14 Shoshannah Tekofsky (UvT), *You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior.*

2017-15 Peter Berck (RUN), *Memory-*

Based Text Correction.

2017-16 Aleksandr Chuklin (UVA), *Understanding and Modeling Users of Modern Search Engines.*

2017-17 Daniel Dimov (UL), *Crowdsourced Online Dispute Resolution.*

2017-18 Ridho Reinanda (UVA), *Entity Associations for Search.*

2017-19 Jeroen Vuurens (UT), *Proximity of Terms, Texts and Semantic Vectors in Information Retrieval.*

2017-20 Mohammadbashir Sedighi (TUD), *Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility.*

2017-21 Jeroen Linssen (UT), *Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds).*

2017-22 Sara Magliacane (VU), *Logics for causal inference under uncertainty.*

2017-23 David Graus (UVA), *Entities of Interest — Discovery in Digital Traces.*

2017-24 Chang Wang (TUD), *Use of Affordances for Efficient Robot Learning.*

2017-25 Veruska Zamborlini (VU), *Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search.*

2017-26 Merel Jung (UT), *Socially intelligent robots that understand and respond to human touch.*

2017-27 Michiel Joosse (UT), *Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors.*

2017-28 John Klein (VU), *Architecture Practices for Complex Contexts.*

2017-29 Adel Alhuraibi (UvT), *From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT".*

2017-30 Wilma Latuny (UvT), *The Power of Facial Expressions.*

2017-31 Ben Ruijl (UL), *Advances in computational methods for QFT calculations.*

2017-32 Thaer Samar (RUN), *Access to and Retrievability of Content in Web Archives.*

2017-33 Brigit van Loggem (OU), *Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity.*

2017-34 Maren Scheffel (OU), *The Evaluation Framework for Learning Analytics.*

2017-35 Martine de Vos (VU), *Interpreting natural science spreadsheets.*

2017-36 Yuanhao Guo (UL), *Shape Analysis for Phenotype Characterisation from High-throughput Imaging.*

2017-37 Alejandro Montes Garcia (TUE), *WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy.*

2017-38 Alex Kayal (TUD), *Normative Social Applications.*

2017-39 Sara Ahmadi (RUN), *Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR .*

2017-40 Altaf Hussain Abro (VUA), *Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems .*

2017-41 Adnan Manzoor (VUA), *Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle.*

2017-42 Elena Sokolova (RUN), *Causal discovery from mixed and missing data with applications on ADHD datasets.*

2017-43 Maaike de Boer (RUN), *Semantic Mapping in Video Retrieval.*

2017-44 Garm Lucassen (UU), *Understanding User Stories - Computational Linguistics in Agile Requirements Engineering.*

2017-45 Bas Testerink (UU), *Decentralized Runtime Norm Enforcement.*
2017-46 Jan Schneider (OU), *Sensor-based Learning Support.*
2017-47 Jie Yang (TUD), *Crowd Knowledge Creation Acceleration.*
2017-48 Angel Suarez (OU), *Collaborative inquiry-based learning.*

2018

2018-01 Han van der Aa (VUA), *Comparing and Aligning Process Representations.*
2018-02 Felix Mannhardt (TUE), *Multi-perspective Process Mining.*
2018-03 Steven Bosems (UT), *Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction.*
2018-04 Jordan Janeiro (TUD), *Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks.*
2018-05 Hugo Huurdeman (UVA), *Supporting the Complex Dynamics of the Information Seeking Process.*
2018-06 Dan Ionita (UT), *Model-Driven Information Security Risk Assessment of Socio-Technical Systems.*
2018-07 Jieting Luo (UU), *A formal account of opportunism in multi-agent systems.*
2018-08 Rick Smetsers (RUN), *Advances in Model Learning for Software Systems.*
2018-09 Xu Xie (TUD), *Data Assimilation in Discrete Event Simulations.*
2018-10 Julienka Mollee (VUA), *Moving forward: supporting physical activity behavior change through intelligent technology.*
2018-11 Mahdi Sargolzaei (UVA), *Enabling Framework for Service-oriented Collaborative Networks.*
2018-12 Xixi Lu (TUE), *Using behavioral context in process mining.*
2018-13 Seyed Amin Tabatabaei (VUA), *Computing a Sustainable Future.*
2018-14 Bart Joosten (UVT), *Detecting Social Signals with Spatiotemporal Gabor Filters.*
2018-15 Naser Davarzani (UM), *Biomarker discovery in heart failure.*
2018-16 Jaebok Kim (UT), *Automatic recognition of engagement and emotion in a group of children.*
2018-17 Jianpeng Zhang (TUE), *On Graph Sample Clustering.*
2018-18 Henriette Nakad (UL), *De Notaris en Private Rechtspraak.*
2018-19 Minh Duc Pham (VUA), *Emergent relational schemas for RDF.*
2018-20 Manxia Liu (RUN), *Time and Bayesian Networks.*
2018-21 Aad Slootmaker (OUN), *EMERGO: a generic platform for authoring and playing scenario-based serious games.*
2018-22 Eric Fernandes de Mello Araujo (VUA), *Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks.*
2018-23 Kim Schouten (EUR), *Semantics-driven Aspect-Based Sentiment Analysis.*
2018-24 Jered Vroon (UT), *Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots.*
2018-25 Riste Gligorov (VUA), *Serious Games in Audio-Visual Collections.*
2018-26 Roelof Anne Jelle de Vries (UT), *Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology.*
2018-27 Maikel Leemans (TUE), *Hierarchical Process Mining for Scalable Software Analysis.*

2019

2019-01 Rob van Eijk (UL), *Web Privacy Measurement in Real-Time Bidding Systems. A Graph-Based Approach to RTB system classification.*

2019-02 Emmanuelle Beauxis-Aussalet (CWI), *Statistics and Visualizations for Assessing Class Size Uncertainty.*

2019-03 Eduardo González López de Murillas (TUE), *Process Mining on Databases: Extracting Event Data from Real-Life Data Sources*.